# Intelligent control for scalable video processing

# Intelligent Control for Scalable Video Processing

Cover design by Frans Schraven (CIS Visuals)

# Intelligent Control for Scalable Video Processing

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen door het College voor Promoties in het openbaar te verdedigen op woensdag 29 november 2006 om 14.00 uur

door

Clemens Christiaan Wüst

geboren te Leeuwarden

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. E.H.L. Aarts
en
prof.dr.ir. P.H.N. de With

Copromotor:
dr.ir. W.F.J. Verhaegh

CIP-DATA KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Wüst, Clemens Christiaan

Intelligent Control for Scalable Video Processing /
Clemens Christiaan Wüst. -
Eindhoven: Eindhoven University of Technology
Thesis Eindhoven.

ISBN-10: 90-74445-74-8
ISBN-13: 978-90-74445-74-0
EAN: 9789074445740

Subject headings: overload, soft real time, scalable video processing, Quality-of-Service, reinforcement learning

The work described in this thesis has been carried out at the Philips Research Laboratories in Eindhoven, the Netherlands, as part of the Philips Research programme.

# Contents

# 1

# Introduction

Video processing is the task of transforming an input video signal into an output signal, for example to improve the quality of the signal, or to adapt the signal to a different standard. This transformation is described by a video algorithm. At a high level, video processing can be seen as the task of processing a sequence of still pictures, called frames. In case insufficient resources have been assigned to process the most compute-intensive frames in time, a severe quality reduction occurs. Evidently, the resources may be enhanced to guarantee a better output quality, but alternatively the video algorithm may be modified.

This thesis describes a combination of scalable video processing and intelligent control that aims at optimizing the output signal within the framework of a limited amount of resources. In this first chapter the various concepts are being defined that are used throughout this thesis and the problem formulated.

## 1.1 Video processing

Video – the Latin word for 'I see' – is the technology of recording, processing, transmitting, and reconstructing motion pictures using analog or digital electronic signals. A video signal consists of a sequence of still pictures, which are called frames. Each frame consists of a number of horizontal lines, and each line consists of a number of pixels. An important characteristic of a video signal is the frame

rate, which is the number of still pictures per unit of time. The number of lines per frame, the number of pixels per line, and the frame rate of a video signal are determined by the applied video standard. For example, the PAL (Phase Alternating Line) color system prescribes a format of 576 lines per frame, 720 pixels per line, and a frame rate of 25 frames per second (fps).

Because digital video signals are becoming increasingly more important than analog video signals, we restrict ourselves to the former in this thesis. For ease of reading, the term 'digital' is usually omitted. In a digital video signal, frames are represented as two-dimensional arrays of pixels. A frame consisting of 576 lines and 720 pixels per line is said to have a picture resolution of 720 by 576 pixels. Per pixel usually two or four bytes of storage space are used to record color and intensity information. Figure 1.1 shows an example of a digital video signal.

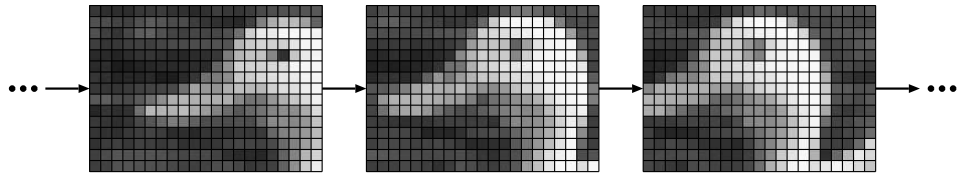

Figure 1.1. A sample of three successive frames from a digital video signal, showing a white duck in motion. Each frame has a picture resolution of 21 by 15 pixels.

Video processing is concerned with the transformation of an input video signal into an output video signal. This transformation is described by a so-called video algorithm. Two major classes of video algorithms are image enhancement algorithms and video format conversion algorithms [De Haan, 2000]. An image enhancement algorithm tries to improve the subjective picture quality of a video signal, i.e., the quality of the video signal as perceived by a viewer. This can be done, for example, by reducing noise or by sharpening edges in the pictures. A video format conversion algorithm transforms an input video signal into an output video signal having different properties, as, for example, a different frame rate, a different picture resolution, or a different number of bytes per pixel.

Another important class of video algorithms is the class of video compression and decompression algorithms. These algorithms are used to deal with the enormous storage requirements imposed by digital video. To give an impression of these storage requirements, consider a video signal with a picture resolution of 720 by 576 pixels. If frames are stored using two bytes per pixel, then each frame requires a storage space of 810 kB. At a frame rate of 25 fps this means that only four minutes of video fit onto a DVD (Digital Versatile Disc) with a capacity of 4.7 GB. To reduce its storage requirements, a digital video signal can be compressed, which

means that redundant information is removed from the signal. This can be done, for example, by storing a frame as a set of differences with respect to a nearly identical neighboring frame, instead of storing the frame as a two-dimensional array of pixels. Using compression, a two-hour movie can be stored on a DVD, or streamed over the internet using only limited bandwidth.

In a compressed video signal, frames are generally not stored as two-dimensional arrays of pixels. Hence, a compressed video signal cannot be used directly for display. For this a decompression algorithm is needed, which restores each frame of a compressed video signal as a two-dimensional array of pixels. The successive frames of a compressed video signal are usually decompressed on the fly, right before they are needed for display.

Video compression is usually lossy, which means that less important information is deliberately not stored in the compressed signal, i.e., some data is lost. As a result, the original video signal cannot be reconstructed exactly from the compressed signal, but only closely approximated. In contrast, if compression is lossless, then no data is lost in the compression step, and the original video signal can be fully reconstructed without any quality reduction. The main advantage of lossy compression over lossless compression is that very high compression rates can be obtained, where the compression rate is defined as the ratio between the size of the original video signal and the size of the compressed signal.

### 1.1.1 MPEG-2

The Motion Picture Experts Group (MPEG), a consortium in which industry and academia have joined forces, has developed various compression standards for audio and video. MPEG compression is also called MPEG encoding, and MPEG decompression is also called MPEG decoding.

MPEG-2 [Haskell *et al.*, 1997; Mitchell *et al.*, 1997] is a lossy compression standard for video that is used, amongst others, for digital television and DVDs. For MPEG-2, the order of the frames in an encoded video signal can differ from the order in which the decoded frames have to be displayed. The order of the frames in an encoded video signal, which corresponds to the order in which the frames have to be decoded, is called the decoding order or transmission order of frames. The order in which the decoded frames have to be displayed is called the display order of frames. The latter corresponds to the original order of the frames, i.e., the order of the frames before encoding. Using MPEG-2, frames are encoded as I-, P-, or B-frames. An I-frame (or intra frame) is self-contained, which means that it can be decoded without any additional information. The P- and B-frames are not self-contained, and can only be decoded using reference frames. A P-frame (or predicted frame) uses one reference frame, which is given by the most recently decoded I- or P-frame. This reference frame appears earlier than the P-frame in

display order. A B-frame (or bi-directionally predicted frame) uses two reference frames, which are given by the two most recently decoded I- or P-frames. For a B-frame, one reference frame appears earlier than the B-frame in display order, and one reference frame appears later than the B-frame in display order. Hence, if B-frames are used, then there is a difference between the decoding order and display order of frames. Frame reordering in the encoding and decoding steps is at the cost of using additional frame buffers, i.e., memory units which are used for the temporal storage of encoded or decoded frames. Figure 1.2 illustrates the difference between the decoding order and the display order of frames.

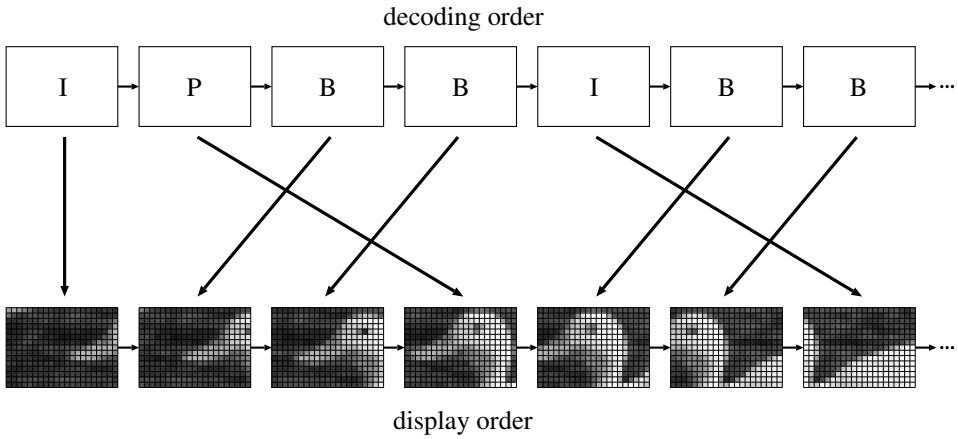


Figure 1.2. An example showing the difference between the decoding order and the display order of frames for MPEG-2. The used GOP structure is IBBPBB.

In display order, the decoded I-, P-, and B-frames usually appear in a repeating pattern called a group of pictures (GOP). A GOP is formed by an I-frame together with all P- and B-frames before the next I-frame. In Figure 1.2, the used GOP structure is IBBPBB.

## 1.2 Embedded video processing in software

Multimedia consumer terminals (MCTs) are consumer electronics devices that are connected to a video broadcast network or a communication network and that provide multimedia experiences to users. Examples of MCTs are TV sets and the boxes that are used to receive digital television from satellite or cable, the so-called set-top boxes. Although MCTs today are mainly autonomously operating devices, they are expected to evolve to cooperating devices in in-home digital networks [Chen, 1997], and beyond that to elements in an ambient intelligent environment [Aarts *et al.*, 2002]. Ambient intelligence refers to a vision where people

are surrounded by numerous intelligent devices, which are embedded into everyday objects. These devices cooperate to provide information, communication, and entertainment experiences to the user.

One of the main tasks of an MCT is to perform high-quality audio and video processing. Traditionally, audio and video processing in MCTs is performed by dedicated hardware components. For example, Figure 1.3 shows the system architecture of a high-end TV, consisting of various hardware components. There is an ongoing trend towards programmable (i.e., software based) MCTs [Bril *et al.*, 2001a; Bril *et al.*, 2001b; Isović and Fohler, 2004]. Rather than requiring additional, dedicated, single-function hardware components for each additional feature, a programmable MCT enables additional features by sharing programmable components. Another advantage of a programmable MCT is that it can be configured and upgraded after production, for example to enhance the functionality of the device, or to adapt the device to new standards.

A programmable MCT is an example of an embedded system, a special purpose computer system which is fully embedded into a device. The core of an embedded system is formed by one or more programmable processors, which are used to run various software tasks. An instance of such a processor is the Philips TriMedia VLIW processor [Rathnam and Slavenburg, 1996], which is optimized for audio and video processing. In an embedded system, resources, such as processor cycles and memory, are shared by the various software tasks to achieve cost effectiveness.

Video processing in software is often characterized by highly fluctuating, content-dependent processing times of frames [Baiceanu *et al.*, 1996]. This is especially true for video algorithms that contain motion estimation, such as natural motion [Braspenning *et al.*, 2002] or MPEG encoding [Mietens *et al.*, 2004], or motion compensation, such as MPEG decoding [Lan *et al.*, 2001; Peng, 2001; Zhong *et al.*, 2002]. There is often a considerable gap between the worst-case and average-case processing times of frames. For example, Figure 1.4 shows the processing times (or *load*) for decoding a sequence of 700 MPEG-2 frames on a TriMedia TM1300 180 MHz processor. The sequence of 700 frames was taken from a larger sequence of in total 136,560 frames. The best-case, average-case, and worst-case processing times of the 700 frames are 9.2 ms, 22.6 ms, and 36.6 ms, respectively, and for the entire sequence of 136,560 frames they are 8.8 ms, 27.2 ms, and 71.4 ms, respectively.

Video processing in dedicated hardware is generally designed to handle worst-case input in time, which is guaranteed to result in high-quality output. If video processing is done in software instead, the same stable, high-quality output can be obtained. To avoid a quality reduction of the output signal, each frame should be processed in time. This could be achieved by assigning sufficient resources to a video processing task, based on its worst-case needs, but this is not cost effec-

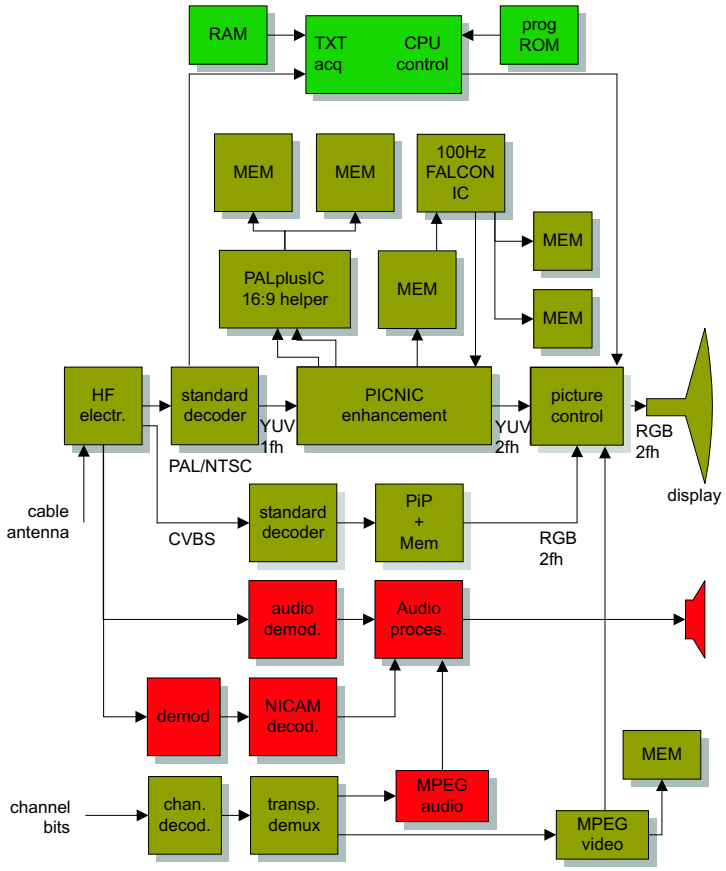


Figure 1.3. The system architecture of a high-end TV, consisting of various hardware components (picture by courtesy of Egbert G.T. Jaspers).

tive. To relax the worst-case needs, video algorithms have been made scalable. A scalable video algorithm (SVA) [Hentschel *et al.*, 2001a; Hentschel *et al.*, 2001b] can process video frames at different quality levels. Each quality level provides a particular trade-off between the time spent on processing a frame, and the resulting picture quality. Examples of scalable algorithms from the video domain are scalable sharpness enhancement [Hentschel *et al.*, 2001a] and scalable MPEG decoding [Lan *et al.*, 2001; Peng, 2001; Zhong *et al.*, 2002]. An example from the 3D graphics domain is scalable graceful degradation [Lafruit *et al.*, 2000].

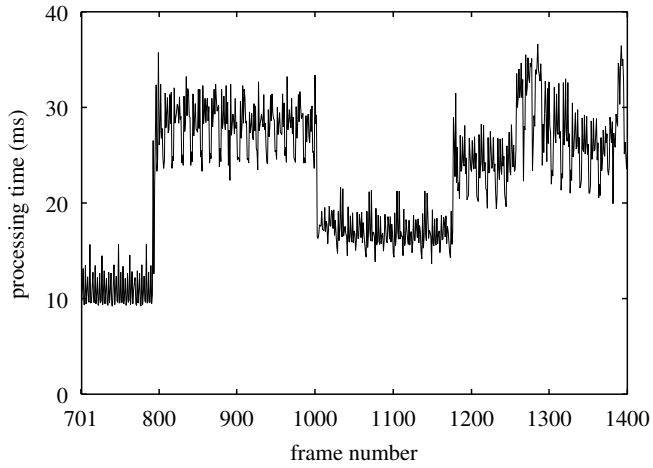


Figure 1.4. The load of decoding a sequence of 700 MPEG-2 frames, taken from the DVD 'Pet Shop Boys – Somewhere'.

## 1.3 Real-time systems

To enable timely delivery of output, software video processing is usually done using a real-time system. Real-time systems [Buttazzo, 1997] are computer systems that must react within precise time bounds to events from their environment. The correct behavior of a real-time system therefore does not only depend on the correctness of produced results, but it also depends on the response times of the system: a result that becomes available too early or too late could be useless or even dangerous. A real-time system is not necessarily a fast system. Although timing requirements can involve both lower and upper bounds on the response times, they are usually only expressed as upper bounds on the response times. These upper bound are called deadlines. In contrast to a real-time system, a general purpose computer system does not provide any guarantees on the times at which results become available. Whereas the objective of a general purpose computer system is typically to optimize average-case response times, the objective of a real-time system is to guarantee that each individual timing requirement is met.

A distinction can be made between hard real-time systems and soft real-time systems. A hard real-time system provides guarantees that system deadlines are met predictably. Hard real-time systems are often applied to control physical hardware, where a missed deadline may cause failure or damage. For example, a hard real-time system can be used to control the times at which fuel is injected into the cilinders of a car's engine. Other application areas of hard real-time systems in-

clude, amongst others, flight control systems, military systems, robotics, and plant control systems.

A soft real-time system guarantees only that deadlines are met generally, but occasional deadline misses are allowed. Soft real-time systems can be applied when meeting deadlines is desirable for the reason of performance, but missing a deadline does not lead to critical failure of the system. This is for example applicable to video processing, where a processed frame that becomes available too late does not lead to failure of the system, but reduces the quality of the output signal.

Real-time systems, like general purpose computer systems, are often based on multitasking, which means that the processor of the system is time shared by various software tasks. A dedicated scheduler switches repeatedly between the execution of the various tasks, which from a distance gives the impression that all tasks run simultaneously. Each task may be viewed as a sequence of jobs that are activated by events, such as the arrival of new input from the environment. For example, upon receiving a video frame one or more jobs may be started to process the frame. Traditionally, the scheduling of real-time tasks is based on a-priori knowledge of the worst-case execution times of jobs.

The jobs of a task can be activated aperiodically or periodically. Accordingly, there are aperiodic and periodic tasks in a real-time system. An example of a periodic task is video processing. For video processing, the successive frames to be processed become available periodically at the input of the system, and processed frames are also needed periodically at the output of the system. For scheduling periodic tasks, an algorithm named fixed-priority preemptive scheduling [Klein *et al.*, 1993; Audsley *et al.*, 1995] is the de facto standard. This algorithm always selects the task with the highest priority for execution. If a task is executing a job and in the meanwhile a job for a higher priority task becomes available, then the execution of the lower priority task is suspended (preempted) in favor of the higher priority task. The problem of scheduling a set of periodic tasks was first studied by Liu and Layland [1973].

## 1.4 QoS RM framework

The work described in this thesis is part of a larger effort, which defines and builds a framework for Quality of Service resource management for high-quality video, named QoS RM [Bril *et al.*, 2001b; Hentschel *et al.*, 2001b; Otero Pérez *et al.*, 2003; Wüst *et al.*, 2004a]. Quality of Service (QoS) is defined as "the collective effect of service performance which determine the degree of satisfaction of a user of the service" [ITU-T, 1994]. The notion of QoS can be used to trade-off different aspects of user-perceived quality in a single measure. The QoS RM framework consists of a multi-layer control hierarchy and a reservation-based resource

manager, and it runs on top of a real-time operating system. Figure 1.5 gives a simplified view of the framework.

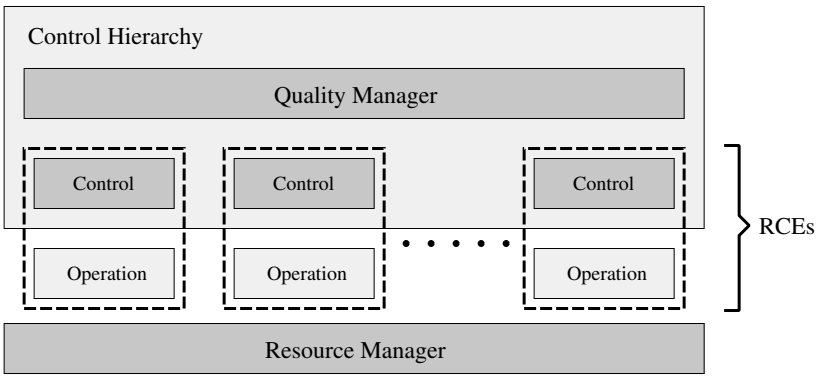


Figure 1.5. A simplified view of the QoS RM framework.

In the framework, the resource manager addresses robustness of the system by assigning periodic resource budgets to so-called resource consuming entities (RCEs). An RCE is a cluster of one or more cooperating tasks. The resource budget of an RCE is enforced by the resource manager, to ensure that parts of the budget cannot be taken away by other RCEs in the system. Guaranteed resource budgets are recognized as a basis for QoS resource management [Mercer *et al.*, 1994; Rajkumar *et al.*, 1998; Feng and Mok, 2002]. The resource manager may re-distribute unused parts of an RCE's budget over the other RCEs in the system as so-called gain time.

An RCE is scalable if it contains a scalable video processing task. A scalable RCE can run at different quality levels. Each quality level provides a particular trade-off between the output quality and the resource needs of the RCE. For each scalable RCE one or more coarse-grain quality levels are defined. Each coarse-grain quality level is defined as a cluster of quality levels. The quality levels that belong to the same coarse-grain quality level all provide fine-grain variations on a particular coarse-grain trade-off between the output quality and the resource needs of the RCE. A quality level can belong to multiple coarse-grain quality levels. An RCE that is not scalable can also be considered scalable, having only a single quality level.

An RCE consists of an operational part and a control part. The operational part performs the actual processing. From the outside of an RCE, the coarse-grain quality level can be set. Given the coarse-grain quality level, the control part of the RCE dynamically fine-tunes the applied quality level during processing, to maximize a local QoS measure for the RCE.

In the control hierarchy, a quality manager is responsible for global QoS optimization using a system-wide notion of utility [Prasad *et al.*, 2003]. The quality manager determines a coarse-grain quality level and a matching resource budget for each active RCE, taking the relative importance of RCEs into account, using a model similar to the one described by Lee *et al.* [1999]. The set of resource budgets and periods for the various RCEs must be schedulable on the processor. To perform the global QoS optimization, for each RCE the quality manager maintains a mapping from coarse-grain quality levels to estimated resource needs. This mapping is dynamically updated using statistics provided by the resource manager.

In this thesis we focus on local QoS optimization for an RCE consisting of a single scalable video processing task. The assumption has been made that insufficient processing-time budget is assigned to the RCE to meet the deadlines of the most compute-intensive frames. Our results are also applicable outside the context of the QoS RM framework, for example in case of a scalable video processing task running on a private processor that does not have sufficient capacity to support the worst-case workload of the task.

## 1.5 Informal problem statement

In the highly competitive market for digital consumer electronics, programmable MCTs are subject to a low bill of material. To be cost effective, it is required that a software video processing task exhibits a high average resource utilization. However, this requirement leads to a dilemma. On the one hand, to meet the deadlines of the successive frames to be processed, we have to assign a periodic processing-time budget to the task based on the task's worst-case needs for processing video frames. On the other hand, as has been shown in Figure 1.4, there is often a large gap between the worst-case and average-case processing times of frames. This means that a worst-case budget is not cost effective.

We address this dilemma as follows. First, the video processing task is considered to be a soft real-time task. For each frame to be processed there is a deadline, which is given by the time at which the processed frame is needed for output. The deadlines of successive frames are strictly periodic in time. In every deadline period we assume that a processing-time budget is assigned to the task that is smaller than the task's worst-case needs for processing video frames. This periodic budget can be viewed as a private processor, running at a fraction of the speed of the actual processor.

Second, we allow the task to work ahead by means of asynchronous processing [Sha *et al.*, 1986]. An asynchronous video processing task starts processing a new frame immediately upon completion of the previous one, without first having to wait for the deadline of the completed frame, provided that a new frame is

available. If no new frame is available, then the task will block. Asynchronous processing reduces the risk of missing deadlines, because the unused part of the budget for an easy frame can be used as a surplus to the budget for the next frame to be processed. The extent to which working ahead can be applied is determined by latency and buffer constraints [Isović *et al.*, 2003].

Finally, we assume that the task makes use of an SVA, i.e., we assume that the task is scalable. Hence, frames can be processed at different quality levels. The higher the selected quality level for a frame, the higher is the resulting picture quality, but also the more processing time is needed. By selecting the right quality levels for frames, deadline misses may be prevented.

Informally, the problem at issue can be stated as follows. We consider a soft real-time scalable video processing task to which is assigned a lower than worst-case processing-time budget. The task can process each frame at different quality levels. Furthermore, the task can work ahead by means of asynchronous processing. For a given sequence of video frames to be processed, which is not known upfront, we consider the problem of selecting the quality level for each frame. The objective that we try to optimize reflects the user-perceived quality, and is determined by a combination of three aspects. First, we consider the quality level at which frames are processed. Applying a higher quality level results in a better picture quality. Second, we consider deadline misses, because deadline misses may result in a severe quality reduction of the output signal. Third, we also consider changes in the applied quality level between successive frames, because (bigger) changes in the quality level may result in (better) perceivable artifacts.

## 1.6 Related work

In the computer science literature various approaches can be found to optimize the output quality of a real-time video processing task with limited resources, the subject of this thesis. First, in Section 1.6.1 we focus on approaches that are applicable to non-scalable video, based on skipping frames. Next, in Section 1.6.2 we focus on approaches that are applicable to scalable video.

Our approach, in which we assume a video processing task with a fixed resource budget, is complemented by a technique called conditionally guaranteed budgets, in which the resource budget of a task can be varied, depending on its actual needs. We discuss this technique in Section 1.6.3.

### 1.6.1 Optimizing approaches for non-scalable video

Hamann *et al.* [2001] model the MPEG decoding task according to the imprecise computation model [Zhao *et al.*, 1995]. The MPEG decoding task is modeled as a periodic task which decodes one group of pictures every period. The decoding

of a group of pictures consists of a mandatory part and an optional part. The mandatory part consists of decoding the I- and P-frames, and the optional part consists of decoding B-frames. Their QoS measure is the percentage of optional parts that meet their deadlines. The disadvantage of the approach is the high latency it requires, which is unacceptable for MCTs.

Isović and Fohler [2004] formulate a real-time scheduling problem for quality-aware frame skipping during MPEG decoding. Given that not all frames can be decoded in time, due to limited resources, only the frames that provide the best picture quality are selected for processing. A frame is processed only if it can be guaranteed that the frame's deadline will be met. Our approach is also based on skipping frames, in case a deadline is missed, as will be discussed in Chapter 2. The method of Isović and Fohler could be used to make a dynamic decision about *which* frames should be skipped.

### 1.6.2 Optimizing approaches for scalable video

Lee *et al.* [1999] present a QoS resource management framework which distributes resources over tasks in a resource-constrained system. In their setup, tasks can be scalable with respect to one or more resources. For each scalable task a finite set of quality levels is defined. A quality level is given by a particular setting for each scalable resource of the task. The resource needs of a scalable task are assumed to be fixed for each quality level. Their objective is to maximize the overall user-perceived output quality of the system, given the availability of only a limited amount of resources. The notion of user-perceived quality is modeled by means of utility functions, and the optimization problem is formulated as a knapsack problem. Whereas the approach of Lee *et al.* addresses the distribution of resources over the various tasks, which is the task of the quality manager in the QoS RM framework, we consider the problem of how each single task makes optimal use of its assigned resources, taking care of fluctuations in the task's workload. In that sense, the two approaches complement each other. Moreover, our approach is more dynamic, because the approach of Lee *et al.* changes the quality level of a task only upon a configuration change of the system.

Lan, Chen and Zhong [2001] describe a method to regulate the varying computation load of a scalable MPEG decoder, which is operated synchronously. Before decoding an MPEG frame, the required computational resources are estimated, and next the decoding is scaled such that it will not exceed a target computation constraint. In contrast to our approach, they only optimize the output quality of individual frames, and not the overall perceived quality over a sequence of frames.

In our approach we implicitly assume that the input signal of the video processing task is of fixed quality. In contrast, Jarnikov, Van der Stok and Wüst [2004] assume that the quality of the input signal can fluctuate over time, in the context of

MPEG decoding. In their approach, each MPEG encoded frame is partitioned into a base layer and a fixed number of enhancement layers. Decoding only the base layer results in a low picture quality, and successive enhancement layers can be decoded incrementally to obtain a better picture quality. The decoder can process frames at different quality levels, where the lowest quality level corresponds to decoding only the base layer, the next quality level corresponds to decoding the base layer and the first enhancement layer, etcetera. Due to loss of data in a wireless network, it is assumed that that the maximum quality level at which frames can be decoded at the receiving side of the network fluctuates dynamically. To optimize the user-perceived quality of the output signal, a QoS controller is used which selects a quality level for each frame to be decoded. The used model extends the model described in this thesis.

Combaz *et al*. [2005a, 2005b] propose a method for QoS control of a scalable video processing task, which was clearly inspired by our work. In their model, they use the same QoS parameters as we do, viz. minimization of deadline misses, maximization of the assigned processing-time budget, and smoothness of quality levels. Based on average-case and worst-case execution times for various scalable and non-scalable subtasks of the task, a QoS controller is constructed. This controller can dynamically adapt the quality level of the various scalable subtasks during the processing of a frame, to optimize the QoS measure for the frame. One main difference with our work is that we do not change the applied quality level during the processing of a frame, to prevent quality fluctuations within frames. A second main difference is that we optimize our QoS measure for an entire sequence of frames to be processed, and not for frames individually.

### 1.6.3 Conditionally guaranteed budgets

Bril [2004] presents a technique called conditionally guaranteed budgets (CGBs) which complements our approach. Whereas we assume a task with a fixed resource budget, CGBs refine existing resource budgets by exploiting the notion of relative importance of tasks (or applications). CGBs require a dedicated mechanism at the level of a resource manager to facilitate an instantaneous budget configuration change. This mechanism allows a more important task to instantaneously receive an anticipated amount of additional resources upon request, at the cost of a predetermined set of less important tasks. Hence, CGBs allow a more important task to maintain an acceptable perceived quality upon a structural load increase that would otherwise result in a severe quality reduction. Our approach can be used in combination with CGBs, to control the quality level of a scalable task at times when its resource budget is fixed.

## 1.7 Thesis outline

In this thesis we present various mathematical strategies that can be used to control the quality level at which a scalable video processing task processes video frames. The goal of the strategies is to maximize the user-perceived quality of the task's output signal, within the framework of a limited amount of resources. First, in Chapter 2 we present a model for real-time video processing in software, and we present the QoS control problem. Next, in Chapter 3 we provide a basic introduction to reinforcement learning, which is used in the subsequent chapters. In Chapter 4 we model the QoS control problem as a Markov decision process. Solving this model results in our first control strategy, which is based on off-line optimization using pre-determined processing-time statistics. In Chapter 5 we present a variant of this strategy, which explicitly takes care of dependencies in the processing times of successive frames. In Chapter 6 we present our last strategy, which hardly requires any prior knowledge, but that learns how to behave optimally from experienced processing times. In Chapters 4 to 6 we validate the presented control strategies by means of simulation experiments, based on processing-time statistics of an MPEG-2 decoder. Finally, in Chapter 7 we briefly discuss user-perception experiments, we come back to various simplifying assumptions that were made in our work, and we summarize the main results.

# 2

# Problem modeling and formulation

In this chapter we present a processing model for a scalable video processing task with soft real-time constraints, and we formulate the problem that is studied in this thesis. The basic processing model is presented in Section 2.1. Next, in Section 2.2 we discuss how input queue overflows and output queue underflows are handled in the model. In Section 2.3 we assign a periodic processing-time budget to the task, and we introduce a measure called progress. Given the processing model, in Section 2.4 we present the objective of the work described in this thesis, and in Section 2.5 we formulate the QoS control problem.

## 2.1 Basic processing model

In this section we present the basic processing model, as depicted in Figure 2.1. We consider a single scalable video algorithm (SVA) that is running in a multitasking system. For convenience, we use the terms task, scalable task, scalable video processing task, and SVA interchangeably. The SVA has to process an indefinite sequence of video frames. The successive frames to be processed are numbered $1, 2, 3, \ldots$. Throughout the thesis, we use integers $f$ and $g$ to indicate frame numbers. The SVA fetches frames to be processed from an input queue, and it writes processed frames to an output queue. The input queue and output queue each consist of a finite number of frame buffers. A frame buffer, or buffer for short, can hold

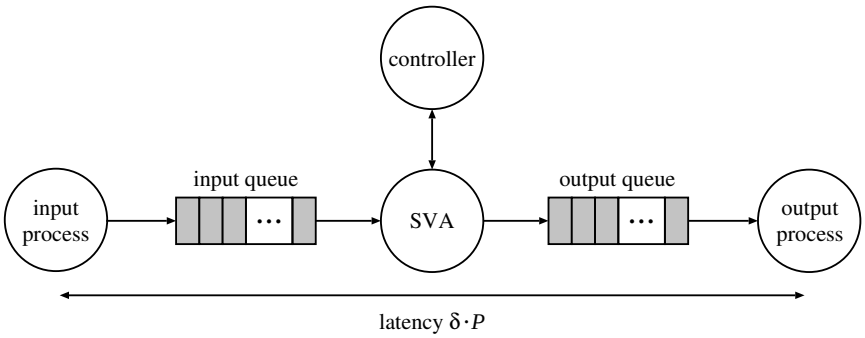


Figure 2.1. The basic processing model.

one unprocessed or processed frame. The memory size of a buffer can vary, depending on the size of the frame it holds. Buffers from the input queue and output queue are called input buffers and output buffers, respectively.

The SVA can process frames at different quality levels, given by a finite set $Q = \{q_1, \ldots, q_{n_Q}\}$. We call quality level $q_i$ higher than quality level $q_j$ if $i > j$. The higher the applied quality level, the more effort is spent by the SVA on processing a frame, which results in a better picture quality. In general, applying a higher quality level results in a higher processing time of a frame. However, this is no strict rule, because the processing time of a frame may be influenced by effects such as cache misses, bus contention, and task switching.

Each frame can be processed at a different quality level. We assume that a frame is always processed at a single quality level, i.e., the used quality level cannot be changed while the frame is being processed. The quality level at which frame $f$ is processed is denoted by $q(f)$. The quality levels $q(1), q(2), \ldots$ for the successively processed frames are chosen by a controller. Before processing frame $f$, the SVA first calls the controller to obtain quality level $q(f)$. We assume that the time needed by the controller to select the quality level for a frame is part of the frame's processing time.

Frames can vary in type. For example, the MPEG-2 standard [Haskell *et al.*, 1997; Mitchell *et al.*, 1997] differentiates between I-, P-, and B-frames. Because frames of different types may have to be processed differently, the processing time of a frame can depend on the frame type. We assume a finite set of frame types $\Phi = \{\phi_1, \ldots, \phi_{n_\Phi}\}$. The type of frame $f$ is denoted by $\phi(f)$. If we choose to not differentiate between different frame types, then we can model this as all frames having the same type $\phi_1$.

An input process, for example a digital video tuner, periodically inserts an unprocessed frame into the input queue, with a time period $P > 0$. The input process

inserts frames in the order in which they have to be processed. An input buffer has the status filled if it contains a frame that is waiting to be processed by the SVA, or if it contains a frame that is currently being processed by the SVA, and the status empty otherwise. Normally, the input process can only insert a frame into an empty input buffer. If no empty input buffer is available, then an input queue overflow occurs. Clearly, input queue overflows should be avoided.

An output process, for example a video renderer, periodically consumes a frame from the output queue, also with period $P$. Hence, we assume that the input frame rate and output frame rate are the same[1]. Consuming a frame from the output queue means that the frame is read, but not removed from the queue. We assume that the output process consumes frames in the order in which they have been processed. Hence, we assume that the input order and output order of frames are the same[1]. An output buffer has the status filled if it contains a frame that is waiting to be consumed by the output process, and the status empty otherwise. If the output process is unable to consume the frame that it needs, i.e., if no filled output buffer is available, then an output queue underflow occurs. Clearly, output queue underflows should be avoided. In Section 2.2 we discuss how input queue overflows and output queue underflows are handled.

The input process and the output process are synchronized with a fixed latency $\delta \cdot P$. This means that if frame $f$ enters the input queue at time $e(f) = e(0) + f \cdot P$, where $e(0)$ is an offset, then it is consumed from the output queue at time $d(f) = e(f) + \delta \cdot P$. We call $\delta$ the periodic latency, $e(f)$ the entry time of frame $f$, and $d(f)$ the deadline of frame $f$. We assume that $\delta$ is an integer number, which implies that the output process consumes a frame from the output queue at the very same moment as the input process inserts a frame into the input queue[1]. On average, the SVA has to process one frame per period $P$. We assume a periodic latency $\delta > 1$, which allows the SVA to process asynchronously: after processing frame $f$, the SVA can immediately start to process frame $f+1$, without first having to wait for deadline $d(f)$, provided that frame $f+1$ is available for processing. For a given periodic latency $\delta > 1$, the SVA can work ahead by at most $\delta - 1$ periods $P$. We apply asynchronous processing to even out the fluctuating processing times of frames in time.

Upon arrival of the first frame in the input queue, at $e(1)$, the SVA starts to process as soon as it is scheduled on the processor. The time at which the SVA starts to process frame $f$, the start point of frame $f$, is denoted by $\alpha(f)$. The time at which the SVA finishes processing frame $f$, the milestone of frame $f$, is denoted by $\omega(f)$. At milestone $\omega(f)$, the SVA either completes processing frame $f$, or it aborts processing frame $f$. The processing of a frame can be aborted in case of an

[1] We come back to this assumption in Chapter 7.

input queue overflow or output queue underflow, as will be discussed in Section 2.2. The processing time of frame $f$ is denoted by $\mu(f)$. We assume that frames have nonzero processing times. Note that $\mu(f) \leq \omega(f) - \alpha(f)$, because in the time interval $[\alpha(f), \omega(f)]$ the SVA can be interrupted by other tasks with a higher priority.

The SVA can start to process a frame if two conditions are met. First, the unprocessed frame should be present in the input queue, and second, there should be an empty output buffer. At the start point of a frame, the SVA first claims exclusive access to both the input buffer containing the unprocessed frame, and one of the empty output buffers. Next, the SVA calls the controller to obtain the quality level at which the frame will be processed. Upon receiving the quality level, the SVA starts to process the frame. At the milestone of the frame, the two claimed buffers are released. The released input buffer gets the status empty, and the released output buffer gets the status filled. At the milestone, if the processing conditions are met for the next frame to be processed, then a start point immediately follows, assuming that the SVA can still use the processor. Otherwise, the SVA is blocked until the processing conditions for the next frame are met, upon which the start point follows as soon as the SVA is scheduled on the processor.

**Example 2.1.** Figure 2.2 shows an example timeline, illustrating the SVA's processing behavior for a periodic latency $\delta = 2$. In the figure, time is indicated by the periodic entry times and deadlines of frames, the start points and milestones of frames are indicated by down-pointing arrows, and the processing of frames is indicated by gray bars. Disregarding quality levels and frame types, the processing times of frames 1 to 5 are given by $\mu(1) = 1.75P$, $\mu(2) = 0.5P$, $\mu(3) = 0.5P$, $\mu(4) = 0.75P$, and $\mu(5) = 1.5P$. For simplicity, we assume that the SVA runs on a private processor.

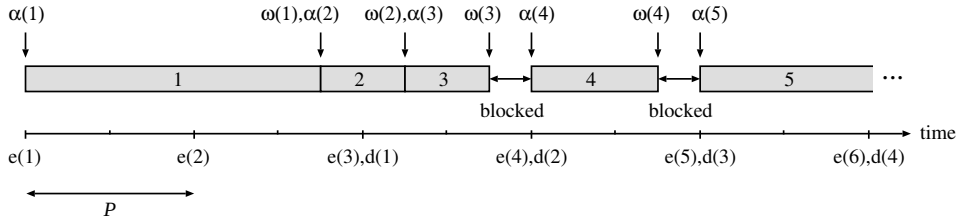


Figure 2.2. An example timeline, illustrating an SVA's processing behavior for a periodic latency $\delta = 2$.

Upon arrival of frame 1 in the input queue, at $e(1)$, the SVA start to process immediately. Frames 1, 2, and 3 are processed in one batch. At $\omega(3)$, the SVA becomes blocked because frame 4 is not yet present in the input queue. At $e(4)$,

the frame enters the input queue, and the SVA continues to process. Similarly, the SVA is blocked from $\omega(4)$ until $\alpha(5)$. □

For each frame $f$ there is a time interval $[e(f), e(f) + \delta P]$ in which the frame has to be processed by the SVA. However, if the number of input buffers or the number of output buffers is chosen smaller than $\delta$, then this time interval is shortened. This can be seen as follows. Let integers $i$ and $j$ denote the number of input buffers and output buffers, respectively. Without loss of generality, assume that $f > j$. To process frame $f$, an empty output buffer is needed. An empty output buffer is available no sooner than deadline $d(f - j)$, when frame $f - j$ is consumed from the output queue, because $j$ output buffers are needed to store frames $f - j, \ldots, f - 1$. Moreover, to prevent the input queue from overflowing, frame $f$ should be completed by $e(f + i)$, because $i$ input buffers are needed to store frames $f, \ldots, f + i - 1$. Hence, frame $f$ has to be processed in time interval $[\max\{e(f), e(f) + (\delta - j)P\}, \min\{e(f) + iP, e(f) + \delta P\}]$. From this, we can see that it is not useful to choose the number of input buffers or the number of output buffers larger than $\delta$. To provide the SVA maximum freedom to work ahead, we assume that the number of input buffers and the number of output buffers are both equal to $\delta$.

In our model we assume that the entry times and deadlines of frames are strictly periodic. In practice, however, there can be jitter in the entry times and deadlines of frames. This means that the input process can try to insert frame $f$ into the input queue a little earlier or later than entry time $e(f)$ in our model, and that the output process can try to consume frame $f$ from the output queue a little earlier or later than deadline $d(f)$ in our model. The problem of jitter can be dealt with as follows. If the input process tries to insert frame $f$ a little earlier than entry time $e(f)$, then it can be the case that no empty input buffer is available for storing frame $f$, which would normally be available. We can compensate for this time difference by adding one extra buffer to the input queue. If the input process tries to insert frame $f$ a little later than entry time $e(f)$, then the SVA may be blocked from $e(f)$ until the time at which frame $f$ arrives in the input queue. However, this blocking will occur only if the SVA is far ahead of its deadlines. If the output process tries to consume frame $f$ a little earlier than deadline $d(f)$, then we have to associate a possible deadline miss with the time at which the output process needs the frame, and not with $d(f)$. If the output process tries to consume frame $f$ a little later than deadline $d(f)$, then the SVA may be blocked from $d(f)$ until the time at which the frame is consumed by the output process, in case no empty output buffer is available. Again, this blocking will occur only if the SVA is far ahead of its deadlines. The blocking can be avoided by adding an extra buffer to the output queue.

The process of inserting a frame into the input queue may take some time. This is not a problem to our model, because the processing of a frame can already be started if the frame is only partially available in the input queue. If the SVA is faster than the input process, then the SVA may get blocked during processing until more input data becomes available. Also the consumption of a frame from the output queue may take some time. As with jitter in the consumption times of frames, this may result in the SVA getting blocked. Again, blocking can be avoided by adding an extra buffer to the output queue.

## 2.2 Overflow and underflow handling

In this section we discuss how input queue overflows and output queue underflows are handled. An output queue underflow corresponds to a deadline miss. First, we present two approaches to handle a deadline miss. Next, we show that input queue overflows are automatically taken care of by the handling of deadline misses.

Normally, if the SVA is processing a frame $f$, and if the frame is not completed by deadline $d(f)$, then the deadline is missed. We consider two approaches to handle a deadline miss. The first approach is to abort processing frame $f$ at deadline $d(f)$, which implies $\omega(f) = d(f)$. At $d(f)$, the two buffers that were claimed by the SVA for processing frame $f$ are released. The released input buffer is used immediately to insert frame $f+\delta$ arriving from the input process. Hence, the status of this buffer remains filled. The released output buffer gets the status empty. Because frame $f+1$ is present in the input queue and an empty output buffer is available, the processing conditions for frame $f+1$ are met. As a result, the milestone of frame $f$ is immediately followed by the start point of frame $f+1$, assuming that the SVA is still scheduled on the processor. We call this approach to handle a deadline miss the aborting approach.

A drawback of the aborting approach is that the processing time which has been spent on an aborted frame $f$ is wasted. Using possibly little additional processing time, frame $f$ could be completed. If it is acceptable that frame $f$ is consumed at a later deadline, at the cost of a temporal inconsistency in the end-to-end latency, then the deadline miss can also be handled as follows. Instead of aborting frame $f$ at deadline $d(f)$, processing is continued, and a new deadline $d(f+1)$ is assigned to the frame. To restore the end-to-end latency, frame $f+1$ is skipped. At $d(f)$, the input buffer which contains frame $f+1$ is overwritten with frame $f+\delta$ arriving from the input process. Hence, the status of this buffer remains filled. If deadline $d(f+1)$ is missed too, then a new deadline $d(f+2)$ is assigned to frame $f$, and frame $f+2$ is skipped. This procedure is repeated until frame $f$ is finally completed. We call this approach to handle a deadline miss the skipping approach. In contrast to the aborting approach, the skipping approach preserves the work that

has been spent on a frame when its deadline is missed. Note that the skipping approach can result in multiple deadline misses per frame, whereas the aborting approach can result in at most one deadline miss per frame.

**Example 2.2.** Figure 2.3 shows two example timelines, which illustrate how a deadline miss is handled by the SVA, for the aborting approach (fig. A) and the skipping approach (fig. B). In both timelines we assume a periodic latency $\delta = 2$. Disregarding quality levels and frame types, the processing times of frames 1 to 5 are given by $\mu(1) = 1.75P$, $\mu(2) = 1.5P$, $\mu(3) = 0.5P$, $\mu(4) = 1.25P$, and $\mu(5) = P$. For simplicity, we assume that the SVA runs on a private processor.

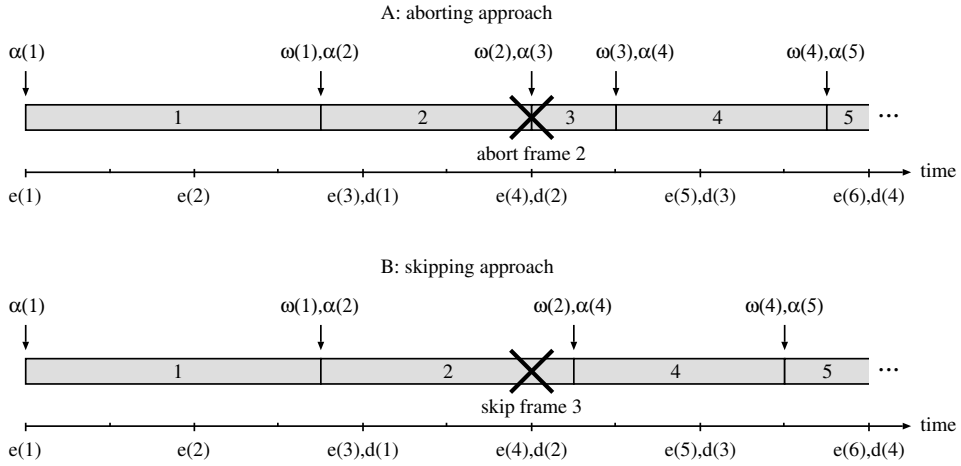


Figure 2.3. Example timelines, illustrating how a deadline miss is handled by the SVA, for the aborting approach (fig. A) and the skipping approach (fig. B).

In both timelines deadline $d(2)$ is missed. Applying the aborting approach, frame 2 is aborted, and the SVA immediately continues to process frame 3. Applying the skipping approach, frame 2 is completed, and a new deadline $d(3)$ is assigned to the frame. Frame 3 is skipped at $d(2)$. □

Applying the skipping approach, frame $f+1$ is skipped if a new deadline $d(f+1)$ is assigned to frame $f$. In general, also a later frame may be skipped. For example, the SVA may skip frame $f+3$ instead of frame $f+1$. This involves that deadlines $d(f+2)$ and $d(f+3)$ are assigned to frames $f+1$ and $f+2$, respectively. To avoid a pile-up of frames in the input queue, the SVA can only skip a frame that is already present in the input queue, or the frame arriving from the input process. In general, the frame that is skipped should be chosen carefully. For example, in MPEG-2 decoding, B-frames can safely be skipped, whereas skipping an I-frame generally stalls the stream [Isović *et al.*, 2003].

Clearly, not only the SVA, but also the output process has to handle each deadline miss. Upon a deadline miss, we assume that the output process performs error concealment. For example, if the output process is a video renderer, then it may redisplay the most recently displayed frame. The renderer can consume this frame once more from the output queue, provided that the SVA has not overwritten the corresponding buffer. Because $\delta > 1$, this can be achieved by letting the SVA never claim the same output buffer two times in a row. If the aborting approach is applied, then the output process may also be able to display the aborted frame, albeit at a lower picture quality. This is for example applicable to frames consisting of multiple layers, where processing a base layer already provides an acceptable picture quality, and successive enhancement layers can be processed to improve the picture quality [Jarnikov *et al.*, 2004]. If the base layer and possibly some enhancement layers have been completed at a missed deadline, then the aborted frame can be used for output.

We now focus on handling input queue overflows. Let integers $i$ and $j$ denote the number of filled input buffers and filled output buffers, respectively. Initially, at $e(0)$, $i = 0$ and $j = 0$. For each frame that enters the input queue, $i$ is increased by one. For each frame that is completed, $i$ is decreased by one and $j$ is increased by one. At $e(\delta)$, when frame $\delta$ enters the input queue, and exactly one period before deadline $d(1)$, at most $\delta - 1$ frames have been completed. Hence, at $e(\delta)$, the sum of $i$ and $j$ becomes $\delta$. Disregarding input queue overflows and deadline misses, at each deadline $i$ is increased by one and $j$ is decreased by one. Because we assume that the number of input buffers corresponds to the number of output buffers, the first input queue overflow coincides with the first deadline miss. At this point in time, $i = \delta$ and $j = 0$. The input queue overflow is hence handled implicitly by the applied deadline miss approach. If the aborting approach is applied, then the released input buffer is used to insert the frame arriving from the input process. If the skipping approach is applied, then either a frame in a filled input buffer, or the frame arriving from the input process is skipped. In the former case, the input buffer containing the skipped frame is used to insert the frame arriving from the input process. Hence, both deadline miss approaches do not change the values $i = \delta$ and $j = 0$. As a result, every subsequent input queue overflow also coincides with a deadline miss, and vice versa. This means that only the handling of deadline misses is to be focused on, just because input queue overflows are handled implicitly by the applied deadline miss approach. Note that if the SVA becomes blocked at a milestone, then there is both no filled input buffer and no empty output buffer, i.e., $i = 0$ and $j = \delta$. The SVA is then blocked until the earliest deadline, when a new frame enters the input queue, and a processed frame is consumed from the output queue.

## 2.3 Budget and progress

The processor on which the SVA runs is applied for multitasking, which means that it executes a number of tasks next to each other. A dedicated scheduler, for example based on fixed-priority preemptive scheduling [Audsley *et al.*, 1995], switches repeatedly from task to task, which gives the appearance that all tasks run simultaneously. To study the SVA in isolation, we assume a fixed share of the available processing time is assigned to the SVA. More specifically, starting at $e(1)$, in each period $P$ we assume that the SVA is guaranteed a fixed processing-time budget $b$ ($0 \leq b \leq P$) by a resource manager; see for example [Otero Pérez *et al.*, 2003]. Budget $b$ can be viewed as a virtual processor, running $P/b$ times slower than the actual processor [Feng and Mok, 2002]. The distribution of the budget over a period is determined by the scheduler.

In Section 2.2 we noted that if the SVA becomes blocked at a milestone, then it is blocked until the earliest next deadline. To not waste processing time, in such a situation we assume that the scheduler immediately withdraws the SVA's remaining budget for the period. The scheduler redistributes the withdrawn budget over the other tasks as so-called gain time [Audsley *et al.*, 1994]. Other tasks may also generate gain time, which may partly be assigned to the SVA in addition to its budget. Also slack time [Lehoczky and Thuel, 1995] may be assigned to the SVA, which is time that has not been allocated by means of resource reservation. Unless mentioned otherwise, we assume that the SVA does not consume gain time or slack time.

**Example 2.3.** Figure 2.4 shows an example timeline, which illustrates the processing behavior of the SVA for $b = 0.5P$. We assume that $\delta = 2$ and that the skipping approach is applied to handle deadline misses. Disregarding quality levels and frame types, the processing times of frames 1 to 5 are given by $\mu(1) = 0.25P$, $\mu(2) = 1.25P$, $\mu(3) = P$, $\mu(4) = 0.5P$, and $\mu(5) = P$.

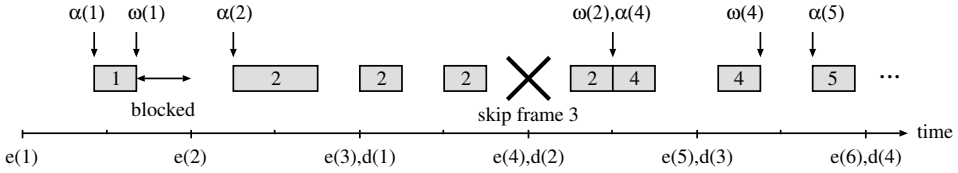


Figure 2.4. An example timeline, illustrating the processing behavior of the SVA for $b = 0.5P$.

Upon arrival at $e(1)$ of frame 1 in the input queue, the SVA starts to process as soon as it is scheduled on the processor, at $\alpha(1)$. At $\omega(1)$, the SVA becomes blocked because frame 2 is not yet present in the input queue. The remaining

budget of $0.25P$ is withdrawn. At $e(2)$, frame 2 enters the input queue, which means that the SVA is no longer blocked. As soon as the SVA is scheduled again, it starts processing frame 2, at $\alpha(2)$. Note that in the period between $e(3)$ and $e(4)$ the budget of $0.5P$ is distributed over two blocks of $0.25P$. Deadline $d(2)$ is missed, and frame 3 is skipped. □

Based on the periodically guaranteed budget $b$ we introduce a measure for frames called progress. The progress of frame $f$ at time $t$, denoted by $\lambda_t(f)$, provides an indication of how much budget is left for processing frame $f$, until the deadline at which the frame is planned to be consumed from the output queue. If we denote the total amount of budget left at time $t$ until deadline $d(f)$ by $b_t(f)$, then the progress of frame $f$ at time $t$ is given by

$$\lambda_t(f) = b_t(f)/b.$$

Normally, frame $f$ is planned to be consumed from the output queue at deadline $d(f)$. However, if the skipping approach is applied, then the planned deadline of consumption can change during processing. For example, if deadline $d(f)$ is missed, then from $d(f)$ onwards frame $f$ is planned to be consumed at deadline $d(f+1)$. Therefore, for the skipping approach the progress is given by

$$\lambda_t(f) = \begin{cases} b_t(f)/b & \text{if } t \leq d(f) \\ b_t(f+i)/b & \text{if } d(f+i-1) < t \leq d(f+i), \text{ for } i \in \mathbb{N}^+. \end{cases}$$

Note that the progress is computed based on guaranteed processing time only, i.e., a possible future consumption of gain time or slack time is ignored. We denote the progress of frame $f$ at start point $\alpha(f)$ and milestone $\omega(f)$ by $\lambda_\alpha(f)$ and $\lambda_\omega(f)$, respectively. Due to the fixed end-to-end latency, within the time interval $[\alpha(f), \omega(f)]$ the progress of frame $f$ is restricted to the interval $[0, \delta]$.

**Example 2.4.**

- In Figure 2.3A, the progress at the successive start points and milestones is given by $\lambda_\alpha(1) = 2$, $\lambda_\omega(1) = 0.25$, $\lambda_\alpha(2) = 1.25$, $\lambda_\omega(2) = 0$, $\lambda_\alpha(3) = 1$, $\lambda_\omega(3) = 0.5$, $\lambda_\alpha(4) = 1.5$, $\lambda_\omega(4) = 0.25$, and $\lambda_\alpha(5) = 1.25$.
- In Figure 2.3B, the progress at the successive start points and milestones is given by $\lambda_\alpha(1) = 2$, $\lambda_\omega(1) = 0.25$, $\lambda_\alpha(2) = 1.25$, $\lambda_\omega(2) = 0.75$, $\lambda_\alpha(4) = 1.75$, $\lambda_\omega(4) = 0.5$, and $\lambda_\alpha(5) = 1.5$.
- In Figure 2.4, the progress at the successive start points and milestones is given by $\lambda_\alpha(1) = 2$, $\lambda_\omega(1) = 1.5$, $\lambda_\alpha(2) = 2$, $\lambda_\omega(2) = 0.5$, $\lambda_\alpha(4) = 1.5$, $\lambda_\omega(4) = 0.5$, and $\lambda_\alpha(5) = 1.5$.

□

We now compute how the progress of a frame changes during processing. Let $f$ and $g$ be any pair of frames which are processed in succession. Usually, frame $g$ corresponds to frame $f+1$, but if the deadline of frame $f$ is missed and the skipping approach is applied, then frame $g$ may also correspond to a later frame. Without considering that the SVA may miss deadline $d(f)$, the progress of frame $f$ at its milestone can be expressed in $\lambda_\alpha(f)$ by

$$\lambda'_\omega(f) = \lambda_\alpha(f) - \frac{\mu(f)}{b}. \tag{2.1}$$

However, if $\lambda'_\omega(f) < 0$, then deadline $d(f)$ has been missed. If the aborting approach is applied, then frame $f$ is aborted at deadline $d(f)$, which implies $\omega(f) = d(f)$. If the skipping approach is applied, then a new deadline is assigned to frame $f$, and processing is continued. Upon completion the frame is used as the earliest next frame to be consumed by the output process. Hence, we find that

$$\lambda_\omega(f) = \begin{cases} 0 & \text{if } \lambda'_\omega(f) < 0, \text{ applying the aborting approach} \\ \lambda'_\omega(f) + \lceil -\lambda'_\omega(f) \rceil & \text{if } \lambda'_\omega(f) < 0, \text{ applying the skipping approach} \\ \lambda'_\omega(f) & \text{if } \lambda'_\omega(f) \geq 0. \end{cases} \tag{2.2}$$

Figure 2.5 shows $\lambda_\omega(f)$ as a function of $\lambda'_\omega(f)$, for the two deadline miss approaches. The number of deadlines misses for frame $f$, denoted by $ndm(f)$, is given by

$$ndm(f) = \begin{cases} 1 & \text{if } \lambda'_\omega(f) < 0, \text{ applying the aborting approach} \\ \lceil -\lambda'_\omega(f) \rceil & \text{if } \lambda'_\omega(f) < 0, \text{ applying the skipping approach} \\ 0 & \text{if } \lambda'_\omega(f) \geq 0. \end{cases} \tag{2.3}$$

The deadline at which frame $g$ is planned to be consumed by the output process falls exactly one time period later than the deadline at which frame $f$ is consumed by the output process. Hence, without considering that the SVA may become blocked at milestone $\omega(f)$, the progress of frame $g$ at its start point is given by

$$\lambda'_\alpha(g) = \lambda_\omega(f) + 1. \tag{2.4}$$

Clearly, $1 \leq \lambda'_\alpha(g) \leq \delta + 1$. If $\lambda'_\alpha(g) > \delta$, then frame $g$ is not present in the input queue at milestone $\omega(f)$, and the SVA becomes blocked. The SVA is then blocked until the earliest next deadline, when frame $g$ enters the input queue. Hence, we find that

$$\lambda_\alpha(g) = \begin{cases} \delta & \text{if } \lambda'_\alpha(g) > \delta \\ \lambda'_\alpha(g) & \text{otherwise.} \end{cases} \tag{2.5}$$

By definition, $\lambda_\alpha(1) = \delta$. Hence, for each frame $f \geq 1$ that is processed it holds that $1 \leq \lambda_\alpha(f) \leq \delta$ and $0 \leq \lambda_\omega(f) \leq \delta$.

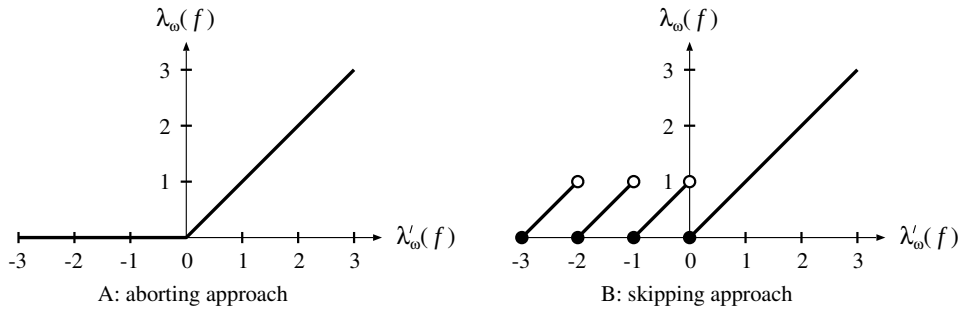


Figure 2.5. $\lambda_\omega(f)$ as a function of $\lambda'_\omega(f)$, for the aborting approach (fig. A) and the skipping approach (fig. B).

## 2.4 Objective

Informally speaking, the objective of the work described in this thesis is to maximize the output quality of the SVA as perceived by a user, for an indefinite sequence of video frames to be processed. In this section we formalize this objective.

As mentioned in Section 1.5, we consider three aspects of user-perceived quality: the quality levels at which frames are processed, deadline misses, and changes in the quality level between successively processed frames. The user-perceived output quality of the SVA is maximized if all frames are processed at the highest quality level, and if there are no deadline misses. This can be achieved by choosing budget $b$ equal to or higher than the worst-case periodic processing-time needs of the SVA, a so-called worst-case budget. If budget $b$ is smaller than worst-case, the situation that we generally consider in this thesis, then we have to trade-off the three user-perception aspects to find an optimum. As mentioned in Section 1.5, we apply the notion of QoS to balance the three aspects in a single QoS measure, which represents the overall user satisfaction.

As a first step towards defining the QoS measure we assign a revenue $r(f)$ to each frame $f$ that has been processed. The revenue of a processed frame is a real number that indicates to what extent the three QoS parameters are satisfied for the frame. We define revenue $r(f)$ by

$$r(f) = R_{\mathrm{ql}}(q(f)) - ndm(f) \cdot P_{\mathrm{dm}} - P_{\mathrm{qlc}}(q(prev(f)), q(f)), \tag{2.6}$$

where

- $R_{\mathrm{ql}}(q)$ is a positive-valued reward for using quality level $q$,
- $P_{\mathrm{dm}}$ is a positive-valued penalty for the occurence of a deadline miss,

- $prev(f)$ is the frame that was processed right before frame $f$ was processed[1], and
- $P_{\text{qlc}}(q, q')$ is a positive-valued penalty for changing the quality level from $q$ to $q'$.

To define the QoS measure, we should not look at the revenues of individual frames, but we should look at the revenues of all frames that are processed. To this end, we define the QoS measure as the average revenue per processed frame (in short: average revenue per frame, or average revenue), which allows us to apply the measure to video sequences of varying length. We assume that the QoS measure reflects the overall user satisfaction of the output signal, provided that the rewards and penalties in (2.6) are well chosen. Tuning the QoS measure, i.e., determining appropriate values for the various rewards and penalties, can be done on the basis of user-perception experiments.

## 2.5 QoS control problem

At each start point of a frame, the controller has to select the quality level at which the frame is processed. The problem we consider in this thesis is to find a quality-level selection strategy for the controller (in short a *control strategy* or a *strategy*) that maximizes the average revenue for an indefinite sequence of frames to be processed. We call this problem the *QoS control problem*.

The QoS control problem is an on-line problem, as the controller has to select the quality level for each frame without knowing how complex the frame is, or how complex the frames are that follow. To estimate the processing time of a frame at a particular quality level, the controller can make use of statistics of frames that have been processed earlier. In Chapter 4 we present a control strategy that is computed based on the processing-time statistics of a reference sequence of frames. In Chapter 6 we present an adaptive control strategy that learns at run time how to behave from experienced processing times.

[1] For frame $f = 1$ we define $q(prev(f)) = q_1$.

# 3

# Reinforcement learning

The QoS control problem formulated in Section 2.5 is a stochastic decision problem, a problem in which optimal decisions are sought subject to uncertainty in instance data. A common way to address a stochastic decision problem is by modeling it as a reinforcement learning task. This chapter provides a short introduction to reinforcement learning, based on the textbook of Sutton and Barto [1998].

First, in Section 3.1 we describe the basic reinforcement learning model. Next, in Section 3.2 we introduce value functions, which lay the foundation for many reinforcement learning algorithms. Using value functions, in Section 3.3 we discuss a number of stochastic dynamic programming algorithms, and in Section 3.4 we discuss the Q-learning algorithm.

## 3.1 Reinforcement learning model

Reinforcement learning is a computational approach to goal-directed learning from interaction. It considers a system in which an agent repeatedly interacts with its environment. The environment can occupy different states, and the agent can influence the state of the environment by taking actions. The set of states that can be occupied by the environment is denoted by $S$, and the set of actions that can be taken by the agent is denoted by $A$. At discrete time steps, $t = 1, 2, \ldots$, the environment presents its state $s_t \in S$ to the agent, and next the agent takes an action

$a_t \in A(s_t)$, where $A(s_t) \subseteq A$ denotes the set of actions that can be taken by the agent if the environment is in state $s_t$[1]. Action $a_t$ has influence on the environment, and, as a result, the state of the environment at time step $t+1$ may differ from state $s_t$. In general, the new state $s_{t+1}$ cannot be derived deterministically from state $s_t$ and action $a_t$, i.e., the state transition is stochastic. At each time step $t > 1$, along with state $s_t$ the environment also presents a revenue $r_t$ to the agent. Revenue $r_t$ is a real number that expresses the intrinsic desirability of the state transition from $s_{t-1}$ to $s_t$. Figure 3.1 shows the interaction between the agent and its environment at a time step $t > 1$.

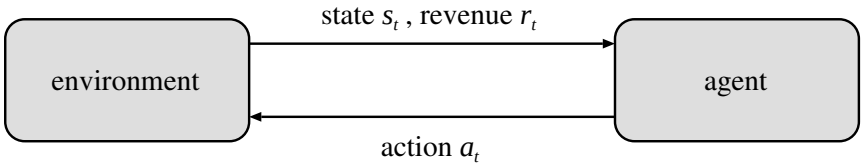


Figure 3.1. The interaction between a reinforcement learning agent and its environment at a time step $t > 1$.

The goal of the agent is to take actions that will maximize the expected return, where the return is defined as a function of all revenues to be received at future time steps. This task is called the reinforcement learning task. In case of a finite time horizon, when the total number of time steps is finite, and known before $t = 1$, the return is usually defined as the sum of all future revenues. In the more general case of an infinite time horizon, which we assume, there is no clearly defined final time step. As a result, the sum of all future revenues may be infinite. To obtain a finite measure, the return at time step $t$, denoted by $R_t$, is defined as the discounted sum of all future revenues, i.e.,

$$R_t = r_{t+1} + \sum_{i=1}^{\infty} \gamma^i r_{t+1+i},$$

where $\gamma$ $(0 \leq \gamma < 1)$ is a discount rate. The return has a finite value as long as the revenues are bounded. If $\gamma = 0$, then the goal of the agent is to maximize the immediate revenues, and the closer $\gamma$ approaches one the stronger the agent takes future revenues into account, i.e., the stronger the agent tries to maximize the average revenue received per time step. In general, an action of the agent does not only influence the new state and the immediate revenue, but through the new

[1] A time step $t$ is actually a (short) time interval that starts when the environment presents its state $s_t$ to the agent, and that ends when the agent returns action $a_t$. In the time interval the agent may perform some computations to select action $a_t$ from set $A(s_t)$, and to improve its strategy for selecting actions. Successive time steps are not necessarily equidistant in time.

state also all subsequent states and all subsequent revenues. Hence, the agent may have to take actions which do not yield the highest possible immediate revenue, to obtain a greater total revenue in the long run.

A state signal of the environment is said to satisfy the Markov property if state $s_{t+1}$ and revenue $r_{t+1}$ can be predicted from state $s_t$ and action $a_t$ as well as they can be predicted from the states, actions, and revenues $s_1, a_1, s_2, r_2, a_2, \ldots, s_t, r_t, a_t$. In other words, a Markov state signal is memoryless. A reinforcement learning task that satisfies the Markov property is called a Markov decision process (MDP) [Van der Wal, 1981; White, 1993; Puterman, 1994; Sutton and Barto, 1998]. A finite MDP is characterized by a finite set $S$ of states and a finite set $A$ of actions. Let $s, s' \in S$, let $a \in A$, and let $t \geq 1$. For a finite MDP, the one-step dynamics of the environment are given by state-transition probabilities

$$p^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\},$$

and corresponding expected revenues

$$r^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}.$$

We assume a finite MDP, and we assume that the state-transition probabilities and expected revenues do not change over time.

To select actions, the agent makes use of a policy. A stochastic policy $\pi$ keeps for each state $s \in S$ and for each action $a \in A(s)$ a probability $\pi(s, a)$ of selecting action $a$ in state $s$. A special case of stochastic policies are deterministic policies. A deterministic policy $\pi$ keeps for each state $s \in S$ a single action $\pi(s) \in A(s)$ to be chosen.

If the state-transition probabilities and expected revenues are available before the first time step, then it is possible to compute an optimal policy for the agent. Alternatively, the agent can also learn an optimal policy at run time, while using the learned policy at the same time to select actions. A learning agent usually starts with an arbitrary policy that is updated (learned) at each time step $t > 1$ using the observed state and revenue.

## 3.2 Value functions

Value functions lay the foundation for many reinforcement learning algorithms. A value function expresses how good a particular policy is in the long run, in terms of the expected return.

The state value of state $s$ under policy $\pi$, denoted by $V^\pi(s)$, is defined as the expected return, starting from state $s$, and following policy $\pi$ to take actions, i.e.,

$$V^\pi(s) = E_\pi\{R_t \mid s_t = s\},$$

where $E_\pi$ denotes the expected value of the return under policy $\pi$. Function $V^\pi$

is called the state-value function of policy $\pi$. The state-value function can be expressed recursively:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{R_t \mid s_t = s\} \\ &= E_{\pi}\{r_{t+1} + \gamma R_{t+1} \mid s_t = s\} \\ &= \sum_{a \in A(s)} \pi(s, a) \sum_{s' \in S} p^a_{ss'} r^a_{ss'} + \gamma E_{\pi}\{R_{t+1} \mid s_t = s\} \\ &= \sum_{a \in A(s)} \pi(s, a) \sum_{s' \in S} p^a_{ss'} (r^a_{ss'} + \gamma E_{\pi}\{R_{t+1} \mid s_{t+1} = s'\}) \\ &= \sum_{a \in A(s)} \pi(s, a) \sum_{s' \in S} p^a_{ss'} (r^a_{ss'} + \gamma V^{\pi}(s')). \end{aligned} \tag{3.1}$$

Equation (3.1) is known as the Bellman equation for $V^{\pi}$. It expresses the state value of state $s$ in the state values of all possible successor states $s'$.

The action value of taking action $a$ in state $s$ under policy $\pi$, denoted by $Q^{\pi}(s, a)$, is defined as the expected return, starting from state $s$, taking action $a$, and thereafter following policy $\pi$ to take actions, i.e.,

$$Q^{\pi}(s, a) = E_{\pi}\{R_t \mid s_t = s,\ a_t = a\}.$$

Function $Q^{\pi}$ is called the action-value function of policy $\pi$. The action-value function can be expressed in the state-value function:

$$\begin{aligned} Q^{\pi}(s, a) &= E_{\pi}\{R_t \mid s_t = s,\ a_t = a\} \\ &= E_{\pi}\{r_{t+1} + \gamma R_{t+1} \mid s_t = s,\ a_t = a\} \\ &= \sum_{s' \in S} p^a_{ss'} r^a_{ss'} + \gamma E_{\pi}\{R_{t+1} \mid s_t = s,\ a_t = a\} \\ &= \sum_{s' \in S} p^a_{ss'} (r^a_{ss'} + \gamma E_{\pi}\{R_{t+1} \mid s_{t+1} = s'\}) \\ &= \sum_{s' \in S} p^a_{ss'} (r^a_{ss'} + \gamma V^{\pi}(s')). \end{aligned} \tag{3.2}$$

A policy $\pi$ is defined to be equivalent to a policy $\pi'$ if $V^{\pi}(s) = V^{\pi'}(s)$ for all $s \in S$, and $\pi$ is defined to be better than $\pi'$ if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in S$ and inequality holds for at least one state $s$. A policy is optimal if it is better than or equivalent to all other policies. Because we assume that the state-transition probabilities and expected revenues do not change over time, always one or more optimal policies exist. We denote any optimal policy by $\pi^*$. Optimal policies share the same state-value function and action-value function, the optimal state-value function and optimal action-value function, which are denoted by $V^*$ and $Q^*$, respectively. Function $V^*$ is defined by

$$V^*(s) = \max_{\pi} V^{\pi}(s) \quad \forall s \in S,$$

and function $Q^*$ is defined by

$$Q^*(s, a) = \max_{\pi} Q^{\pi}(s, a) \quad \forall s \in S, \quad \forall a \in A(s).$$

Because $V^*$ and $Q^*$ are both optimal value functions, it must hold that

$$V^*(s) = \max_{a \in A(s)} Q^*(s, a) \quad \forall s \in S.$$

Using (3.2), for $V^*$ we can now derive that

$$\begin{aligned} V^*(s) &= \max_{a \in A(s)} Q^*(s, a) \\ &= \max_{a \in A(s)} \sum_{s' \in S} p^a_{ss'}(r^a_{ss'} + \gamma V^*(s')), \end{aligned} \tag{3.3}$$

and for $Q^*$ we can derive that

$$\begin{aligned} Q^*(s, a) &= \sum_{s' \in S} p^a_{ss'}(r^a_{ss'} + \gamma V^*(s')) \\ &= \sum_{s' \in S} p^a_{ss'}(r^a_{ss'} + \gamma \max_{a' \in A(s')} Q^*(s', a')). \end{aligned} \tag{3.4}$$

Equation (3.3), known as the Bellman optimality equation for $V^*$, is a system of $|S|$ nonlinear equations in $|S|$ unknowns. Equation (3.4), known as the Bellman optimality equation for $Q^*$, is a system of $|S| \cdot |A|$ nonlinear equations in $|S|$ unknowns.

If the state-transition probabilities $p^a_{ss'}$ and expected revenues $r^a_{ss'}$ are known, then the Bellman optimality equations can be solved. Given $V^*$, an optimal policy $\pi^*$ can be found by for each state $s$ assigning a probability of zero to every action $a \in A(s)$ for which (3.3) does not attain the maximum value for $s$, and by randomly distributing a probability of one over all actions $a \in A(s)$ for which (3.3) attains the maximum value for $s$. Given $Q^*$, an optimal policy can be found by for each state $s$ assigning a probability of zero to every action $a \in A(s)$ for which $Q^*(s, a)$ does not attain the maximum value for $s$, and by randomly distributing a probability of one over all actions $a \in A(s)$ for which $Q^*(s, a)$ attains the maximum value for $s$. Note that for each state it is possible to assign a probability of one to only a single action, and a probability of zero to all other actions. Hence, always at least one optimal deterministic policy exists. Without loss of optimality, we restrict ourselves to deterministic policies only. As compared to a stochastic policy, a deterministic policy can be stored more efficiently, and it can be looked-up more easily.

## 3.3 Stochastic dynamic programming algorithms

To find an optimal policy, the Bellman optimality equations for $V^*$ or $Q^*$ have to be solved. Although $V^*$ and $Q^*$ can be computed exactly, it is common practice to find a good approximation, at the benefit of a significantly shorter computation time. Usually this has very little or no influence on the quality of the policy that

is derived from the solution. In this section we discuss three stochastic dynamic programming algorithms that can be used to find an optimal policy in acceptable time: policy iteration, value iteration, and successive approximation.

The policy iteration algorithm consists of two procedures, which are executed alternatingly: policy evaluation and policy improvement. Policy evaluation is a procedure that computes the state-value function $V^\pi$ of a given policy $\pi$. Next, policy improvement uses $V^\pi$ as input to construct a policy $\pi'$ that is better than or equivalent to $\pi$. In turn, the constructed policy $\pi'$ is used as input for policy evaluation. In combination, the two procedures can be used to compute an optimal policy $\pi^*$.

The policy evaluation procedure is given by (3.1), transformed into an iterative update rule:

$$V_{i+1}(s) := \sum_{s' \in S} p_{ss'}^{\pi(s)} (r_{ss'}^{\pi(s)} + \gamma V_i(s')) \quad \text{for } i = 1, 2, \ldots .$$

This update rule can be programmed using two arrays, one to store the state values $V_{i+1}(s)$ that are computed, and one to store the given state values $V_i(s)$. Function $V^\pi$ is a fixed point for the update rule, because (3.1) assures equality in this case. Given an arbitrarily chosen state-value function $V_1$, the sequence of iteratively computed state-value functions $V_2, V_3, \ldots$ can be shown to converge to $V^\pi$ as $i \to \infty$. The procedure is usually stopped when $|V_{i+1}(s) - V_i(s)| < \varepsilon$ for all $s \in S$, for some positive error $\varepsilon$ that is chosen close to zero.

Next, the policy improvement procedure is based on the following theorem [Sutton and Barto, 1998].

**Theorem 3.1.** *Let $\pi$ and $\pi'$ be policies such that $V^\pi(s) \leq Q^\pi(s, \pi'(s))$ for all $s \in S$. Then $V^\pi(s) \leq V^{\pi'}(s)$ for all $s \in S$.* □

Hence, for a policy $\pi$, which is implicitly given by state-value function $V^\pi$, an equivalent or better policy $\pi'$ can be constructed by selecting

$$\begin{aligned} \pi'(s) &:= \arg\max_{a \in A(s)} Q^\pi(s, a) \\ &= \arg\max_{a \in A(s)} E\{r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t = s, a_t = a\} \\ &= \arg\max_{a \in A(s)} \sum_{s' \in S} p_{ss'}^a (r_{ss'}^a + \gamma V^\pi(s')), \end{aligned} \tag{3.5}$$

for all $s \in S$. If $\pi'$ is equivalent to $\pi$, i.e., if $V^{\pi'}(s) = V^\pi(s)$ for all $s \in S$, then, using (3.5), we find that

$$V^{\pi'}(s) = \max_{a \in A(s)} \sum_{s' \in S} p_{ss'}^a (r_{ss'}^a + \gamma V^{\pi'}(s')).$$

This is the same as Bellman optimality equation (3.3). Hence, $\pi'$ and $\pi$ are both

optimal policies.

The policy iteration algorithm starts with an arbitrarily chosen policy. In each iteration of the algorithm, by successively performing a run of the policy evaluation procedure and a run of the policy improvement procedure, a policy is constructed that is better than or equivalent to the policy of the previous iteration. If two successive policies are equivalent, then they are both optimal, and the algorithm stops. After every policy evaluation run, the final state-value function can be used as initial state-value function for the next policy evaluation run. This is expected to speed-up convergence, because the policies of successive iterations are likely to be closely related.

Formally, the policy evaluation procedure requires an infinite number of iterations to compute the state-value function of a given policy. In practice, the procedure is stopped after a finite number of iterations, when the difference between two successive state-value functions is small enough. Nevertheless, even if the procedure is stopped after only a single iteration, then the policy iteration algorithm can be shown to converge to an optimal policy. The variant of policy iteration in which the policy evaluation procedure is always stopped after only a single iteration is called value iteration. The single iteration of policy evaluation and the policy improvement procedure can be combined into a single iterative update rule:

$$V_{i+1}(s) := \max_{a \in A(s)} \sum_{s' \in S} p^a_{ss'} (r^a_{ss'} + \gamma V_i(s')).$$

Formally, the value iteration algorithm requires an infinite number of iterations to find an optimal policy. In practice, the algorithm is stopped after a finite number of iterations, when the difference between two successive state-value functions is small enough.

Policy iteration and value iteration both cannot handle a discount rate $\gamma = 1$. Hence, to maximize the average revenue per state transition, $\gamma$ should be chosen very close to one, but that will slow down convergence. To truly maximize the average revenue, we can apply the successive approximation algorithm [Van der Wal, 1981]. This algorithm is basically the same as value iteration, but it uses a discount rate $\gamma = 1$ and a different stop criterion. Because $\gamma = 1$, the successively computed state-value functions $V_i$ no longer converge to $V^*$. Instead, for each state $s$ the difference $V_{i+1}(s) - V_i(s)$ converges to an optimum as $i \to \infty$, viz. the expected average revenue per state transition for an optimal policy. The algorithm is stopped when $\max_{s \in S}(V_{i+1}(s) - V_i(s)) - \min_{s \in S}(V_{i+1}(s) - V_i(s)) < \varepsilon$ for some positive error $\varepsilon$ that is chosen close to zero. Figure 3.2 shows pseudo-code for the successive approximation algorithm. Apart from computing an optimal policy, the algorithm also returns the expected average revenue per state transition for an optimal policy.

**for** each $s \in S$ **do**
    initialize $V(s)$ arbitrarily;
**repeat**
    **for** each $s \in S$ **do**
        $V_{\text{old}}(s) := V(s)$;
    *max_dif* $:= -\infty$;
    *min_dif* $:= +\infty$;
    **for** each $s \in S$ **do**
    **begin**
        $V(s) := \max_{a \in A(s)} \sum_{s' \in S} p^a_{ss'}(r^a_{ss'} + V_{\text{old}}(s'))$;
        **if** $V(s) - V_{\text{old}}(s) >$ *max_dif*
        **then** *max_dif* $:= V(s) - V_{\text{old}}(s)$;
        **if** $V(s) - V_{\text{old}}(s) <$ *min_dif*
        **then** *min_dif* $:= V(s) - V_{\text{old}}(s)$;
    **end**
**until** *max_dif* $-$ *min_dif* $< \varepsilon$;
**for** each $s \in S$ **do**
    $\pi(s) := \arg\max_{a \in A(s)} \sum_{s' \in S} p^a_{ss'}(r^a_{ss'} + V(s'))$;
*expected_average_revenue* $:=$ (*max_dif* $+$ *min_dif*)$/2$;

Figure 3.2. Pseudo-code for the successive approximation algorithm.

## 3.4 Q-learning

Q-learning [Watkins, 1989] is an on-line reinforcement learning algorithm that does not require a model of the environment in terms of state-transition probabilities and expected revenues. The agent starts without any knowledge, and at the successive time steps it has to learn how to take actions, to maximize the expected return. To learn optimal behavior, the agent maintains an estimate of the optimal action-value function $Q^*$. The estimated action value of state $s \in S$ and action $a \in A(s)$ is denoted by $Q(s, a)$. Initially, the $Q$-values are chosen arbitrarily. At each time step $t > 1$, prior to selecting action $a_t$, the agent updates action value $Q(s_{t-1}, a_{t-1})$ using Sutton's temporal difference error [Sutton, 1988]. This error is defined as the difference between the immediate revenue $r_t$ plus the discounted estimated state value of state $s_t$, and $Q(s_{t-1}, a_{t-1})$. The state value of $s_t$ is estimated by $\max_{a \in A(s_t)} Q(s_t, a)$. The update at time step $t > 1$ is given by

$$\begin{aligned} Q(s_{t-1}, a_{t-1}) &:= Q(s_{t-1}, a_{t-1}) + \psi(r_t + \gamma \max_{a \in A(s_t)} Q(s_t, a) - Q(s_{t-1}, a_{t-1})) \\ &= (1 - \psi) Q(s_{t-1}, a_{t-1}) + \psi(r_t + \gamma \max_{a \in A(s_t)} Q(s_t, a)), \end{aligned}$$

where $\psi$ $(0 < \psi \leq 1)$ is a step-size parameter called the learning rate. The learning rate controls how much weight is given to the new experience, as opposed to the old

estimate $Q(s_{t-1}, a_{t-1})$. If $\psi = 1$, then full weigth is given to the new experience, and the closer $\psi$ approaches zero, the more function $Q$ is building up an average over all past experiences.

For learning it is unimportant which actions are taken by the agent. This is because action value $Q(s_{t-1}, a_{t-1})$ is always updated using the action that looks optimal in state $s_t$. In other words, the policy that is learned does not have to correspond to the policy that is applied for control. For this reason, Q-learning is called an off-policy algorithm. To maximize the expected return the agent should take so-called greedy actions, i.e., actions $a_t$ for which $Q(s_t, a_t)$ is maximal. However, to guarantee convergence of $Q$ to $Q^*$, each estimate $Q(s, a)$ should be updated many times. This can only be guaranteed if the agent takes non-greedy actions as well. This dilemma between exploiting and exploring actions is addressed by means of $\varepsilon$-greedy action selection. This means that most of the time the agent selects a greedy action, but with a probability $\varepsilon$ the agent selects an action at random. Usually, in the beginning $\varepsilon$ is chosen close to one, and as time passes by the value of $\varepsilon$ is moved slowly towards zero. Figure 3.3 shows pseudo-code for the Q-learning algorithm.

---

**for** each $s \in S$ **do**
    **for** each $a \in A(s)$ **do**
        initialize $Q(s, a)$ arbitrarily;
{*time step* $t = 1$}
observe state $s$;
$a := \arg\max_{a \in A(s)} Q(s, a)$;
**while** true **do**
**begin**
    $s_{\text{old}} := s$;
    $a_{\text{old}} := a$;
    take action $a$;
    {*time step* $t > 1$}
    observe state $s$ and revenue $r$;
    $Q(s_{\text{old}}, a_{\text{old}}) := Q(s_{\text{old}}, a_{\text{old}}) + \psi(r + \gamma \max_{a \in A(s)} Q(s, a) - Q(s_{\text{old}}, a_{\text{old}}))$;
    **if** Random$[0, 1] < \varepsilon$
    **then** randomly select action $a$ from set $A(s)$;
    **else** $a := \arg\max_{a \in A(s)} Q(s, a)$;
**end**

---

Figure 3.3. Pseudo-code for the Q-learning algorithm.

# 4

# Off-line solution approach

In Chapter 2 we presented a processing model for an SVA, and we formulated the QoS control problem. Next, in Chapter 3 we gave an introduction to reinforcement learning. Given this, we now present our first solution approach to the QoS control problem. Our approach is based on modeling the problem as a finite MDP, and solving this model before run time using predetermined processing-time statistics. The solution is given by a policy that can be used by the SVA's controller at run time to select the quality level for each frame to be processed. We call this approach the off-line solution approach.

First, in Section 4.1 we model the QoS control problem as a finite MDP, and we estimate state-transition probabilities and expected revenues for the model. Next, in Section 4.2 we discuss how the MDP model is solved. In Section 4.3 we discuss how the resulting policy can be used as a quality-level control strategy. Finally, in Section 4.4 we assess the effectiveness of the approach by means of simulation experiments.

## 4.1 Finite MDP model

We now model the QoS control problem as a finite MDP. Because a Markov decision process is by definition memoryless, in the model it is implicitly assumed that the processing times of successive frames are independent. In practice, however,

there may be dependencies in the processing times of successive frames. We come back to this issue in Section 4.4.6, and we discuss it further in Chapter 5.

In the model, the role of environment is assigned to the SVA, and the role of agent is assigned to the controller. The discrete time steps at which the environment presents its state and a revenue to the agent and at which the agent takes an action are given by the start points of frames, at which the SVA calls the controller to obtain the quality level. Hence, the set $A$ of actions that can be taken by the agent is given by the finite set of quality levels $Q$. Because each frame can be processed at any quality level, we have $A(s) = Q$ for all $s \in S$.

At each start point of a frame, the controller has to select the quality level at which the frame will be processed, based on the state of the SVA. Because we assume a budget that is smaller than the SVA's worst-case needs for processing frames at the highest quality level, $q_{n_Q}$, the controller should sometimes select a lower quality level for frames, to prevent deadline misses. Hence, the state signal should comprise information that can be used to estimate whether or not a frame's deadline will be missed if the frame is processed at a particular quality level. As a first component of the state we therefore consider the progress of the frame to be processed at the start point, because a higher progress leads to a lower probability of missing the frame's deadline. As derived in Section 2.3, the progress of a frame at its start point is a real number from the interval $[1, \delta]$. Because we need a finite set of states, we define a finite set $\Lambda$ of $n_\Lambda$ ($\geq 2$) equal-sized progress intervals $\lambda_1, \ldots, \lambda_{n_\Lambda}$ between 1 and $\delta$ by

$$\lambda_i = \begin{cases} [1 + \frac{(i-1)(\delta-1)}{n_\Lambda}, 1 + \frac{i(\delta-1)}{n_\Lambda}) & \text{if } i < n_\Lambda \\ [\delta - \frac{\delta-1}{n_\Lambda}, \delta] & \text{if } i = n_\Lambda. \end{cases}$$

The lower bound and upper bound of a progress interval $\lambda \in \Lambda$ are denoted by $\underline{\lambda}$ and $\overline{\lambda}$, respectively. As a second component of the state we consider the frame type, because it may provide information on the frame's processing complexity, and thus on the probability of missing the frame's deadline. Finally, because we want to minimize the number and size of quality-level changes between successively processed frames, as a third component of the state we consider the quality level that was used for the most recently processed frame, the so-called previous quality level. As mentioned, for frame 1 we define the previous quality level to be the lowest one, i.e., $q_1$. Hence, we define the set $S$ of states by $\Lambda \times \Phi \times Q$. The state given by a progress interval $\lambda$, a frame type $\phi$, and a previous quality level $q$ is denoted by $(\lambda, \phi, q)$. For a state $s \in S$, we denote the corresponding progress interval, frame type, and previous quality level by $\lambda_s$, $\phi_s$, and $q_s$, respectively.

Consider the transition from state $s \in S$ to state $s' \in S$ under the selection of

quality level $q \in Q$ in state $s$. For state $s'$, progress interval $\lambda_{s'}$ can be predicted from progress interval $\lambda_s$, frame type $\phi_s$, and quality level $q$ using state-transition probabilities $p^q_{ss'}$. The previous quality level $q_{s'}$ is given by quality level $q$. To obtain a memoryless state signal, we have to assume that frame type $\phi_{s'}$ can be predicted from frame type $\phi_s$ only. Note that this assumption may not be suitable for some video sequences.

Given the above, we now derive state-transition probabilities and corresponding expected revenues for the model.

### 4.1.1 State-transition probabilities

To compute the state-transition probabilities $p^q_{ss'}$, we consider the transition from state $s \in S$ to state $s' \in S$ under the selection of quality level $q \in Q$ in state $s$. After the transition it must hold that $q_{s'} = q$, which means that $p^q_{ss'} = 0$ for any state $s'$ with $q_{s'} \neq q$. For all other states $s'$ the state-transition probabilities can be computed as follows.

Let $f$ and $g$ be any pair of frames which are processed in succession. Usually, frame $g$ corresponds to frame $f+1$, but if the deadline of frame $f$ is missed and the skipping approach is applied, then frame $g$ may also correspond to a later frame. Given the progress $\lambda_\alpha(f)$ and the type $\phi(f)$ of frame $f$, we define $P^{g:\lambda,\phi}_{\lambda_\alpha(f),\phi(f)}$ as the probability that frame $g$ has a progress $\lambda_\alpha(g) \in \lambda$ and a type $\phi(g) = \phi$. This probability is the product of two independent probabilities. The first one is the probability that a frame of type $\phi$ is processed immediately after a frame of type $\phi(f)$ has been processed, denoted by $\Pr(\phi(f), \phi)$. This probability can be estimated using statistics on how frequently the different frame types appear successively. The second one is the probability that $\lambda_\alpha(g) \in \lambda$, given $\lambda_\alpha(f)$ and $\phi(f)$. For this second probability, denoted by $P^{g:\lambda}_{\lambda_\alpha(f),\phi(f)}$, we derive that

$$P^{g:\lambda}_{\lambda_\alpha(f),\phi(f)} = \begin{cases} \Pr(\lambda_\alpha(g) < \overline{\lambda}) & \\ = 1 - \Pr(\lambda_\alpha(g) \geq \overline{\lambda}) & \text{if } \lambda = \lambda_1 \\ \Pr(\lambda_\alpha(g) \geq \underline{\lambda}) & \text{if } \lambda = \lambda_{n_\Lambda} \\ \Pr(\underline{\lambda} \leq \lambda_\alpha(g) < \overline{\lambda}) & \\ = \Pr(\lambda_\alpha(g) \geq \underline{\lambda}) - \Pr(\lambda_\alpha(g) \geq \overline{\lambda}) & \text{otherwise.} \end{cases} \tag{4.1}$$

In this equation, we compute the probabilities $\Pr(\lambda_\alpha(g) \geq x)$ for values of $x \in (1, \delta)$ as follows. For frame type $\phi \in \Phi$ and quality level $q \in Q$, let random variable $X_{\phi q}$ denote the time needed by the SVA to process a single frame of type $\phi$ at quality level $q$. Let $F_{\phi q}$ denote the cumulative distribution function of $X_{\phi q}$, i.e.,

$$F_{\phi q}(x) = \Pr(X_{\phi q} \leq x). \tag{4.2}$$

For the aborting approach, for $1 < x < \delta$ we derive that

$$\begin{aligned}
\Pr(\lambda_\alpha(g) \geq x) &= \{\text{using (2.5) and } x < \delta\} \\
&\quad \Pr(\lambda'_\alpha(g) \geq x) \\
&= \{\text{using (2.4)}\} \\
&\quad \Pr(\lambda_\omega(f) \geq x-1) \\
&= \{\text{using (2.2) and } x-1 > 0\} \\
&\quad \Pr(\lambda'_\omega(f) \geq x-1) \\
&= \{\text{using (2.1)}\} \\
&\quad \Pr(X_{\phi(f)q(f)} \leq b(\lambda_\alpha(f) - x + 1)) \\
&= \{\text{using (4.2)}\} \\
&\quad F_{\phi(f)q(f)}(b(\lambda_\alpha(f) - x + 1)).
\end{aligned}$$

For the skipping approach, for $1 < x < 2$ we derive that

$$\begin{aligned}
\Pr(\lambda_\alpha(g) \geq x) &= \{\text{using (2.5), } x < 2 \text{ and } \delta \geq 2\} \\
&\quad \Pr(\lambda'_\alpha(g) \geq x) \\
&= \{\text{using (2.4)}\} \\
&\quad \Pr(\lambda_\omega(f) \geq x-1) \\
&= \{\text{using (2.2) and } 0 < x-1 < 1\} \\
&\quad \Pr(\lambda'_\omega(f) \geq x-1) + \sum_{i=1}^{\infty} \Pr(x-1-i \leq \lambda'_\omega(f) < 1-i) \\
&= \{\text{using (2.1)}\} \\
&\quad \Pr(X_{\phi(f)q(f)} \leq b(\lambda_\alpha(f) - x + 1)) \\
&\quad + \sum_{i=1}^{\infty} \Pr(X_{\phi(f)q(f)} \leq b(\lambda_\alpha(f) - x + 1 + i)) \\
&\quad - \sum_{i=1}^{\infty} \Pr(X_{\phi(f)q(f)} \leq b(\lambda_\alpha(f) - 1 + i)) \\
&= \{\text{using (4.2)}\} \\
&\quad F_{\phi(f)q(f)}(b(\lambda_\alpha(f) - x + 1)) \\
&\quad + \sum_{i=1}^{\infty} F_{\phi(f)q(f)}(b(\lambda_\alpha(f) - x + 1 + i)) \\
&\quad - \sum_{i=1}^{\infty} F_{\phi(f)q(f)}(b(\lambda_\alpha(f) - 1 + i)),
\end{aligned}$$

and for $2 \leq x < \delta$ we derive that

$$\begin{aligned}
\Pr(\lambda_\alpha(g) \geq x) &= \{\text{using (2.5) and } x < \delta\} \\
&\quad \Pr(\lambda'_\alpha(g) \geq x) \\
&= \{\text{using (2.4)}\} \\
&\quad \Pr(\lambda_\omega(f) \geq x-1) \\
&= \{\text{using (2.2) and } x-1 \geq 1\} \\
&\quad \Pr(\lambda'_\omega(f) \geq x-1) \\
&= \{\text{using (2.1)}\} \\
&\quad \Pr(X_{\phi(f)q(f)} \leq b(\lambda_\alpha(f) - x + 1)) \\
&= \{\text{using (4.2)}\} \\
&\quad F_{\phi(f)q(f)}(b(\lambda_\alpha(f) - x + 1)).
\end{aligned}$$

To compute the state-transition probabilities $p^q_{ss'}$, let $s$ be the state of the SVA at start point $\alpha(f)$, and let $s'$ be the state of the SVA at start point $\alpha(g)$. To apply (4.1), a value for $\lambda_\alpha(f)$ is needed. Unfortunately, $\lambda_\alpha(f)$ is not completely determined by state $s$, i.e., we only know that $\lambda_\alpha(f)$ is somewhere in progress interval $\lambda_s$. A worst-case approximation is obtained by using the lower bound of the progress interval, i.e., by approximating

$$\lambda_\alpha(f) \approx \underline{\lambda_s}. \tag{4.3}$$

Given the above, the state-transition probabilities $p^q_{ss'}$ can be approximated as follows, depending on the applied deadline miss approach and progress interval $\lambda_{s'}$:

- aborting approach, $\lambda_{s'} = \lambda_1$:

$$p^q_{ss'} \approx \Pr(\phi_s, \phi_{s'}) \cdot (1 - F_{\phi_s q}(b(\underline{\lambda_s} - \overline{\lambda_{s'}} + 1))) \tag{4.4}$$

- aborting approach, $\lambda_{s'} = \lambda_{n_\Lambda}$:

$$p^q_{ss'} \approx \Pr(\phi_s, \phi_{s'}) \cdot F_{\phi_s q}(b(\underline{\lambda_s} - \underline{\lambda_{s'}} + 1)) \tag{4.5}$$

- aborting approach, $\lambda_{s'} \neq \lambda_1$ and $\lambda_{s'} \neq \lambda_{n_\Lambda}$:

$$\begin{aligned}
p^q_{ss'} \approx\ & \Pr(\phi_s, \phi_{s'}) \cdot \\
& (F_{\phi_s q}(b(\underline{\lambda_s} - \underline{\lambda_{s'}} + 1)) - F_{\phi_s q}(b(\underline{\lambda_s} - \overline{\lambda_{s'}} + 1)))
\end{aligned} \tag{4.6}$$

- skipping approach, $\lambda_{s'} = \lambda_1$ and $\overline{\lambda_{s'}} < 2$:

$$\begin{aligned} p^q_{ss'} \approx \quad & \Pr(\phi_s, \phi_{s'}) \cdot \\ & (1 - F_{\phi_s q}(b(\underline{\lambda_s} - \overline{\lambda_{s'}} + 1)) \\ & - \sum_{i=1}^{\infty} (F_{\phi_s q}(b(\underline{\lambda_s} - \overline{\lambda_{s'}} + 1 + i)) - F_{\phi_s q}(b(\underline{\lambda_s} - 1 + i)))) \end{aligned}$$

- skipping approach, $\lambda_{s'} = \lambda_1$ and $\overline{\lambda_{s'}} \geq 2$:

$$p^q_{ss'} \approx \quad \Pr(\phi_s, \phi_{s'}) \cdot (1 - F_{\phi_s q}(b(\underline{\lambda_s} - \overline{\lambda_{s'}} + 1)))$$

- skipping approach, $\lambda_{s'} = \lambda_{n_\Lambda}$ and $\underline{\lambda_{s'}} < 2$:

$$\begin{aligned} p^q_{ss'} \approx \quad & \Pr(\phi_s, \phi_{s'}) \cdot \\ & (F_{\phi_s q}(b(\underline{\lambda_s} - \underline{\lambda_{s'}} + 1)) \\ & + \sum_{i=1}^{\infty} (F_{\phi_s q}(b(\underline{\lambda_s} - \underline{\lambda_{s'}} + 1 + i)) - F_{\phi_s q}(b(\underline{\lambda_s} - 1 + i)))) \end{aligned}$$

- skipping approach, $\lambda_{s'} = \lambda_{n_\Lambda}$ and $\underline{\lambda_{s'}} \geq 2$:

$$p^q_{ss'} \approx \quad \Pr(\phi_s, \phi_{s'}) \cdot F_{\phi_s q}(b(\underline{\lambda_s} - \underline{\lambda_{s'}} + 1))$$

- skipping approach, $\lambda_{s'} \neq \lambda_1$ and $\lambda_{s'} \neq \lambda_{n_\Lambda}$ and $\overline{\lambda_{s'}} < 2$:

$$\begin{aligned} p^q_{ss'} \approx \quad & \Pr(\phi_s, \phi_{s'}) \cdot \\ & (F_{\phi_s q}(b(\underline{\lambda_s} - \underline{\lambda_{s'}} + 1)) - F_{\phi_s q}(b(\underline{\lambda_s} - \overline{\lambda_{s'}} + 1)) \\ & + \sum_{i=1}^{\infty} (F_{\phi_s q}(b(\underline{\lambda_s} - \underline{\lambda_{s'}} + 1 + i)) - F_{\phi_s q}(b(\underline{\lambda_s} - \overline{\lambda_{s'}} + 1 + i)))) \end{aligned}$$

- skipping approach, $\lambda_{s'} \neq \lambda_1$ and $\lambda_{s'} \neq \lambda_{n_\Lambda}$ and $\underline{\lambda_{s'}} < 2$ and $\overline{\lambda_{s'}} \geq 2$:

$$\begin{aligned} p^q_{ss'} \approx \quad & \Pr(\phi_s, \phi_{s'}) \cdot \\ & (F_{\phi_s q}(b(\underline{\lambda_s} - \underline{\lambda_{s'}} + 1)) - F_{\phi_s q}(b(\underline{\lambda_s} - \overline{\lambda_{s'}} + 1)) \\ & + \sum_{i=1}^{\infty} (F_{\phi_s q}(b(\underline{\lambda_s} - \underline{\lambda_{s'}} + 1 + i)) - F_{\phi_s q}(b(\underline{\lambda_s} - 1 + i)))) \end{aligned}$$

- skipping approach, $\lambda_{s'} \neq \lambda_1$ and $\lambda_{s'} \neq \lambda_{n_\Lambda}$ and $\underline{\lambda_{s'}} \geq 2$:

$$p^q_{ss'} \approx \quad \Pr(\phi_s, \phi_{s'}) \cdot (F_{\phi_s q}(b(\underline{\lambda_s} - \underline{\lambda_{s'}} + 1)) - F_{\phi_s q}(b(\underline{\lambda_s} - \overline{\lambda_{s'}} + 1)))$$

Clearly, the larger the number of progress intervals, the better the modeling of the state-transition probabilities becomes, as the approximated values for $\lambda_\alpha(f)$ in (4.3) move closer to their real values.

### 4.1.2 Expected revenues

To compute the expected revenues $r^q_{ss'}$, consider processing any frame $f$. Furthermore, let $s$ be the state of the SVA at start point $\alpha(f)$, and let $q$ be the quality level at which frame $f$ is processed. For frame $f$, the reward for the applied quality level and the penalty for changing the quality level follow straightforwardly from state $s$ and quality level $q$, and are given by $R_{\mathrm{ql}}(q)$ and $P_{\mathrm{qlc}}(q_s, q)$, respectively. We now compute the expected number of deadlines misses for frame $f$. Recall from Section 2.3 that $\lambda'_\omega(f)$ denotes the progress of frame $f$ at its milestone, before considering possible deadline misses. If the aborting approach is applied, then at most one deadline is missed for frame $f$. Hence, the expected number of deadline misses is given by the probability of missing deadline $d(f)$:

$$\begin{aligned}\Pr(\lambda'_\omega(f) < 0) &= \{\text{using (2.1)}\}\\ &\quad 1 - \Pr(X_{\phi(f)q(f)} \le b \cdot \lambda_\alpha(f))\\ &= \{\text{using (4.2)}\}\\ &\quad 1 - F_{\phi(f)q(f)}(b \cdot \lambda_\alpha(f)).\end{aligned}$$

If the skipping approach is applied, then multiple deadlines can be missed for frame $f$, and the expected number of deadline misses is given by

$$\begin{aligned}\sum_{i=1}^{\infty} i \cdot \Pr(-i \le \lambda'_\omega(f) < 1-i) &= \{\text{using (2.1)}\}\\ &\quad \sum_{i=1}^{\infty} i \cdot \Pr(X_{\phi(f)q(f)} \le b(\lambda_\alpha(f)+i))\\ &\quad -\sum_{i=1}^{\infty} i \cdot \Pr(X_{\phi(f)q(f)} \le b(\lambda_\alpha(f)+i-1))\\ &= \{\text{using (4.2)}\}\\ &\quad \sum_{i=1}^{\infty} i \cdot F_{\phi(f)q(f)}(b(\lambda_\alpha(f)+i))\\ &\quad -\sum_{i=1}^{\infty} i \cdot F_{\phi(f)q(f)}(b(\lambda_\alpha(f)+i-1)).\end{aligned}$$

Again using (4.3) to approximate $\lambda_\alpha(f)$, for the aborting approach the expected revenues $r^q_{ss'}$ are approximated by

$$r^q_{ss'} \approx R_{\mathrm{ql}}(q) - P_{\mathrm{dm}} \cdot (1 - F_{\phi_s q}(b \cdot \underline{\lambda_s})) - P_{\mathrm{qlc}}(q_s, q),$$

and for the skipping approach they are approximated by

$$r^q_{ss'} \approx R_{\mathrm{ql}}(q) - P_{\mathrm{dm}} \cdot \sum_{i=1}^{\infty} i \cdot (F_{\phi_s q}(b(\underline{\lambda_s} + i)) - F_{\phi_s q}(b(\underline{\lambda_s} + i - 1))) - P_{\mathrm{qlc}}(q_s, q).$$

Note that the expected revenues $r^q_{ss'}$ are independent of state $s'$. Hence, we can denote the expected revenues by $r^q_s$. Due to approximation (4.3), a larger number of progress intervals $n_\Lambda$ results in a more accurate modeling of the expected revenues.

## 4.2 Solving the MDP model

We now discuss how the above-defined MDP model can be solved. First we have to fill in a number of parameters in the model. We assume that the set $Q$ of quality levels, the set $\Phi$ of frame types, the periodic latency $\delta$, the applied deadline miss approach, the budget $b$, and the different revenue parameters $R_{\mathrm{ql}}(q)$, $P_{\mathrm{dm}}$, and $P_{\mathrm{qlc}}(q, q')$ are given by the context of the SVA and the system in which it runs. To define the set of states, we have to choose the number $n_\Lambda$ of progress intervals. As mentioned, the larger $n_\Lambda$ is chosen, the more accurately the state-transition probabilities and expected revenues are approximated. To compute the state-transition probabilities and expected revenues, the probabilities $\Pr(\phi, \phi')$ and the cumulative distribution functions $F_{\phi q}$ must be available. The distribution functions $F_{\phi q}$ can be estimated by processing one or more representative video sequences $n_Q$ times, at each of the $n_Q$ different quality levels, and by collecting processing-time statistics for each frame type $\phi$ and for each quality level $q$. By also collecting statistics on the frame types that appear successively, the probabilities $\Pr(\phi, \phi')$ can be estimated.

Using these parameters, the set of states and the set of actions are defined, and the state-transition probabilities and expected revenues can be computed. Next, the MDP model can be solved, which results in an optimal policy. For the MDP there are in total $n_Q^{n_\Lambda \cdot n_\Phi \cdot n_Q}$ different deterministic policies, of which one or more are optimal. We first give an example to illustrate how the state-transition probabilities and expected revenues are computed, and how an optimal policy can be derived.

**Example 4.1.** Consider an SVA that can process frames at two different quality levels, $q_1$ and $q_2$. The frames do not vary in type, i.e., all frames are of type $\phi_1$. To handle deadline misses we apply the aborting approach. Let the periodic latency and the budget be given by $\delta = 2$ and $b = 40$ ms, respectively. To define the set of states we choose $n_\Lambda = 4$ progress intervals: $\lambda_1 = [1, 1.25)$, $\lambda_2 = [1.25, 1.5)$, $\lambda_3 = [1.5, 1.75)$, and $\lambda_4 = [1.75, 2]$. To estimate the cumulative distribution functions we use the processing times of 100 frames. Applying quality level $q_1$, the frames result in the following processing times: 18 ms (10 frames), 28 ms (20 frames), 32 ms (25 frames), 38 ms (5 frames), 42 ms (10 frames), 48 ms (10 frames), and 52 ms (20

frames). Applying quality level $q_2$, the processing time for each frame is exactly 5 ms higher. Let the revenue parameters be given by $R_{\mathrm{ql}}(q_1) = 0$, $R_{\mathrm{ql}}(q_2) = 5$, and $P_{\mathrm{dm}} = 20$. For simplicity we assume that quality-level changes are not penalized.

Because frames do not vary in type and quality-level changes are not penalized, we can reduce the set of states to the set of progress intervals only. Let random variable $X_q$ denote the processing time of a single frame at quality level $q$, and let $F_q$ denote the cumulative distribution function of $X_q$, i.e., $F_q(x) = \Pr(X_q \leq x)$. Because frames do not vary in type, the probabilities $\Pr(\phi_s, \phi_{s'})$ in (4.4)–(4.6) are all one, and distribution function $F_{\phi_1 q}$ is given by $F_q$. Applying (4.6), state-transition probability $p^{q_1}_{\lambda_2\lambda_3}$, for example, is given by $F_{q_1}(40(1.25 - 1.5 + 1)) - F_{q_1}(40(1.25 - 1.75 + 1)) = F_{q_1}(30) - F_{q_1}(20)$. To compute this, we observe that 30 out of 100 frames have a processing time of at most 30 ms at quality level $q_1$, and 10 out of 100 frames have a processing time of at most 20 ms. Hence $p^{q_1}_{\lambda_2\lambda_3} = 0.2$. The other state-transition probabilities and expected revenues are estimated analogously, and are given in Table 4.1.

Table 4.1. The state-transition probabilities and expected revenues.

| | $p^{q_1}_{\lambda_i\lambda_1}$ | $p^{q_1}_{\lambda_i\lambda_2}$ | $p^{q_1}_{\lambda_i\lambda_3}$ | $p^{q_1}_{\lambda_i\lambda_4}$ | $p^{q_2}_{\lambda_i\lambda_1}$ | $p^{q_2}_{\lambda_i\lambda_2}$ | $p^{q_2}_{\lambda_i\lambda_3}$ | $p^{q_2}_{\lambda_i\lambda_4}$ | $r^{q_1}_{\lambda_i}$ | $r^{q_2}_{\lambda_i}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| $i = 1$ | 0.7 | 0.2 | 0.1 | 0.0 | 0.9 | 0.1 | 0.0 | 0.0 | −8.0 | −4.0 |
| $i = 2$ | 0.4 | 0.3 | 0.2 | 0.1 | 0.45 | 0.45 | 0.1 | 0.0 | −4.0 | −1.0 |
| $i = 3$ | 0.2 | 0.2 | 0.3 | 0.3 | 0.3 | 0.15 | 0.45 | 0.1 | 0.0 | 5.0 |
| $i = 4$ | 0.0 | 0.2 | 0.2 | 0.6 | 0.0 | 0.3 | 0.15 | 0.55 | 0.0 | 5.0 |

At each start point of a frame, the state of the SVA is given by one of the four progress intervals, and given the state the controller selects either quality level $q_1$ or $q_2$. Hence, there are $2^4 = 16$ different policies, of which at least one is optimal. Let $(q_1, q_1, q_2, q_2)$ denote the policy of selecting quality level $q_1$ in states $\lambda_1$ and $\lambda_2$, and quality level $q_2$ in states $\lambda_3$ and $\lambda_4$. Using the state-transition probabilities, this policy results in the Markov chain depicted in Figure 4.1.

Let $p(\lambda)$ denote the (stationary) probability that the SVA's state at the start point of a frame is given by progress interval $\lambda$. The Markov chain for policy $(q_1, q_1, q_2, q_2)$ results in the following set of equations for $p(\lambda_1)$ to $p(\lambda_4)$.

$$\begin{cases} 1 = p(\lambda_1) + p(\lambda_2) + p(\lambda_3) + p(\lambda_4) \\ p(\lambda_1) = 0.7p(\lambda_1) + 0.4p(\lambda_2) + 0.3p(\lambda_3) \\ p(\lambda_2) = 0.2p(\lambda_1) + 0.3p(\lambda_2) + 0.15p(\lambda_3) + 0.3p(\lambda_4) \\ p(\lambda_3) = 0.1p(\lambda_1) + 0.2p(\lambda_2) + 0.45p(\lambda_3) + 0.15p(\lambda_4) \\ p(\lambda_4) = 0.1p(\lambda_2) + 0.1p(\lambda_3) + 0.55p(\lambda_4) \end{cases}$$

Solving this set of equations yields $p(\lambda_1) = \frac{498}{1015}$, $p(\lambda_2) = \frac{225}{1015}$, $p(\lambda_3) = \frac{198}{1015}$, and $p(\lambda_4) = \frac{94}{1015}$. Hence, the expected average revenue per frame of policy

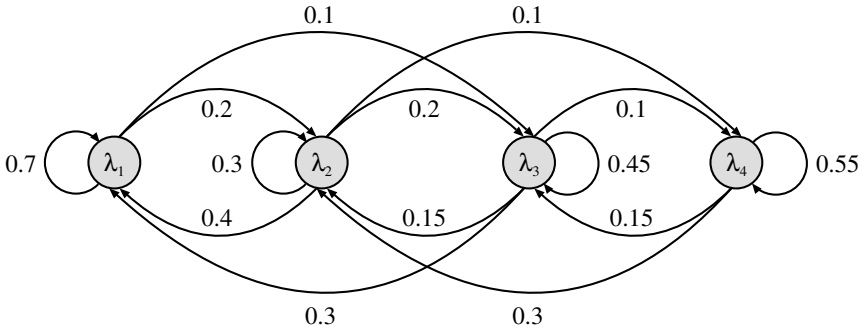


Figure 4.1. The Markov chain for policy $(q_1, q_1, q_2, q_2)$.

Table 4.2. The expected average revenues of the different policies.

| policy | expected average revenue |
|---|---|
| $(q_1, q_1, q_1, q_1)$ | $-1096/261 \approx -4.20$ |
| $(q_1, q_1, q_1, q_2)$ | $-2150/599 \approx -3.59$ |
| $(q_1, q_1, q_2, q_1)$ | $-1684/455 \approx -3.70$ |
| $(q_1, q_1, q_2, q_2)$ | $-3424/1015 \approx -3.37$ |
| $(q_1, q_2, q_1, q_1)$ | $-1828/435 \approx -4.20$ |
| $(q_1, q_2, q_1, q_2)$ | $-446/115 \approx -3.88$ |
| $(q_1, q_2, q_2, q_1)$ | $-476/125 \approx -3.81$ |
| $(q_1, q_2, q_2, q_2)$ | $-424/115 \approx -3.69$ |
| $(q_2, q_1, q_1, q_1)$ | $-520/153 \approx -3.40$ |
| $(q_2, q_1, q_1, q_2)$ | $-1118/359 \approx -3.11$ |
| $(q_2, q_1, q_2, q_1)$ | $-188/59 \approx -3.19$ |
| $(q_2, q_1, q_2, q_2)$ | $-668/221 \approx -3.02$ |
| $(q_2, q_2, q_1, q_1)$ | $-225/68 \approx -3.31$ |
| $(q_2, q_2, q_1, q_2)$ | $-178/55 \approx -3.24$ |
| $(q_2, q_2, q_2, q_1)$ | $-408/127 \approx -3.21$ |
| $(q_2, q_2, q_2, q_2)$ | $-3746/1175 \approx -3.19$ |

$(q_1, q_1, q_2, q_2)$ is given by $p(\lambda_1)r_{\lambda_1}^{q_1} + p(\lambda_2)r_{\lambda_2}^{q_1} + p(\lambda_3)r_{\lambda_3}^{q_2} + p(\lambda_4)r_{\lambda_4}^{q_2} = -\frac{3424}{1015} \approx -3.37$. The expected average revenues of all 16 policies are given in Table 4.2. It follows that policy $(q_2, q_1, q_2, q_2)$ is the only optimal policy, having an expected average revenue of $-\frac{668}{221} \approx -3.02$. Note that, although choosing quality level $q_2$ in a state always results in a higher immediate revenue than choosing quality level $q_1$, in state $\lambda_2$ it is apparently better to choose quality level $q_1$. □

To find an optimal policy without exhaustive search, we can apply, amongst others, the policy iteration algorithm, the value iteration algorithm, or the successive approximation algorithm, as discussed in Section 3.3. The computation time of these algorithms grows roughly quadratically with the number of states. Hence, when choosing the number of progress intervals, one has to make a trade-off between the accuracy of a solution and the time required to find it.

## 4.3 Off-line strategy

An optimal policy can as follows serve as a strategy for the controller. At each start point of a frame $f$, the SVA calls the controller to obtain quality level $q(f)$. In this call the SVA provides progress $\lambda_\alpha(f)$ and frame type $\phi(f)$ to the controller. The controller uses this information to determine the state of the SVA. The state is given by the progress interval corresponding to $\lambda_\alpha(f)$, the previous quality level for frame $f$, and frame type $\phi(f)$. To determine the previous quality level, the controller has to remember the quality level selected for the last-processed frame. As mentioned earlier, for frame 1 we assume a previous quality level $q_1$. Given the state, the controller looks up the policy to select the quality level, and next it returns the selected quality level to the SVA. We call this control strategy the *off-line strategy*, because the applied policy is computed off line (i.e., before we run the SVA) and it is not changed at run time.

An important issue is the run-time overhead (or processing-time overhead) imposed by the off-line strategy. Assuming a video signal with a picture resolution of $720 \times 576$ pixels, for each frame there are 414,720 pixels to be computed. Using the off-line strategy, at each start point of a frame the controller successively has to determine the state of the SVA, select the quality level for the frame by performing a table lookup, update an internal register to store the selected quality level, and return the selected quality level to the SVA. Since this all can be done in a small number of computational steps, we consider the run-time overhead of the off-line strategy to be low.

Another important issue is the storage overhead imposed by the off-line strategy. A policy has a space complexity of $\mathcal{O}(n_\Lambda \cdot n_\Phi \cdot n_Q \cdot \log(n_Q))$. The storage overhead of the off-line strategy can be reduced by considering only monotonic policies. We call a policy monotonic if, for each integer $i \in \{2, \ldots, n_\Lambda\}$, for each frame type $\phi \in \Phi$, and for each previous quality level $q \in Q$, the quality level to be chosen in state $(\lambda_i, \phi, q)$ is the same or higher than the quality level to be chosen in state $(\lambda_{i-1}, \phi, q)$. In practice, optimal policies are usually monotonic, but this is no strict rule. We constructed Example 4.1 especially to show that an optimal policy can be non-monotonic. For a monotonic policy, per frame type and previous quality level only $n_Q$ progress intervals have to be stored, viz. for each quality-level

action the highest progress interval with that particular action. Hence, a monotonic policy has a space complexity of $\mathcal{O}(n_\Phi \cdot n_Q^2 \cdot \log(n_\Lambda))$. In practice, the number $n_\Lambda$ of progress intervals is usually chosen much larger than the number $n_Q$ of quality levels.

So, to relax the storage overhead imposed by the off-line strategy, we only use monotonic policies. In case an optimal policy turns out to be non-monotonic, we transform it as follows into a monotonic one. For each frame type $\phi \in \Phi$ and for each quality level $q \in Q$, we pairwise consider the progress intervals $\lambda_{i-1}$ and $\lambda_i$, for $i = 2, \ldots, n_\Lambda$. If the quality level to be chosen in state $(\lambda_{i-1}, \phi, q)$ is higher than the quality level to be chosen in state $(\lambda_i, \phi, q)$, then we change the quality level to be chosen in state $(\lambda_i, \phi, q)$ to the one of state $(\lambda_{i-1}, \phi, q)$. In general, this transformation is at the cost of losing optimality. However, we take possible performance losses for granted, as they are usually small in practice. In Example 4.1, for example, the optimal policy $(q_2, q_1, q_2, q_2)$, with an expected average revenue of $-3.02$, would be transformed into the monotonic policy $(q_2, q_2, q_2, q_2)$, with an expected average revenue of $-3.19$.

Figure 4.2 shows an example monotonic policy for an SVA with four quality levels and a periodic latency $\delta = 3$, and for $n_\Lambda = 8$ progress intervals. For simplicity, we assume that frames do not vary in type. The figure has four columns, one for each previous quality level. Each possible state of the SVA at the start point of a frame is given by a particular progress interval in a particular column. The gray scale indicates the quality level to be chosen. For example, if the progress at the start point of a frame is 2.6 and the previous quality level is $q_2$, then the off-line strategy selects quality level $q_3$.

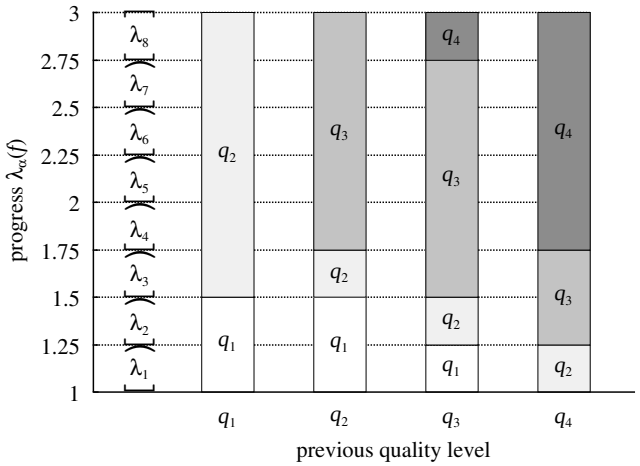


Figure 4.2. An example monotonic policy.

## 4.4 Simulation experiments

To assess the effectiveness of the off-line strategy, and to study the behavior of the strategy, we have run simulation experiments. These experiments are based on processing-time statistics of a scalable MPEG-2 decoder running on a TriMedia 1300 (180 MHz) processor [Rathnam and Slavenburg, 1996]. The decoder was made available to us by colleagues at Philips Research [Peng, 2001; Zhong *et al.*, 2002]. We used the decoder in the setting of a demonstrator annex research platform [Otero Pérez and Nitescu, 2002; Hentschel *et al.*, 2003]. Given an MPEG-2 encoded video stream as input, the decoder sequentially processes (decodes) the frames of the stream. The decoder can process frames at four quality levels, in increasing quality order named $q_1$ to $q_4$.

In the decoder, scalability is achieved by pruning input data of the Inverse Discrete Cosine Transform (IDCT) process. This transformation reconstructs picture data from a set of coefficients. If a fraction of the least important coefficients are not used in the transformation (the pruning), then the picture data can still be reconstructed, albeit at the cost of a lower visual quality. The fewer coefficients that are used in the transformation, the smaller is the required number of computations.

### Traces

To collect data for the experiments, we selected 18 MPEG-2 sequences from DVD; see Table 4.3. The 18 sequences all have a frame rate of 25 fps. Using the decoder, we processed each sequence four times, one time at each of the four different quality levels, and we measured the per-frame processing times. The sequences were processed without imposing deadlines on frames. For each sequence we created a so-called trace, a table in which we collected the measured processing time. The successive rows of a trace correspond to the successive frames of the sequence, and there is a column for each quality level. There is also a column for storing the MPEG-2 frame type (I, P, or B). We labeled the 18 traces with the letters A to R. Table 4.3 provides for each trace the title of the source DVD, the number of frames in the sequence, and the average processing time per frame for quality levels $q_1$ to $q_4$.

By concatenating traces A to R in alphabetical order, we also created one large trace of in total 2,997,326 frames. At a frame rate of 25 fps, this trace corresponds to over 33 hours of video. We labeled the concatenated trace with the name CONCAT. Figure 4.3 shows for each quality level a cumulative distribution function of the time required to process a single frame, based on the processing times of trace CONCAT. A small fraction of the frames have a processing time lower than 10 ms or higher than 45 ms, which is not visible in the figure.

Table 4.3. The title of the source DVD, the number of frames, and the average processing time per frame for quality levels $q_1$ to $q_4$, for the various traces used in the experiments.

| trace | title of source DVD | #frames | average processing time per frame (ms) $q_1$ | $q_2$ | $q_3$ | $q_4$ |
|---|---|---|---|---|---|---|
| A | 'Allo 'Allo series 1 & 2 (disc 1) | 230,936 | 23.2 | 24.1 | 25.4 | 28.0 |
| B | 'Allo 'Allo series 1 & 2 (disc 2) | 186,560 | 23.9 | 24.7 | 26.0 | 28.8 |
| C | 'Allo 'Allo series 1 & 2 (disc 3) | 232,461 | 23.9 | 24.8 | 25.9 | 28.9 |
| D | Amsterdamned | 162,900 | 22.9 | 23.7 | 24.7 | 28.0 |
| E | Antz | 119,233 | 22.5 | 23.2 | 24.2 | 26.4 |
| F | De Lift | 141,739 | 21.3 | 21.8 | 22.6 | 25.3 |
| G | Falling Down | 161,842 | 22.7 | 23.7 | 25.4 | 27.3 |
| H | Flodder | 164,158 | 20.7 | 21.2 | 22.2 | 24.5 |
| I | The Very Best of the Muppet Show | 151,363 | 22.9 | 23.6 | 24.6 | 26.8 |
| J | Pet Shop Boys – Montage | 166,740 | 25.0 | 25.8 | 28.3 | 30.3 |
| K | Passenger 57 | 120,164 | 23.8 | 24.7 | 26.7 | 29.1 |
| L | The Leaning Tower of Pisa | 75,075 | 21.7 | 22.3 | 23.2 | 25.5 |
| M | PSV Hoogtepunten | 138,987 | 21.3 | 22.4 | 23.3 | 24.9 |
| N | Pet Shop Boys – Somewhere | 136,560 | 22.2 | 22.9 | 24.2 | 27.2 |
| O | Van Kooten & De Bie 7 | 161,057 | 22.2 | 23.1 | 24.6 | 27.0 |
| P | Violent City | 162,287 | 19.4 | 20.1 | 20.9 | 22.4 |
| Q | The World is not Enough | 184,035 | 20.5 | 21.2 | 22.4 | 24.2 |
| R | Yes Minister series 1 | 301,229 | 21.5 | 22.2 | 23.0 | 24.8 |
| CONCAT | — | 2,997,326 | 22.3 | 23.1 | 24.3 | 26.7 |
| ARTIFICIAL | — | 5,000,000 | 22.3 | 23.1 | 24.3 | 26.7 |

As mentioned earlier, in the MDP model it is implicitly assumed that the processing times of successive frames are independent. To assess the off-line strategy using data that satisfies this assumption, we created a trace named ARTIFICIAL by independently drawing 5,000,000 real numbers from interval $(0, 1)$, and by looking up for each number the processing time at each of the four quality levels in the distribution functions of Figure 4.3. In trace ARTIFICIAL we do not distinguish between different frame types.

### Settings

In the MDP model we use the following default settings for the various parameters. These settings are varied in Sections 4.4.2 till 4.4.5.

- We use a periodic latency $\delta = 3$, which implies that the SVA can work ahead by at most two periods $P$; other values of $\delta$ are evaluated in Section 4.4.2. Because the used MPEG-2 sequences all have a frame rate of 25 fps, we have

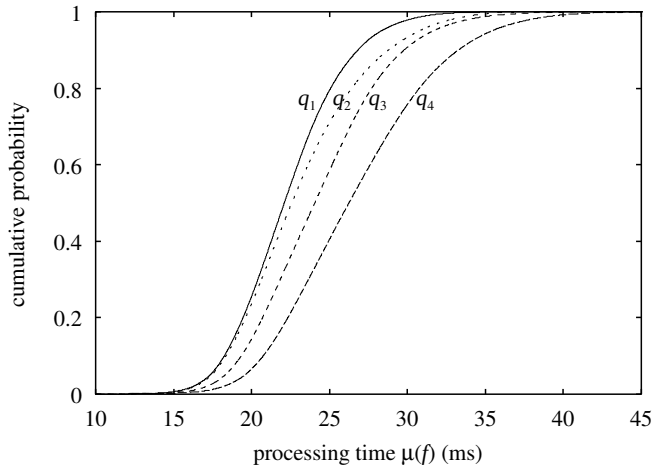


Figure 4.3. A cumulative distribution function of the time required to process a single frame, for each quality level of the scalable MPEG-2 decoder, based on the processing times of trace CONCAT.

$P = 40$ ms.

- To handle deadline misses we apply the skipping approach; the difference with the aborting approach is evaluated in Section 4.4.3.
- We use $n_\Lambda = 300$ progress intervals; other values of $n_\Lambda$ are evaluated in Section 4.4.4.
- Based on a discussion with experts in the video domain, we define the revenue parameters as follows. The rewards for processing a frame at a particular quality level are given by $R_{\mathrm{ql}}(q_1) = 4$, $R_{\mathrm{ql}}(q_2) = 6$, $R_{\mathrm{ql}}(q_3) = 8$, and $R_{\mathrm{ql}}(q_4) = 10$. The deadline-miss penalty is set to $P_{\mathrm{dm}} = 10,000$, which roughly means that we allow at most one deadline miss per 1000 frames, given $R_{\mathrm{ql}}(q_4) = 10$; other values of the deadline-miss penalty are evaluated in Section 4.4.5. The penalties $P_{\mathrm{qlc}}$ for changing the quality level between two successively processed frames are given by 10, 100, and 1000 for increasing or decreasing the quality level by one, two, or three levels, respectively.

To solve the MDP model, we use the successive approximation algorithm with a convergence error $\varepsilon = 0.001$; see Section 3.3. This algorithm computes an optimal policy for the MDP, and a corresponding expected average revenue per frame. As mentioned earlier, if the computed policy is non-monotonic, then we transform it into a monotonic one.

We use OFF($\tau$) as a shorthand notation for the off-line strategy, using a policy that was computed for the above-mentioned default parameter settings and for distribution functions $F_{\phi q}$ derived from the processing times of a trace $\tau$. We call trace $\tau$ the statistics trace. Strategy OFF($\tau$) can use multiple policies, computed for different values of budget $b$. For any change in the parameter settings with respect to the default settings we extend the shorthand notation. For example, if we use the aborting approach instead of the skipping approach in the MDP model, then we denote the strategy by OFF($\tau$, aborting approach).

To assess the off-line strategy, or any other control strategy, for different values of budget $b$ we simulate that the SVA processes a sequence of MPEG-2 frames, based on the processing model described in Chapter 2. The control strategy is used to select the quality level for each frame, and the processing times of the successive frames in the sequence are taken from a trace $\tau$. We call this a simulation of the control strategy on trace $\tau$, and we call trace $\tau$ the simulation trace. In every simulation we measure various performance aspects, such as the number of frames that are processed at each quality level, the number of deadline misses, the number and size of the quality-level changes between successively processed frames, and the average revenue received per processed frame.

As a rule, the parameters that are used in a simulation always match the parameters that were used to derive the applied control strategy. For example, in a simulation of strategy OFF(CONCAT, $\delta = 4$) on a trace $\tau$ we also use a periodic latency of 4. Also, if we use a budget of 30 ms in the simulation, then the off-line strategy uses a policy that was computed for the exactly same budget. The only exception to this rule is that the used statistics trace can differ from the used simulation trace.

To evaluate a particular parameter setting for the off-line strategy, or for any other control strategy in one of the subsequent chapters, we always use trace CONCAT as statistics trace. This is because trace CONCAT provides a kind of average over the traces A to R, which mutually vary in processing complexity. In Section 6.6.4 we discuss whether trace CONCAT is indeed a good choice. As simulation trace we mainly use trace ARTIFICIAL. Because this trace does not contain frame type information, in the MDP model we assume that all frames have the same type. In Section 5.4.3 we evaluate the usage of different frame types in the MDP model.

**Experiment overview**

In Section 4.4.1 we assess the off-line strategy for the default parameter settings. Next, in Section 4.4.2 we vary the periodic latency, in Section 4.4.3 we vary the deadline miss approach, in Section 4.4.4 we vary the number of progress intervals, and in Section 4.4.5 we vary the deadline miss penalty. Finally, in Section 4.4.6 we

use trace CONCAT instead of trace ARTIFICIAL as simulation trace.

### 4.4.1 Results for the default parameter settings

As an initial experiment, we applied strategy OFF(CONCAT) in 61 simulations on trace ARTIFICIAL, for budgets 10 ms, 10.5 ms, ..., 40 ms. Figures 4.4, 4.5, 4.6, and 4.7 show the average revenue per frame, the number of deadline misses, the average increase in quality level per frame, and the number of frames processed at each quality level, respectively, as a function of the budget, as measured in the different simulations. The figures only show results over the budget range 20 ms – 30 ms. We do not show a graph of the average decrease in quality level, as it is almost identical to the average increase in quality level. This is because the sum of all quality level increases and the sum of all quality level decreases in a simulation can differ at most $n_Q - 1 = 3$.

For low values of the budget we observe that the strategy only selects quality level $q_1$, and that the average revenue is strongly negative, which is due to unavoidable deadline misses and the relatively high deadline miss penalty. For each missed deadline one frame is skipped. The larger the budget, the better the SVA is capable of meeting deadlines. Hence, as the budget increases, the number of frames that are processed at $q_1$ first grows towards the number of frames of trace ARTIFICIAL, which is 5,000,000. At $b = 23$ ms, the number of deadline misses is small enough (1604) to result in a positive average revenue (0.79). At $b = 23.5$ ms, the controller starts to select $q_2$ as well, and the number of deadline misses drops further to 313. As the budget further increases, the most frequently selected quality level moves gradually to $q_4$, and the average revenue converges to the maximum value of 10. This maximum value of 10 is due to the reward of 10 for processing a frame at quality level $q_4$, and the vanishing of both deadline misses and quality-level changes. From $b = 30$ ms onwards the number of deadline misses is zero, and almost all frames are processed at $q_4$. In Figure 4.6, the peaks at $b = 23.5$ ms, $b = 24.5$ ms and $b = 26$ ms correspond to the budgets at which the strategy starts using a higher quality level as well.

To give an impression of the different constituents of the average revenue, consider the simulation for $b = 25$ ms. In this simulation, we measure that 62,278 frames are processed at $q_1$, 277,975 frames are processed at $q_2$, and 4,659,709 frames are processed at $q_3$. We measure in total 38 deadline misses. The quality level is increased 42,892 times by one level, decreased 42,516 times by one level, and decreased 187 times by two levels. This results in an average revenue of $(62,278 \cdot 4 + 277,975 \cdot 6 + 4,659,709 \cdot 8 - 38 \cdot 10,000 - (42,892 + 42,516) \cdot 10 - 187 \cdot 100)/4,999,962 \approx 7.59$.

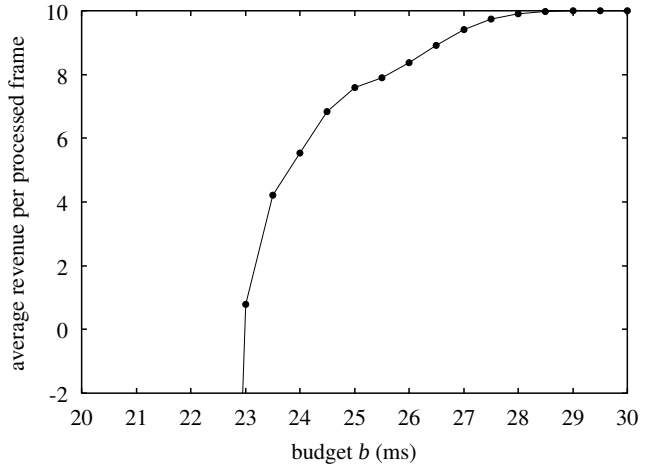


Figure 4.4. The average revenue as a function of the budget, for strategy OFF(CONCAT) applied in simulations on trace ARTIFICIAL.

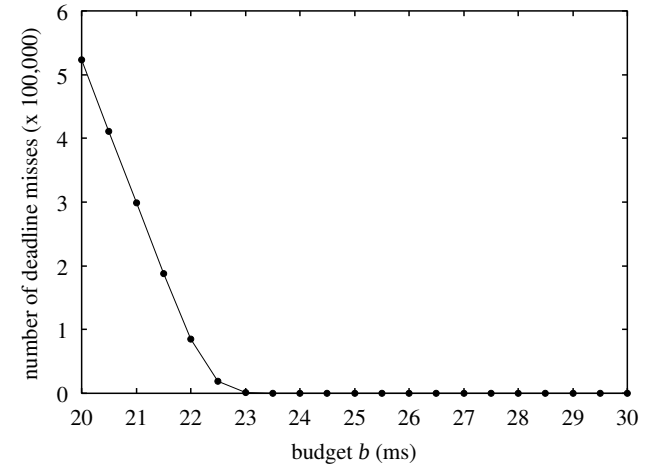


Figure 4.5. The number of deadline misses as a function of the budget, for strategy OFF(CONCAT) applied in simulations on trace ARTIFICIAL.

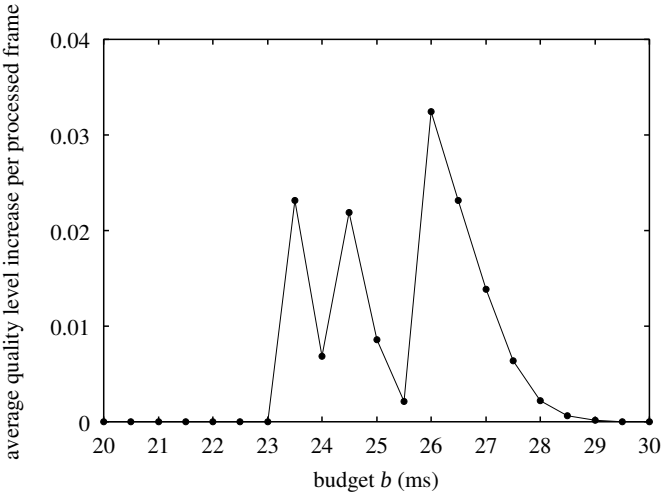


Figure 4.6. The average increase in quality level as a function of the budget, for strategy OFF(CONCAT) applied in simulations on trace ARTIFICIAL.

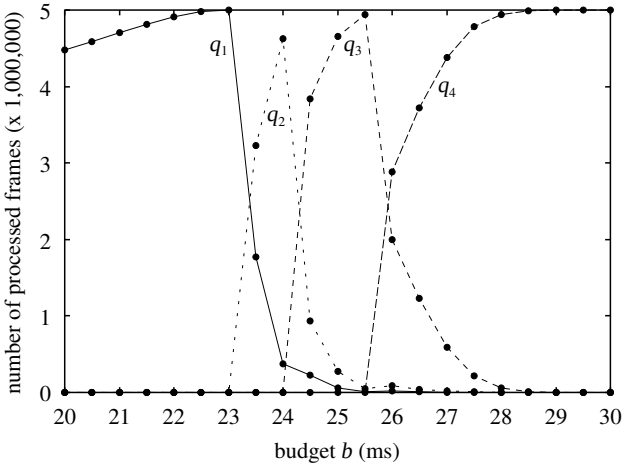


Figure 4.7. The number of frames processed at each quality level as a function of the budget, for strategy OFF(CONCAT) applied in simulations on trace ARTIFICIAL.

### 4.4.2 Varying the latency

To study the influence of the latency on the results we obtained for the default parameter settings, we applied the strategies OFF(CONCAT, $\delta = 2$) to OFF(CONCAT, $\delta = 6$) each in 61 simulations on trace ARTIFICIAL, for budgets 10 ms, 10.5 ms, ..., 40 ms. Note that strategy OFF(CONCAT, $\delta = 3$) is the same as strategy OFF(CONCAT). Figure 4.8 shows the average revenue per frame that we measured in the simulations for the different strategies, as a function of the budget.

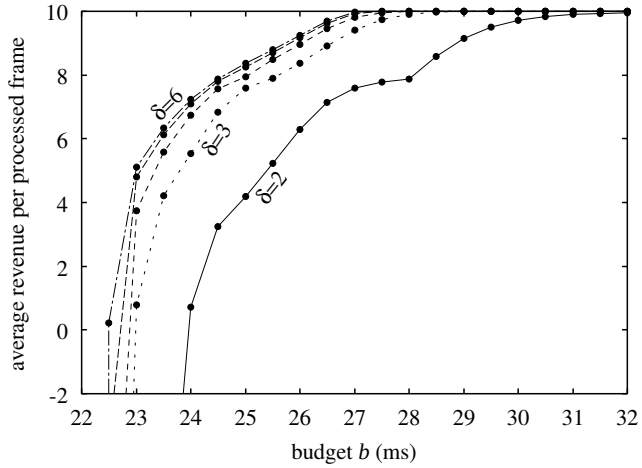


Figure 4.8. The average revenue for strategies OFF(CONCAT, $\delta = 2$) to OFF(CONCAT, $\delta = 6$) applied in simulations on trace ARTIFICIAL, as a function of the budget.

We see that a higher latency results in a higher average revenue, which is logical because a higher latency gives more space to the SVA to even out the varying load of successive frames. As a result, larger bursts in the load of frames can be handled without missing a deadline. Second, we see that the increase in average revenue for successive values of $\delta$ becomes smaller, as the numbers of bursts that cannot be handled properly, decreases (rapidly). Additionally, the smaller $\delta$, the stronger the off-line strategy has to reduce its quality level if the progress decreases, to prevent deadline misses.

For some systems a high latency is unacceptable. This is for example the case when the number of buffers is required to be small, due to a limited amount of system memory. Another example is when a video stream and a corresponding audio stream are processed using independent devices. In that case, using a high latency for video processing may cause the audio stream and video stream to run out of sync. User perception studies show that for lip synchronization a delay of 80 ms

between audio and video (audio ahead of video, or video ahead of audio) generally remains undetected, whereas a delay of 160 ms is considered to be unacceptable [Steinmetz, 1996]. However, for many systems it is not that hard to keep audio and video synchronized, in which case a high latency can be allowed.

From the above experiment we do not see a clear benefit of choosing the latency very high. To give the SVA some space to work ahead, we use $\delta = 3$ as default value for the periodic latency, i.e., the SVA can work ahead by at most two periods. At a frame rate of 25 fps, a lip synchronization delay of 80 ms corresponds to two video frames.

### 4.4.3 Varying the deadline miss approach

To study the influence of the deadline miss approach on the results we obtained for the default parameter settings, we applied the strategies OFF(CONCAT, aborting approach) and OFF(CONCAT, skipping approach) each in 61 simulations on trace ARTIFICIAL, for budgets 10 ms, 10.5 ms, ..., 40 ms. Note that strategy OFF(CONCAT, skipping approach) is the same as strategy OFF(CONCAT). Figures 4.9 and 4.10 show the average revenue per frame and the number of deadline misses, respectively, that we measured in the simulations for both strategies, as a function of the budget.

For both strategies we observe that deadlines are missed up to and including a budget of 29.5 ms. For smaller budgets we observe that the skipping approach results in fewer deadline misses than the aborting approach, and therefore in a higher average revenue.

Applying the skipping approach, if the deadline of a frame is missed, then a new deadline is assigned to the frame, and a later frame is skipped. Because for small budgets many deadlines are missed, also many frames are skipped. However, even for small budgets the number of deadline misses is considerably smaller than the total number of frames, which is 5,000,000. Due to the work-preserving nature of the skipping approach, still a reasonable number of frames are completed. For budgets up to and including 23 ms the strategy only selects quality level $q_1$, and the SVA tries to complete as many frames as possible within the given time. As a result, the number of deadline misses decreases roughly linearly as a function of the budget. At $b = 23.5$ ms, when the strategy starts to trade-off a higher quality level for deadline misses and quality-level changes, the number of deadline misses does not immediately drop to zero, but starts decreasing at a lower rate. At $b = 30$ ms the number of deadline misses becomes zero.

Applying the aborting approach, if the deadline of a frame is missed, then the frame is aborted and all work is lost. Because the aborting approach is not work-preserving, for small budgets the number of deadline misses approaches the total number of frames. In a worst-case situation a frame's deadline falls right before the

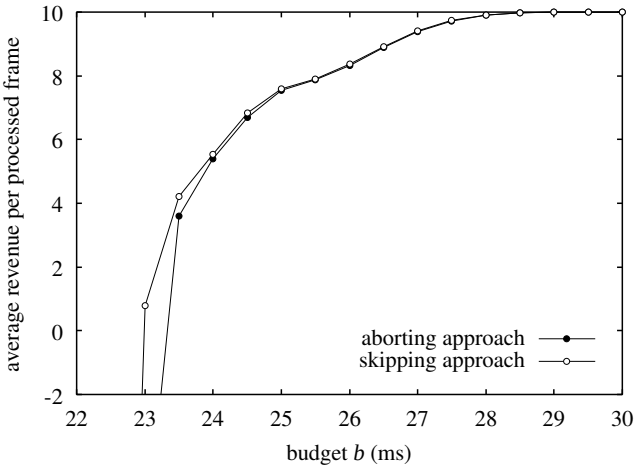


Figure 4.9. The average revenue for strategies OFF(CONCAT, aborting approach) and OFF(CONCAT, skipping approach) applied in simulations on trace ARTIFICIAL, as a function of the budget.

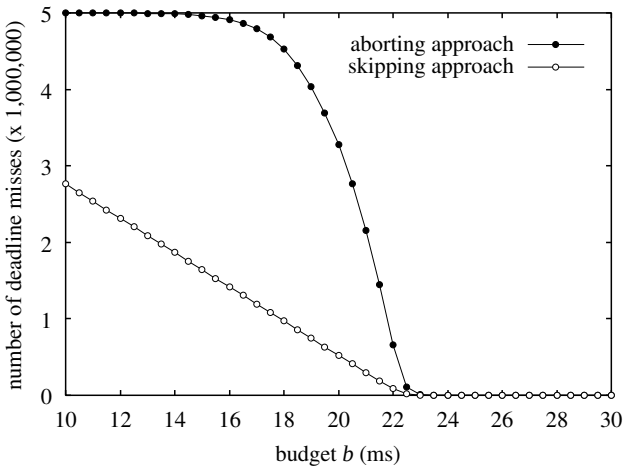


Figure 4.10. The number of deadline misses for strategies OFF(CONCAT, aborting approach) and OFF(CONCAT, skipping approach) applied in simulations on trace ARTIFICIAL, as a function of the budget.

frame is completed, which means that a lot of processing time is wasted. The shape of the graph of the number of deadline misses can be explained as follows. From (2.1) – (2.5) it can be derived that if a frame is aborted, then the next frame to be processed has a progress of exactly one at the start point. If the progress of a frame at its start point is one, then the SVA has exactly $b$ time to process the frame without missing the frame's deadline. If all frames would have a progress of one at the start point, then for increasing values of the budget the number of deadline misses would decrease roughly according to the distribution of the processing times of the frames. For small budgets many deadlines are missed, which means that many frames indeed have a progress of one at the start point. As the budget increases, the number of deadline misses decreases, which means that the number of frames with a progress of one at the start point also decreases. Hence, as the budget increases, the number of deadline misses starts decreasing more linearly, as for the skipping approach. At $b = 24$ ms, the controller starts to trade-off a higher quality level for deadline misses and quality-level changes. From this budget onwards the strategy behaves roughly similarly to the strategy for the skipping approach.

### 4.4.4 Varying the number of progress intervals

To study the influence of the number $n_\Lambda$ of progress intervals on the results we obtained for the default parameter settings, we have chosen twelve different values of $n_\Lambda$: 25, 40, 60, 90, 135, 200, 300, 450, 675, 1000, 1500, and 2250. In succession, these values increase roughly by a factor of 1.5. For each value of $n_\Lambda$ we applied strategy OFF(CONCAT, $n_\Lambda$) in 61 simulations on trace ARTIFICIAL, for budgets $10\,\text{ms}, 10.5\,\text{ms}, \ldots, 40\,\text{ms}$. Note that strategy OFF(CONCAT, $n_\Lambda = 300$) is the same as strategy OFF(CONCAT). Figure 4.11 shows the computation time of successive approximation[1], the expected average revenue per frame of strategy OFF(CONCAT, $n_\Lambda$) as computed by successive approximation, and the average revenue per frame for strategy OFF(CONCAT, $n_\Lambda$) applied in a simulation on trace ARTIFICIAL, as a function of $n_\Lambda$, for $b = 27$ ms. The computation time of successive approximation includes the time needed to compute the required state-transition probabilities and expected revenues beforehand.

We observe that the computation time of successive approximation grows roughly quadratically with $n_\Lambda$. This computation time is not a real point of concern, because the algorithm is applied before run time, and we are primarily interested in a low run-time overhead of the controller. Nevertheless, this observation gives reason to not choose the number of progress intervals unnecessarily large.

In the figure, we observe that for an increasing number of progress intervals

[1] The computation time was measured on a Toshiba Satellite 1410-304 personal computer (Mobile Intel Celeron 1.8 GHz processor, 512 MB RAM). The successive approximation algorithm was implemented using Microsoft Visual C++ 6.0.

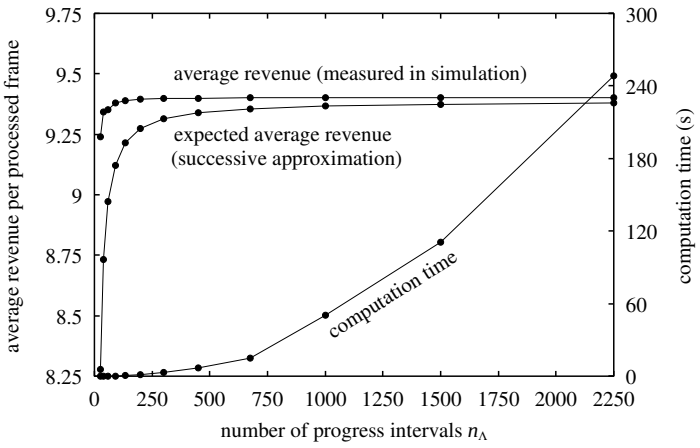


Figure 4.11. The computation time of successive approximation (right axis), the expected average revenue of strategy OFF(CONCAT, $n_\Lambda$) as computed by successive approximation (left axis), and the average revenue for strategy OFF(CONCAT, $n_\Lambda$) applied in a simulation on trace ARTIFICIAL (left axis), as a function of $n_\Lambda$, for $b = 27$ ms.

the expected average revenue becomes less pessimistic. This pessimism is due to worst-case approximation (4.3) in the MDP model. The average revenue measured in the simulations is persistently higher than the expected average revenue, and the distance between the two becomes smaller as the number of progress intervals increases. The average revenue measured in the simulations quickly converges to a value of approximately 9.4, and from $n_\Lambda = 300$ progress intervals onwards this value does not significantly increase anymore. Even though the expected average revenue has not yet converged at $n_\Lambda = 300$, this number of progress intervals already results in a nearly optimal strategy for the simulation trace. For the other 60 values of the budget we observe a similar convergence result. For this reason, we have selected $n_\Lambda = 300$ as the default number of progress intervals.

### 4.4.5 Varying the revenue parameters

The deadline miss penalty has the highest value of all revenue parameters. Initial tests showed that the effects of varying the quality-level rewards and the quality-level change penalties are marginal, in the sense that all settings work similarly well. Hence, in this section we discuss only the effect of changing the deadline miss penalty.

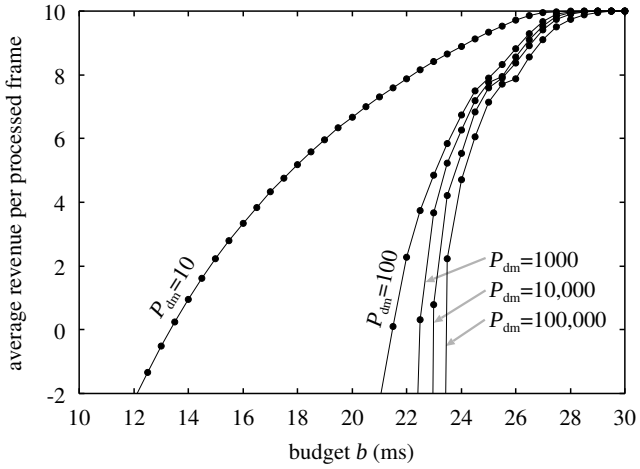


Figure 4.12. The average revenue for different strategies OFF(CONCAT, $P_{\mathrm{dm}}$) applied in simulations on trace ARTIFICIAL, as a function of the budget.

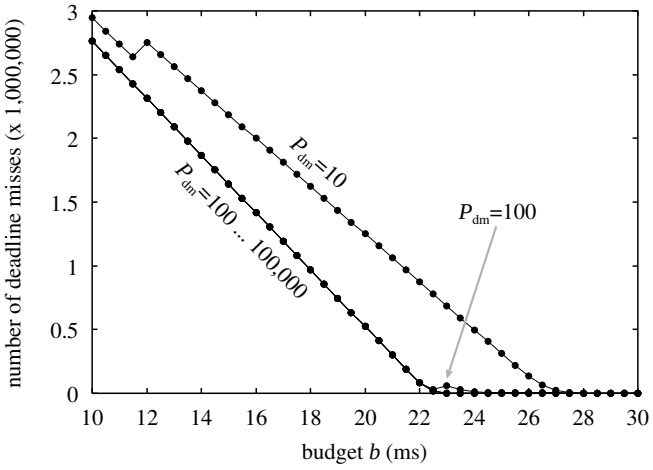


Figure 4.13. The number of deadline misses for different strategies OFF(CONCAT, $P_{\mathrm{dm}}$) applied in simulations on trace ARTIFICIAL, as a function of the budget.

To study the influence of deadline-miss penalty $P_{\text{dm}}$ on the results we obtained for the default parameter settings, we have chosen five different values of $P_{\text{dm}}$: 10, 100, 1000, 10,000, and 100,000. For each value of $P_{\text{dm}}$ we applied strategy OFF(CONCAT, $P_{\text{dm}}$) in 61 simulations on trace ARTIFICIAL, for budgets 10 ms, 10.5 ms, ..., 40 ms. Note that strategy OFF(CONCAT, $P_{\text{dm}} = 10,000$) is the same as strategy OFF(CONCAT). Figures 4.12 and 4.13 show the average revenue per frame and the number of deadline misses that we measured in the simulations for the different strategies, as a function of the budget.

For a given budget, we observe that a higher deadline miss penalty results in a lower average revenue. Up to and including a budget of 22 ms, the four strategies OFF(CONCAT, $P_{\text{dm}}$) for $P_{\text{dm}} = 100$ to $P_{\text{dm}} = 100,000$ only select quality level $q_1$. As a result, their graphs for the number of deadline misses overlap. The four strategies start using higher quality levels at budgets of 22.5 ms, 23.5 ms, 23.5 ms, and 24.0 ms, respectively. The strategy for $P_{\text{dm}} = 100,000$ starts using quality level $q_3$ at $b = 26.5$ ms, which results in a dip in the average revenue at $b = 26$ ms. Other dips in the average revenue can be explained similarly.

Because of its low deadline miss penalty, strategy OFF(CONCAT, $P_{\text{dm}} = 10$) also uses high quality levels at small budgets. Up to and including a budget of 11.5 ms the strategy only selects $q_3$, and from a budget of 12 ms onwards the strategy only selects $q_4$. The switch from $q_3$ to $q_4$ results in a jump in the number of deadline misses. Apparently, the penalty for a deadline miss is outweighted by the reward for a high quality level. If one wants to avoid deadline misses, then the deadline miss penalty should be chosen significantly higher than the rewards for the different quality levels.

### 4.4.6 Processing-time independence revisited

As mentioned earlier, the MDP model implicitly assumes that the processing times of successive frames are independent. So far, we only used trace ARTIFICIAL as simulation trace, a trace that was especially constructed to satisfy this assumption. To evaluate the effect of using a simulation trace that corresponds to real video content, we applied strategy OFF(CONCAT) in 61 simulations on trace CONCAT, for budgets 10 ms, 10.5 ms, ..., 40 ms. We compare the results of these simulations with the results we obtained earlier for strategy OFF(CONCAT) applied in simulations on trace ARTIFICIAL, in Section 4.4.1. Figures 4.14 and 4.15 show the average revenue per frame and the average number of deadline misses per frame, respectively, as a function of the budget, for strategy OFF(CONCAT) applied in simulations on both trace ARTIFICIAL and trace CONCAT. Because the two simulation traces vary in length, we do not consider the actual number of deadline misses, but only an average.

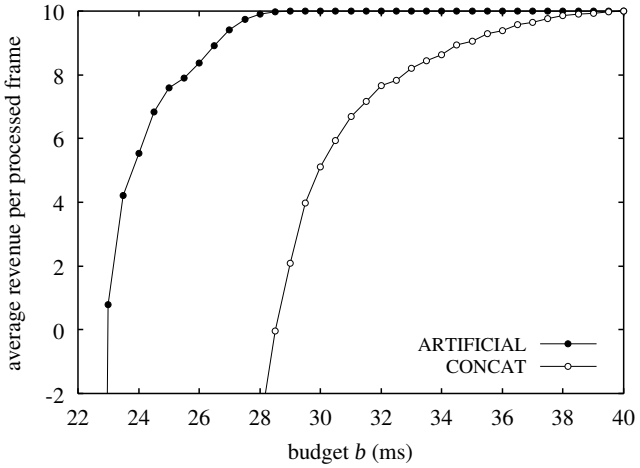


Figure 4.14. The average revenue for strategy OFF(CONCAT) applied in simulations on trace ARTIFICIAL and trace CONCAT, as a function of the budget.

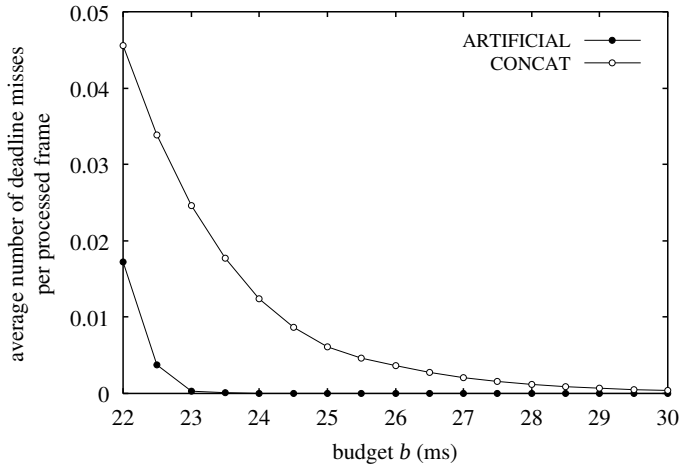


Figure 4.15. The average number of deadline misses for strategy OFF(CONCAT) applied in simulations on trace ARTIFICIAL and trace CONCAT, as a function of the budget.

We observe that the performance of the off-line strategy is disappointing if trace CONCAT is used as simulation trace. For example, using trace ARTIFICIAL as simulation trace, at $b = 24$ ms the average number of deadline misses is almost zero and the average revenue is 5.5. At the same budget, for trace CONCAT the average number of deadlines misses is 0.012 and the average revenue is $-118.7$. This remarkable difference in performance can be explained as follows. The MDP model is memoryless, which means that the processing times of successive frames are implicitly assumed to be independent. For trace ARTIFICIAL this is indeed a valid assumption, because its processing times have been drawn independently from distribution functions. However, the assumption is not valid for trace CONCAT. For example, Figure 4.16 shows the processing times of frames 15,000 – 17,500 of trace ARTIFICIAL and trace CONCAT, for quality level $q_4$. As we can see, trace CONCAT shows dependencies in the processing times of successive frames. These dependencies are due to the related content of successive frames. The off-line strategy is not prepared for such dependencies. In particular, if there is an accumulation of high processing times, then the strategy selects the quality levels for frames too optimistically, because on average it does not expect too many difficult frames in a row. As a result, many deadlines are missed. Similarly, if there is an accumulation of low processing times, then the strategy is too pessimistic, because on average it does not expect so many easy frames in a row.

Because the off-line strategy is not prepared to handle dependencies in the processing times of successive frames, it performs sub-optimal. In the next chapter we enhance the off-line strategy to get over this shortcoming.

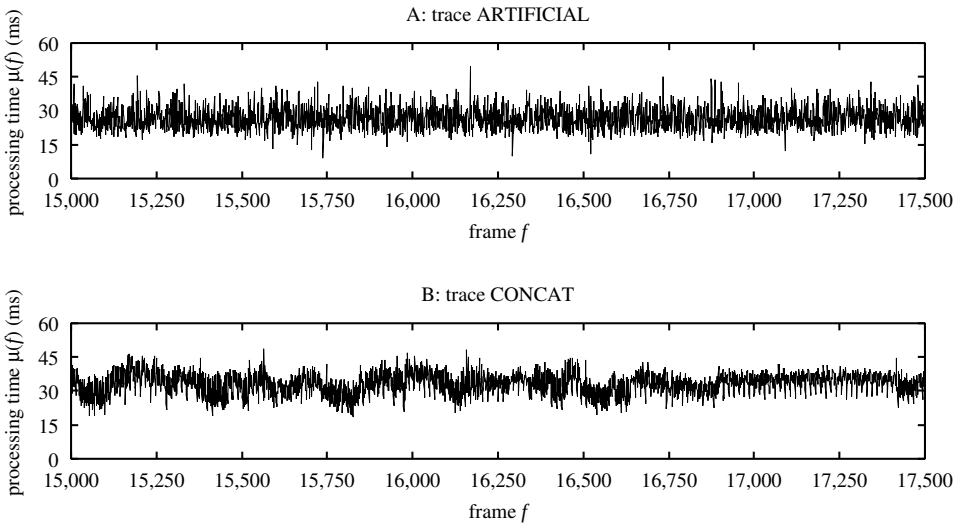


Figure 4.16. The processing times of frames 15,000 – 17,500 of trace ARTIFICIAL (fig. A) and trace CONCAT (fig. B), for quality level $q_4$.

# 5

# Handling load dependencies

At the end of Chapter 4 we observed that the off-line strategy performs well if the processing times of successive frames are independent, but that the strategy falls short when there are load dependencies. In this chapter we present an approach to tackle this shortcoming of the strategy. First, in Section 5.1 we distinguish between short-term and structural load fluctuations, and we define a run-time measure for the structural load. Next, in Section 5.2 we discuss an assumption that is needed to define this run-time measure. Based on the run-time measure for the structural load, in Section 5.3 we enhance the off-line strategy, to take care of dependencies in the processing times of successive frames. Finally, in Section 5.4 we assess the effectiveness of the enhanced strategy by means of simulation experiments.

## 5.1 Load fluctuations

The processing time of a frame may depend on various aspects, as, for example, the quality level at which the frame is processed, the frame type, cache misses, and the overhead imposed by task switching and control. Due to the random or short-term dynamic nature of these aspects, they typically result in short-term fluctuations in the processing times of successive frames. For many video algorithms, the processing time of a frame may also (strongly) depend on the video content. For these algorithms, long-term dependencies in the video content of successive frames may

result in long-term dependencies in the processing times of the frames. In particular, the processing times of frames in the same video shot are usually highly correlated.

To illustrate the influence of the video content on the processing times of frames, in Figure 5.1 we show the processing times of frames 700 – 1700 of trace N, at quality level $q_4$, and we compare these processing times with the corresponding video content. The subsequence consisting of frames 700 – 792 shows a black title screen. Next, subsequence 793 – 1002 is a video shot of a living room with people, and subsequence 1003 – 1176 is a video shot of two similar living rooms. At frame 1177 a concert begins. The concert starts with some dark video shots, and gradually more light and movement appears in the video. The peak in the load at frame 1392 corresponds to a sudden light flash.

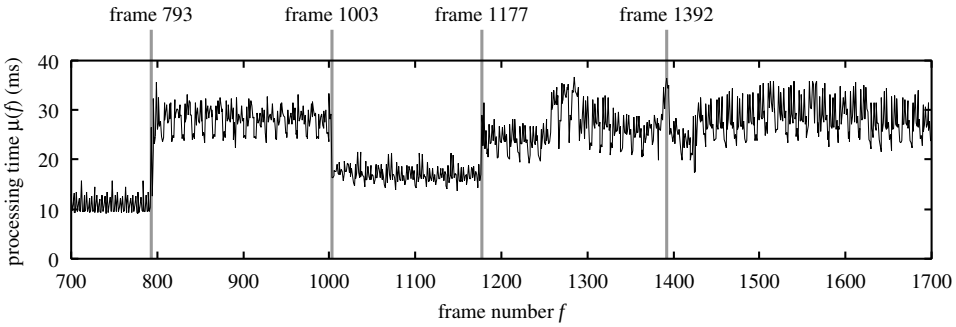


Figure 5.1. The processing times of frames 700 – 1700 of trace N, at quality level $q_4$.

In Figure 5.1 we can distinguish between short-term load fluctuations, as, for example, the load fluctuations within the subsequences 700 – 792 and 793 – 1002, and structural load fluctuations, as, for example, the significant jumps in the load at frames 793, 1003, and 1177. Once the concert begins, at frame 1177, the structural load fluctuations become more smoothly. In the figure, the short-term load fluctuations are mainly due to difference in decoding complexity between MPEG-2 I-, P-, and B-frames, and the structural fluctuations are mainly due to the varying complexity of the video content that is processed.

Given the above, we consider the load of a video processing task to consist of a content-dependent structural load, around which short-term load fluctuations take place. In the off-line strategy, the controller selects the quality level for each frame based on the progress and the previous quality level. To take care of dependencies in the processing times of successive frames, the controller has to select the quality levels for frames based on the structural load as well, but it has to ignore the short-term load fluctuations. Hence, we are interested in a measure for the structural load

that can be computed by the controller at run time.

For an SVA, load fluctuations are partly due to the different quality levels at which successive frames are processed. To remove this influence of the quality level, as a first step towards defining a run-time measure for the structural load we define the complexity factor of a completed frame $f$ by

$$c(f) = \frac{\mu(f)}{\bar{\mu}(q(f))},$$

where $\bar{\mu}(q)$ denotes the expected processing time of a frame processed at quality level $q \in Q$. The $\bar{\mu}(q)$-values for the different quality levels $q$ are determined before run time, for example by computing the average processing time per frame at each quality level, for the processing times in a trace. The used trace should, of course, be representative for the sequence of frames that is processed at run time. For the various traces that we use throughout this thesis, the average processing times are given in Table 4.3.

We assume that the complexity factor of a completed frame is more or less independent of the quality level at which the frame has been processed. We need this assumption, because at run time we can measure the processing time of a frame at only one quality level, the quality level at which the frame is processed, which is not necessarily the quality level at which other frames are processed. We discuss this assumption in more detail in Section 5.2.

The complexity factors of successively completed frames comprise both short-term and structural fluctuations. To remove the short-term fluctuations, we can apply various types of low-pass filters, such as a finite impulse response (FIR) filter or an infinite impulse response (IIR) filter [Oppenheim and Schafer, 1975]. An FIR filter computes a weighted average over a window of past inputs. An IIR filter uses a feedback mechanism, which means that the filter's output is implicitly based on all past inputs. An example of an IIR filter is the exponential recency-weighted average [Sutton and Barto, 1998]. This filter weights recent inputs more heavily than long-past ones. Let $x_i$ and $y_i$ denote the $i$-th input and output of the filter, respectively. Given input $x_i$, output $y_i$ is given by

$$y_i = (1-\theta)y_{i-1} + \theta x_i, \tag{5.1}$$

where $\theta$ is a step-size parameter $(0 < \theta \leq 1)$. Initial value $y_0$ is usually chosen equal to $x_1$. Using this filter, in the computation of $y_i$, to each input $x_j$ $(1 \leq j \leq i)$ a weight of $\theta(1-\theta)^{i-j}$ is assigned, and to $y_0$ a weight of $(1-\theta)^i$ is assigned. The sum of these $i+1$ weights is one, and each weight decreases exponentially as $i$ increases. The closer to zero $\theta$ is chosen, the stronger the filter takes past inputs into account. For $\theta = 1$ each output $y_i$ is given by input $x_i$.

We denote the filtered value of complexity factor $c(f)$ by $c'(f)$. We call $c'(f)$

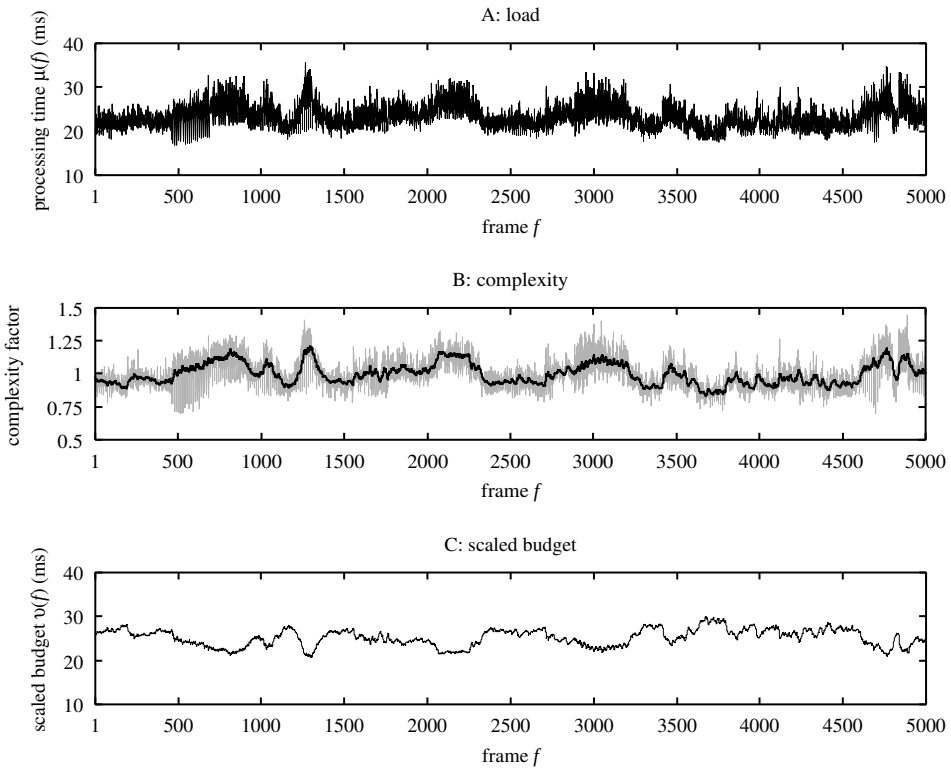


Figure 5.2. The processing times for decoding a sequence of MPEG-2 frames, where each frame is processed randomly at one of four different quality levels (fig. A), the corresponding complexity factors (gray) and running complexity factors (black), computed using filter (5.1) with $\theta = 0.1$ (fig. B), and the corresponding scaled budgets for $b = 25$ ms (fig. C).

the running complexity factor of frame $f$. If we use the running complexity factor without referring to a particular frame, then we implicitly refer to the running complexity factor of the most recently completed frame. For initialization, if there is no completed frame, then we assume a running complexity factor of one.

The running complexity factor of a frame $f$ is a run-time measure for the hardness of the frames in the near vicinity of $f$. At run time, a running complexity factor of, say, 1.2 indicates that the frames that are currently being processed are roughly 1.2 times harder to process than expected. For the SVA, this is like processing frames with a running complexity factor of one, but using a budget that is 1.2 times smaller than the actual budget $b$. In other words, a budget of 30 ms appears to be a budget of only $30\,\text{ms}/1.2 = 25\,\text{ms}$. Similarly, for a running complexity factor

of 0.8 that same budget appears to be 37.5 ms. We call these instinctive budgets of 25 ms and 37.5 ms scaled budgets. More precisely, we define the scaled budget of a completed frame $f$ by

$$\upsilon(f) = \frac{b}{c'(f)}.$$

Again, if we use the scaled budget without referring to a particular frame, then we implicitly refer to the scaled budget of the most recently completed frame. For initialization, if there is no completed frame, then we assume a scaled budget of $b$.

To illustrate the complexity factor, running complexity factor, and scaled budget, we created a small trace consisting of the processing times for the first 5000 frames of trace E. Figure 5.2A shows the processing times of the 5000 frames, where for each frame we have selected the applied quality level at random from set $Q$. If we compute the complexity factor of each frame, using $\bar{\mu}(q)$-values that are given by the different average processing times of the 5000 frames, then we obtain the gray graph of complexity factors that is shown in Figure 5.2B. The black line in this figure shows the running complexity factors, computed using filter (5.1) with a step-size parameter $\theta = 0.1$. Next, Figure 5.2C shows the corresponding graph of the scaled budget, for $b = 25$ ms.

## 5.2 Complexity-factor assumption

In this section we discuss the assumption that the complexity factor of a completed frame is more or less independent of the quality level at which the frame has been processed. Because we use the complexity factor of a frame only to compute the running complexity factor of the frame, we evaluate the assumption from the perspective of the running complexity factor.

For each quality level $q_1$ to $q_4$ we have computed complexity factor $c(f)$ and running complexity factor $c'(f)$ for each frame $f$ of trace CONCAT. To compute the complexity factors, we used the average processing times of trace CONCAT as given in Table 4.3 for the different $\bar{\mu}(q)$-values. To compute the running complexity factors, for each quality level we applied a copy of filter (5.1) with a step-size parameter $\theta = 0.1$. We denote the complexity factor and running complexity factor of a frame $f$ that has been processed at quality level $q$ by $c(f, q)$ and $c'(f, q)$, respectively. Using this notation, we define the maximum relative difference in running complexity factor of a frame $f$ by $\min_{q \in Q} c'(f, q) / \max_{q \in Q} c'(f, q)$. Figure 5.3 shows, for each quality level $q$, a cumulative distribution function of running complexity factor $c'(f, q)$. Next, Figure 5.4 shows a cumulative distribution function of the maximum relative difference in running complexity factor.

In Figure 5.3 we observe that the running complexity factors of frames vary roughly between 0.6 and 1.4, for each of the four quality levels. We observe that

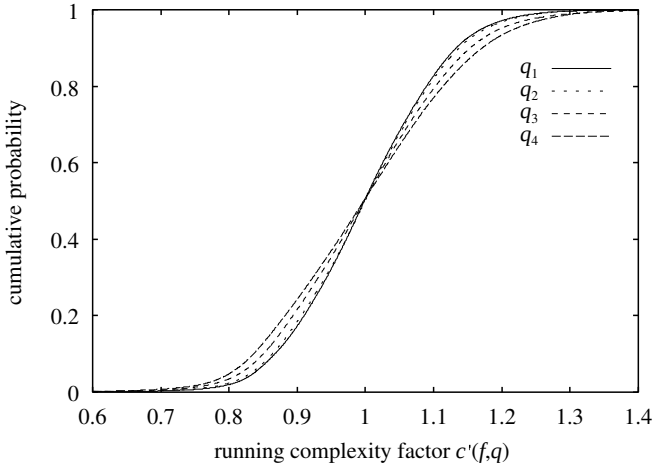


Figure 5.3. A cumulative distribution function of running complexity factor $c'(f, q)$ for each quality level $q$, for the frames $f$ of trace CONCAT.

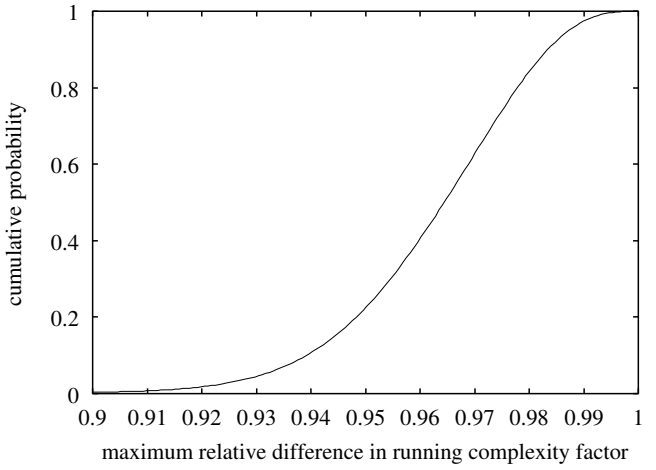


Figure 5.4. A cumulative distribution function of the maximum relative difference in running complexity factor, for the frames of trace CONCAT.

the four distribution functions are relatively close to each other. Next, in Figure 5.4 we observe that the maximum relative difference in running complexity factor varies roughly between 0.9 and 1. Based on this result, we consider the running complexity factor for a frame to be roughly independent of the applied quality level.

## 5.3 Enhanced off-line strategy

In this section we discuss our approach to enhance the off-line strategy, to take care of dependencies in the processing times of successive frames. First, in Section 5.3.1 we consider a straightforward approach to enhance the off-line strategy. Because this approach has severe disadvantages, in Section 5.3.2 we present an alternative approach to enhance the off-line strategy, the so-called budget scaling approach. Additional steps needed for the latter approach are discussed in Sections 5.3.3 and 5.3.4. In Section 5.3.5 we discuss the run-time overhead of the enhanced strategy. Finally, in Section 5.3.6 we compare the budget scaling approach with the straightforward approach.

### 5.3.1 A straightforward approach

To take care of dependencies in the processing times of successive frames in the off-line strategy, a straightforward approach is to extend the definition of a state in the MDP model with an extra component, viz. the running complexity factor. Because the running complexity factor is a continuous measure, it has to be discretized in the same way as we discretized the progress, i.e., using intervals. Using the extra state component, the controller can select the quality levels for frames based on the structural load as well, without having the interference of short-term load fluctuations. This approach requires that the controller keeps track of the running complexity factors of frames at run time.

The main disadvantage of this approach is that it suffers from from the so-called curse of dimensionality [Sutton and Barto, 1998]. This means that the number of states grows exponentially with the number of state components. Due to the larger set of states, a larger set of processing-time statistics is needed to obtain the same accuracy of the distribution functions $F_{\phi q}$ of $F_q$ in the MDP model. Moreover, the set of states can easily grow too large to solve the MDP model in reasonable time. For example, in Section 4.4.4 we measured a computation time of 3.0 s to solve the MDP model for $b = 27$ ms, $n_\Lambda = 300$, and $n_\Phi = 1$. If we assume a quadratic dependence between the number of states and the time required to compute a policy, and if we also use a granularity of 300 intervals to discretize the running complexity factor, then the computation time will grow to over three days. Moreover, if we also distinguish between the different MPEG-2 frame types in the MDP model,

then the computation time will grow to over 28 days. Even worse, we may have to compute a policy for many different values of the budget, as the run-time budget of the SVA is usually not known in advance. Hence, the straightforward approach to enhance the off-line strategy is practically infeasible.

### 5.3.2 Budget scaling approach

To avoid the disadvantages of the straightforward approach, we use an alternative approach to enhance the off-line strategy, the so-called budget scaling approach. The idea of the approach is the following. At run time we let the controller compute the running complexity factors of the successively completed frames. A running complexity factor of $c'$ at the start point of a frame $f$ indicates that the frames in the near vicinity of $f$ are roughly $c'$ times harder than expected (ignoring short-term fluctuations in the complexity factors of the frames). For the controller this feels like the frames are exactly as hard as expected, but using a budget that is $c'$ times smaller than the actual budget $b$, i.e., a scaled budget of $b/c'$. To select quality level $q(f)$, we hence let the controller apply a policy that has been computed for exactly this situation, i.e., a policy computed for scaled budget $\upsilon = b/c'$.

To compute a policy for a scaled budget $\upsilon$, in the MDP model we fill in scaled budget $\upsilon$ for budget $b$. Furthermore, in the MDP model we use processing-time distribution functions $F_{\phi q}$ or $F_q$ which, for each quality level $q$, only comprise short-term load fluctuations around the structural load. In Section 5.3.3 we discuss how these distribution functions can be derived.

Before run time, we compute a policy – in principle – for each possible value of the scaled budget. Next, at run time, to select the quality level for a frame, we let the controller apply a policy that was computed for the actual value of the scaled budget. Hence, the controller always applies a policy that is optimized for handling short-term load fluctuations around the actual structural load. We call the off-line strategy that is enhanced with this budget scaling approach the enhanced off-line strategy, or, in short, the *enhanced strategy*. Note that in the enhanced strategy we do not change the SVA's budget for processing, $b$, but only the budget for which the controller looks up the policy.

Clearly, because the scaled budget is a continuous measure, it is not feasible to solve the MDP model for each possible value of the scaled budget. Therefore, in practice we solve the MDP model only for a small well-chosen set of scaled budgets. To approximate the policy for a particular value of the scaled budget, we apply linear interpolation between the computed policies. We discuss this process in more detail in Section 5.3.4.

### 5.3.3 Trace normalization

As mentioned, in the MDP model we should use processing-time distribution functions $F_{\phi q}$ or $F_q$ that for each quality level $q$ only comprise short-term load fluctuations around the structural load. We derive these distribution functions from the processing times of a so-called normalized trace. In this section we discuss how we can generate a normalized trace from a given source trace.

To generate a normalized trace from a given source trace, we mimic the run-time effect of scaling, and we compensate for that in the processing times of the source trace. Given the processing times in the source trace, for each quality level $q$ we first compute the average processing time per frame. We use these average processing times for the different $\bar{\mu}(q)$-values. Next, we again consider the processing times of the successive frames in the source trace. For each frame $f$, and for each quality level $q$, we compute complexity factor $c(f,q)$ using expected processing time $\bar{\mu}(q)$, and we compute running complexity factor $c'(f,q)$ using the same low-pass filter as will be applied at run time. To compute the running complexity factors $c'(f,q)$ for a frame $f$ at the different quality levels $q$, we use $n_Q$ copies of the filter, one copy for each quality level. For the normalized trace, we next define the processing time of frame $f$ at quality level $q$ by

$$\mu^{\mathrm{n}}(f,q) = \begin{cases} \mu(f,q) & \text{for } f = 1 \\ \\ \mu(f,q)/c'(f-1,q) & \text{for } f > 1, \end{cases} \tag{5.2}$$

where $\mu(f,q)$ denotes the processing time of frame $f$ at quality level $q$, as given in the source trace. Note that in (5.2) we use denominator $c'(f-1,q)$ instead of $c'(f,q)$, to match the run-time situation where the controller at the start point of a frame $f$ only knows the running complexity factor of the just-completed frame $f-1$, and not of the frame $f$ to be processed. If the applied low-pass filter is well-chosen, then the processing times in the normalized trace will for each quality level $q$ only comprise short-term load fluctuations around the structural load.

To synchronize with the budget scaling process at run time, we let the controller apply the $\bar{\mu}(q)$-values and the filter that were also applied to create the normalized trace that was used to compute the policies. Note that the running complexity factors that are computed by the controller at run time are based on the varying quality levels at which frames are processed. As mentioned, this is unavoidable, because at run time we can measure the processing time of a frame for only one quality level, the quality level at which the frame is processed, which is not necessarily the quality level at which other frames are processed. Clearly, in the process of generating a normalized trace we do not have this restriction, because the source trace contains the processing times of each frame at all quality levels.

### 5.3.4 Linear interpolation of policies

We now discuss the interpolation step. As mentioned, we solve the MDP model for a finite set of scaled budgets, and we let the controller interpolate linearly between the computed policies to select the quality level for a frame. We explain this process using Figures 5.5 and 5.6. To derive these figures, we used an MDP model with four quality levels, $q_1$ to $q_4$, and a periodic latency $\delta = 3$. For simplicity, in the MDP model we do not distinguish between different frame types. We solved the MDP model for scaled budgets 15 ms, 16 ms, ..., 40 ms. Figure 5.5 shows the policy that was computed for a scaled budget of 25 ms. Next, Figure 5.6 shows the space of all computed policies, as a function of the scaled budget, but the view is restricted to previous quality level $q_3$ only. We can make such a figure for each previous quality level. The dashed column for previous quality level $q_3$ in Figure 5.5 corresponds to the dashed column for a scaled budget of 25 ms in Figure 5.6. In Figure 5.6, we have connected the boundaries of the successive (sub)policies by lines.

To select the quality level for a frame, the controller interpolates linearly between the computed policies. For example, at the start point of a frame $f$, if the scaled budget is 28.5 ms, then the controller interpolates linearly between the policies for the scaled budgets of 28 ms and 29 ms to select quality level $q(f)$. If the progress of frame $f$ at the start point is given by $\lambda_\alpha(f) = 2$, and if the previous quality level is given by $q_3$, then the controller selects quality level $q_4$. This is indicated by the '+' in the figure. We do not extrapolate policies, i.e., for scaled budgets smaller than 15 ms or larger than 40 ms the controller uses the policies for scaled budgets of 15 ms and 40 ms, respectively.

### 5.3.5 Run-time overhead

Due to the budget scaling approach, the enhanced strategy results in a higher run-time overhead than the off-line strategy. At each start point of a frame, the enhanced strategy first has to compute a complexity factor, a running complexity factor, and a scaled budget. Next, the strategy has to interpolate between the computed policies to select the quality level for the frame to be processed. Because these computations can all be done in a small number of steps, we consider the run-time overhead of the enhanced strategy to be comparable to the run-time overhead of the off-line strategy.

### 5.3.6 Budget scaling approach vs. straightforward approach

In the straightforward approach of Section 5.3.1, the running complexity factor has to be discretized to obtain a finite set of states for the MDP model. This discretization has to be done using intervals, to be able to compute state-transition probabilities and expected revenues for the MDP model. In the budget scaling approach,

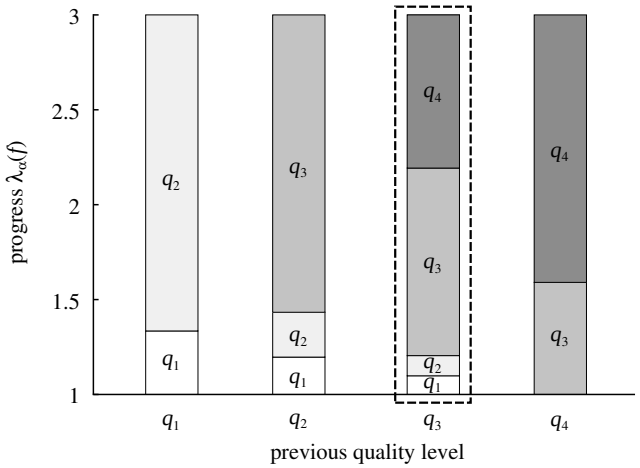


Figure 5.5. A policy computed for a scaled budget of 25 ms.

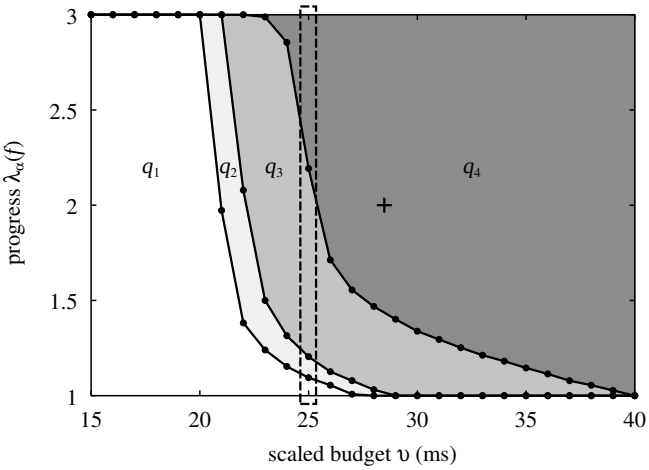


Figure 5.6. A space of policies computed for scaled budgets 15 ms, 16 ms, ..., 40 ms. The view is restricted to previous quality level $q_3$ only.

the scaled budget is also discretized. However, this is not done using intervals, but by defining a finite set of scaled budget values for which a policy is computed, and by applying linear interpolation between the computed policies at run time. The latter discretization method is possible because we use the scaled budget indirectly as a state component. Because a discretization step using intervals is less accurate than a discretization step using linear interpolation, in the budget scaling approach we can discretize the scaled budget at a much lower granularity than the running complexity factor in the straightforward approach, to obtain a solution of the same accuracy. Hence, significantly less computation time is needed to derive the set of policies for the budget scaling approach than is needed to derive the single policy for the straightforward approach. Moreover, if the budget of the SVA is not known before run time, then using the straightforward approach we have to compute a policy for different possible values of the budget.

## 5.4 Simulation experiments

To evaluate the enhanced strategy, we have run simulation experiments. These experiments are a continuation of the experiments described in Section 4.4.

### Settings

Given the default parameter settings defined in Section 4.4, for the enhanced strategy we add one default parameter setting, namely an IIR filter with a step-size parameter $\theta = 0.1$. We evaluate other values of $\theta$ in Section 5.4.2.

To implement the enhanced strategy, we always solve the MDP model 61 times, for scaled budgets 10 ms, 10.5 ms, ..., 40 ms. Hence, the enhanced strategy has to interpolate between 61 policies. We use ENH($\tau$) as a shorthand notation for the enhanced strategy, using a set of policies computed for the default settings and a statistics trace $\tau$. Statistics trace $\tau$ is used as source trace to generate a normalized trace. The normalized trace is used as input for the MDP model, to derive policies for the strategy. The $\bar{\mu}(q)$-values and the low-pass filter that are used in the trace normalization process are also used at run time by the strategy, to compute the complexity factors and running complexity factors of completed frames. For any change in the parameter settings with respect to the default settings, we extend the shorthand notation. For example, if we use an IIR filter with a step-size parameter of 0.2 instead of 0.1, then we denote the strategy by ENH($\tau, \theta = 0.2$).

### Benchmark strategies

To assess the performance of control strategies, we introduce two benchmark strategies. As a simple reference strategy we introduce Q4, which selects quality level $q_4$ for each frame to be processed. Hence, this strategy corresponds to no (intelligent) control at all. As a best-case strategy we introduce CLV, a clairvoyant

strategy which optimally selects the quality level for each frame based on complete knowledge of the processing times of future frames. Strategy CLV is virtual, in the sense that it cannot be used in a simulation, but it only provides an upper bound on the average revenue that can be attained in a simulation. This upper bound is computed off line, based on the processing times given in a simulation trace. At the start point of a frame, a run-time strategy can only guess what the processing time of the frame at a particular quality level will be. The clairvoyant strategy is not a run time strategy, because it knows this processing time in advance, as well as the processing times of all the frames that follow. The closer a run-time strategy performs to the clairvoyant strategy, the better it is.

To compute an upper bound on the average revenue that can be obtained in a simulation on a trace $\tau$ for a budget $b$, as provided by the clairvoyant strategy, we apply dynamic programming [Bellman, 1957]. We do this as follows. Let $n$ denote the number of frames of trace $\tau$. We consider successively the frames $n, n-1, \ldots, 1$. For each frame $f$ that we consider, for each state $s \in S$, and for each quality level $q \in Q$, we compute the effect of processing frame $f$ at quality level $q$, starting from state $s$ at the start point of frame $f$. This computation is based on (2.1) – (2.5), in which the processing time of frame $f$ at quality level $q$ is taken from trace $\tau$. In particular, we compute the frame that has to be processed by the SVA after processing frame $f$, denoted by $g(f, s, q)$, the resulting state of the SVA at the start point of frame $g(f, s, q)$, denoted by $s'(f, s, q)$, and the revenue corresponding to the state transition, denoted by $r'(f, s, q)$. Usually, frame $g(f, s, q)$ corresponds to frame $f+1$, but if the skipping approach is applied the frame may also corresponds to a later frame. Note that frame $g(f, s, q)$ and state $s'(f, s, q)$ are undefined if $f$ is the last frame to be processed, for example when $f = n$, or when $f = n-1$ and frame $n$ is skipped. Because the number $n$ of frames is given, we have a finite time horizon. Hence, we can maximize the sum of the revenues over all processed frames. To compute the maximum average revenue, we also have to know the number of processed frames. We next give the required recursive equations for dynamic programming.

For each frame $f$ and for each state $s$, we compute an accumulated revenue $r^{\text{acc}}(f, s)$ and a number $i(f, s)$ of processed frames. If $f$ or $s$ is undefined, then we define $r^{\text{acc}}(f, s) = 0$ and $i(f, s) = 0$. Otherwise, to compute $r^{\text{acc}}(f, s)$ and $i(f, s)$, we first compute quality level $q(f, s)$ by[1]

$$q(f, s) = \arg\max_{q \in Q} (r'(f, s, q) + r^{\text{acc}}(g(f, s, q), s'(f, s, q))).$$

[1] Note that, in theory, another quality level $q(f, s)$ might result in a smaller number $i(f, s)$, which in turn might result in a larger fraction $r^{\text{acc}}(f, s)/i(f, s)$. However, this is very unlikely, because a smaller number $i(f, s)$ means that more deadlines have been missed, and deadline misses are heavily penalized.

Next, we compute the accumulated revenue $r^{\mathrm{acc}}(f,s)$ by

$$r^{\mathrm{acc}}(f,s) = r'(f,s,q(f,s)) + r^{\mathrm{acc}}(g(f,s,q(f,s)), s'(f,s,q(f,s))),$$

and we compute the number $i(f,s)$ by

$$i(f,s) = 1 + i(g(f,s,q(f,s)), s'(f,s,q(f,s))).$$

Given the state $s \in S$ of the SVA at the start point of frame 1, an upper bound on the average revenue that can be attained in the simulation is given by $r^{\mathrm{acc}}(1,s)/i(1,s)$. From the dynamic programming computation we can also derive other performance aspects, such as the optimal quality level to be chosen for each frame, and the number of deadline misses.

**Experiment overview**

First, in Section 5.4.1 we study the behavior of the enhanced strategy for the default settings, and we benchmark the strategy against the off-line strategy, strategy Q4, and the clairvoyant strategy. Next, in Section 5.4.2 we vary step-size parameter $\theta$. Finally, in Section 5.4.3 we give an answer to the still pending question whether it is useful to distinguish between the different MPEG-2 frame types in the MDP model.

### 5.4.1 Results for the default settings

To study the behavior of the enhanced strategy for the default settings, and to benchmark the strategy against other strategies, we applied the strategies Q4, OFF(CONCAT), ENH(CONCAT), and CLV each in 61 simulations on trace CONCAT, for budgets 10 ms, 10.5 ms, ..., 40 ms. Figures 5.7 and 5.8 show the average revenue per frame and the number of deadline misses, respectively, that we measured in the different simulations, as a function of the budget. Next, Figures 5.9 and 5.10 show the number of frames that were processed at each quality level, for strategies OFF(CONCAT) and ENH(CONCAT), respectively, as a function of the budget.

For all strategies we observe that the average revenue is strongly negative for small budgets, and that the average revenue converges to the maximum value of 10 as the budget increases. An average revenue of nearly 10 implies that almost no deadlines are missed and that almost all frames are processed at quality level $q_4$. The enhanced strategy performs closest to optimum, and clearly outperforms strategies Q4 and OFF(CONCAT). For example, at a budget of 27 ms, strategies Q4, OFF(CONCAT), ENH(CONCAT), and CLV attain average revenues of $-402.6$, $-12.0$, 0.2, and 5.0, respectively. Due to convergence, the difference in average revenue between strategies ENH(CONCAT) and CLV becomes smaller as the budget increases. Although this difference is still significant at a budget of 27 ms, we can also look at the results from a different perspective. Table 5.1

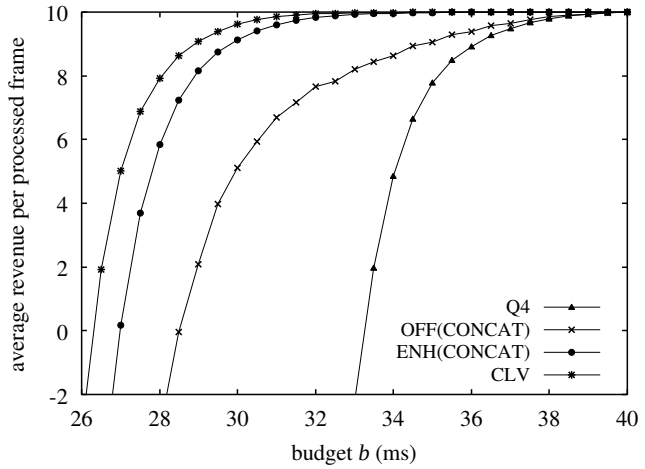


Figure 5.7. The average revenue for strategies Q4, OFF(CONCAT), ENH(CONCAT), and CLV applied in simulations on trace CONCAT, as a function of the budget.

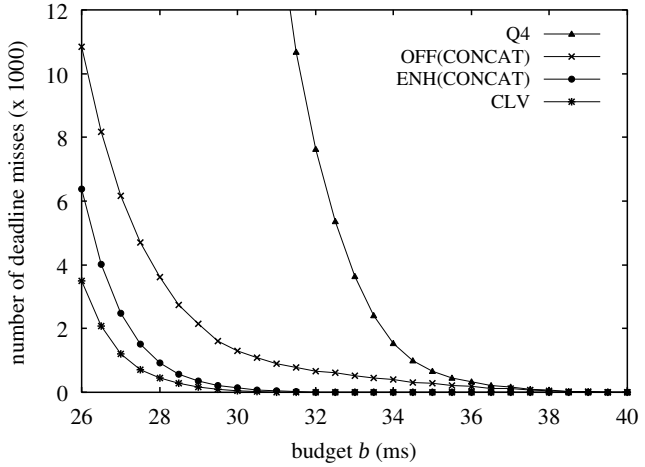


Figure 5.8. The number of deadline misses for strategies Q4, OFF(CONCAT), ENH(CONCAT), and CLV applied in simulations on trace CONCAT, as a function of the budget.

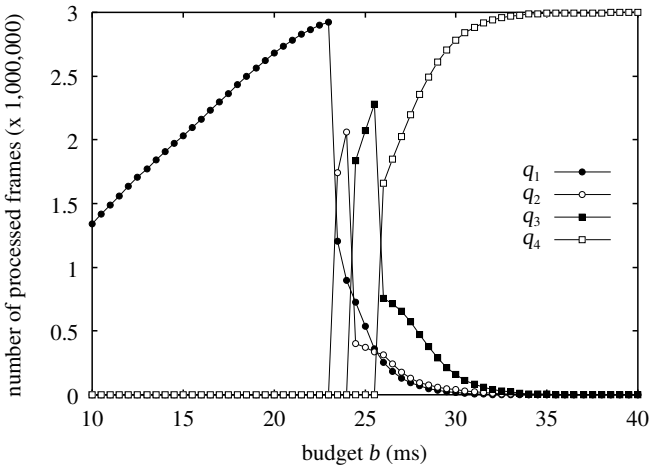


Figure 5.9. The number of frames processed at each quality level, as a function of the budget, for strategy OFF(CONCAT) applied in simulations on trace CONCAT.

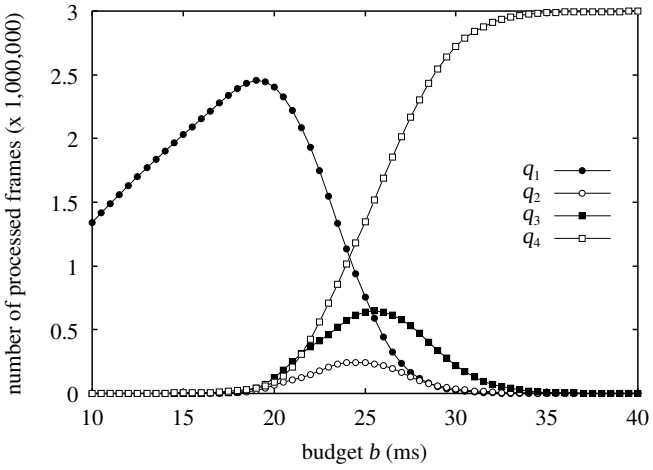


Figure 5.10. The number of frames processed at each quality level, as a function of the budget, for strategy ENH(CONCAT) applied in simulations on trace CONCAT.

shows for each strategy the various budgets required to attain different values of the average revenue, computed by using linear interpolation between the simulation results. To attain an average revenue of 5, CLV requires a budget of 27.0 ms, ENH(CONCAT) requires a budget of 27.8 ms (+0.8 ms), OFF(CONCAT) requires a budget of 30.0 ms (+3.0 ms), and Q4 requires a budget of 34.0 ms (+7.0 ms). Hence, at the cost of a slightly larger budget, the enhanced strategy performs equally well as the clairvoyant strategy. The main difference between the enhanced strategy and the clairvoyant strategy is that the clairvoyant strategy can prepare for structural load increases, whereas the enhanced strategy can only react to them.

Table 5.1. The budgets required to attain different values of the average revenue, for strategies CLV, Q4, OFF(CONCAT), and ENH(CONCAT) applied in simulations on trace CONCAT.

| average revenue | CLV | Q4 | OFF(CONCAT) | ENH(CONCAT) |
|---|---|---|---|---|
| 0.0 | 26.3 ms | 33.3 ms (+7.0 ms) | 28.5 ms (+2.2 ms) | 27.0 ms (+0.7 ms) |
| 1.0 | 26.4 ms | 33.4 ms (+7.0 ms) | 28.7 ms (+2.3 ms) | 27.1 ms (+0.7 ms) |
| 2.0 | 26.5 ms | 33.5 ms (+7.0 ms) | 29.0 ms (+2.5 ms) | 27.3 ms (+0.8 ms) |
| 3.0 | 26.7 ms | 33.7 ms (+7.0 ms) | 29.2 ms (+2.5 ms) | 27.4 ms (+0.7 ms) |
| 4.0 | 26.8 ms | 33.9 ms (+7.1 ms) | 29.5 ms (+2.7 ms) | 27.6 ms (+0.8 ms) |
| 5.0 | 27.0 ms | 34.0 ms (+7.0 ms) | 30.0 ms (+3.0 ms) | 27.8 ms (+0.8 ms) |
| 6.0 | 27.3 ms | 34.3 ms (+7.0 ms) | 30.5 ms (+3.2 ms) | 28.1 ms (+0.8 ms) |
| 7.0 | 27.6 ms | 34.7 ms (+7.1 ms) | 31.3 ms (+3.7 ms) | 28.4 ms (+0.8 ms) |
| 8.0 | 28.0 ms | 35.2 ms (+7.2 ms) | 32.7 ms (+4.7 ms) | 28.9 ms (+0.9 ms) |
| 9.0 | 28.9 ms | 36.1 ms (+7.2 ms) | 34.8 ms (+5.9 ms) | 29.8 ms (+0.9 ms) |
| 9.9 | 31.4 ms | 38.7 ms (+7.3 ms) | 38.4 ms (+7.0 ms) | 32.6 ms (+1.2 ms) |

If we look at the number of deadline misses, then we observe that the enhanced strategy outperforms strategies Q4 and OFF(CONCAT). At the cost of a slightly larger budget, the number of deadline misses for the enhanced strategy is at the same level as for the clairvoyant strategy. Because deadline misses are heavily penalized, a small increase in the number of deadline misses may result in a large decrease in the average revenue. For this reason, the difference in the number of deadline misses between the off-line strategy and the enhanced strategy looks relatively small as compared to the difference in average revenue. For strategies ENH(CONCAT) and CLV, the number of deadline misses becomes zero at budgets of 35 ms and 32 ms, respectively. Apparently, for the enhanced strategy the work-ahead of two periods suffices to absorb load fluctuations from a budget of 35 ms onwards. At a budget of 35 ms, strategies Q4 and OFF(CONCAT) still result in 664 and 284 deadline misses, respectively, and at a budget of 40 ms they each result in two deadline misses.

It is also interesting to look at the quality levels that are chosen by the different strategies. For small budgets the off-line strategy only selects quality level $q_1$, and many deadlines are missed. For each missed deadline one frame is skipped. The more budget is given to the SVA, the better it is capable of meeting deadlines. Hence, as the budget increases, the number of frames that are processed at $q_1$ first grows towards the number of frames of trace CONCAT, which is 2,997,326. Right before this level is reached, the strategy starts using $q_2$, at a budget of 23.5 ms. Next, the strategy starts using $q_3$ and $q_4$ at budgets of 24.5 ms and 26 ms, respectively. When the strategy starts using a higher quality level, it switches radically to the new quality level, which results in the steep edges in Figure 5.9. At a budget of 28.5 ms, the first budget at which the off-line strategy attains a positive average revenue, 1.7%, 2.6%, 12.5%, and 83.2% of the frames are processed at quality levels $q_1$, $q_2$, $q_3$, and $q_4$, respectively.

For small budgets the enhanced strategy also mainly uses quality level $q_1$, and the number of frames that are processed at $q_1$ grows towards the number of frames of trace CONCAT as the budget increases. However, in contrast to the off-line strategy, the enhanced strategy also uses higher quality levels at small budgets. For example, at a budget of 10 ms, 32 frames are processed at $q_2$, 59 frames are processed at $q_3$, and 272 frames are processed at $q_4$. The usage of high quality levels at small budgets is due to subsequences that turn out to be easier than expected, for which the strategy applies a policy corresponding to a scaled budget that is larger than budget $b$. Due to the budget scaling approach, the usage of the different quality levels is more smoothly balanced over the budget spectrum. At a budget of 28.5 ms, 2.7%, 2.6%, 13.5%, and 81.2% of the frames are processed at quality levels $q_1$, $q_2$, $q_3$, and $q_4$, respectively. Although these fractions are roughly the same for the off-line strategy, the enhanced strategy delivers a much better job in preventing deadline misses.

### 5.4.2 Varying step-size parameter $\theta$

To study the influence of step-size parameter $\theta$ on the results we obtained for the default settings, we defined a set $\Theta$ with twelve candidate step-size parameters: $0.001, 0.01, 0.05, 0.1, 0.15, \ldots, 0.5$. For each $\theta \in \Theta$, we applied strategy ENH(CONCAT, $\theta$) in 61 simulations on trace CONCAT, for budgets 10 ms, 10.5 ms, ..., 40 ms. Note that strategy ENH(CONCAT, $\theta = 0.1$) is the same as strategy ENH(CONCAT). Figure 5.11 shows the average revenue that we measured in the simulations for the different strategies, as a function of $\theta$, for budgets between 27 ms and 31 ms.

For small budgets the average revenue is strongly negative for every $\theta \in \Theta$. As the budget increases, the average revenue turns positive at $b = 27$ ms, for $\theta = 0.1$, $\theta = 0.15$, $\theta = 0.2$, and $\theta = 0.25$. In Figure 5.11, the distance between the

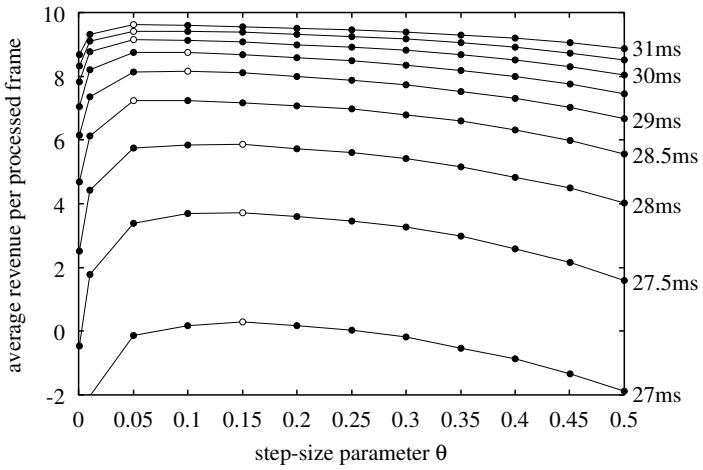


Figure 5.11. The average revenue for different strategies ENH(CONCAT, θ) applied in simulations on trace CONCAT, as a function of step-size parameter θ, for budgets between 27 ms and 31 ms. For each budget, we indicate the highest average revenue over all twelve values of θ by means of a white dot.

Table 5.2. Error measure $e(B, \theta)$ for different budget sets $B$ and for each $\theta \in \Theta$.

| θ | $e(B(10, 40), \theta)$ | $e(B(27, 40), \theta)$ | $e(B(27, 31), \theta)$ |
|---|---|---|---|
| 0.001 | 70.23 | 25.45 | 21.97 |
| 0.01 | 50.19 | 10.10 | 9.19 |
| 0.05 | 44.71 | 0.93 | 0.91 |
| 0.1 | 41.23 | 0.30 | 0.22 |
| 0.15 | 33.86 | 0.57 | 0.37 |
| 0.2 | 32.02 | 1.63 | 1.26 |
| 0.25 | 30.51 | 2.76 | 2.19 |
| 0.3 | 31.26 | 4.29 | 3.47 |
| 0.35 | 35.10 | 6.42 | 5.28 |
| 0.4 | 41.08 | 8.99 | 7.43 |
| 0.45 | 52.25 | 12.18 | 10.09 |
| 0.5 | 64.25 | 16.09 | 13.36 |

successive budget lines becomes smaller, which is due to the convergence of the average revenue to 10 as the budget increases. The budget lines show maximum average revenues for values of $\theta$ in the range $[0.05, 0.15]$.

We came to the default setting $\theta = 0.1$ as follows. Let $x(b, \theta)$ denote the average revenue that we measured in the simulation for budget $b$ and step-size parameter $\theta$. Furthermore, let $B(i, j)$ denote the subset of budgets from set $\{10\,\text{ms}, 10.5\,\text{ms}, \ldots, 40\,\text{ms}\}$ in the range $[i\,\text{ms}, j\,\text{ms}]$. For a set of budgets $B$ we define error measure $e(B, \theta)$ by

$$e(B, \theta) = \sum_{b \in B} (\max_{\theta' \in \Theta} x(b, \theta') - x(b, \theta)).$$

Table 5.2 shows $e(B, \theta)$ for the different values $\theta \in \Theta$, and for the budget sets $B(10, 40)$, $B(27, 40)$, and $B(27, 31)$. The smaller the error measure for a given set of budgets $B$, the better. Because we are interested in a positive average revenue, we focus on budgets of 27 ms and larger. Hence, based on the results in Table 5.2, we have selected $\theta = 0.1$ as default step-size parameter.

### 5.4.3 Distinguishing between MPEG-2 frame types

In this section we give an answer to the pending question whether it is useful to distinguish between the different MPEG-2 frame types I, P, and B in the MDP model. To indicate that a strategy was derived taking MPEG-2 frame types into account in the MDP model, we extend the name of the strategy with 'IPB'.

There are a number of disadvantages to distinguishing between different frame types in the MDP model. A first and main disadvantage is that the processing-time distribution functions in the MDP model, which are derived from a statistics trace, have a lower accuracy than normal, and that the functions can vary in accuracy. For example, trace CONCAT contains the processing times of 235,632 I-frames, 782,460 P-frames, and 1,979,234 B-frames. Hence, a distribution function $F_{\phi q}$ is less accurate for an I-frame than for a P-frame, and for a P-frame the distribution function is less accurate than for a B-frame. If we do not distinguish between I-, P-, and B-frames, then the distribution functions in the MDP model are all based on the processing times of 2,997,326 frames. Although a distribution function based on the processing times of 235,632 frames is probably accurate enough, the accuracy could be insufficient if a smaller statistics trace is used.

A second disadvantage is that the time needed to solve the MDP model increases significantly. For example, Figure 5.12 shows the time needed by successive approximation to compute policies for strategies OFF(CONCAT) and OFF(CONCAT, IPB), for budgets $10\,\text{ms}, 10.5\,\text{ms}, \ldots, 40\,\text{ms}$. We observe that the computation time varies as a function of the budget. If we distinguish between I-, P-, and B-frames in the MDP model, then the number of states in the model grows

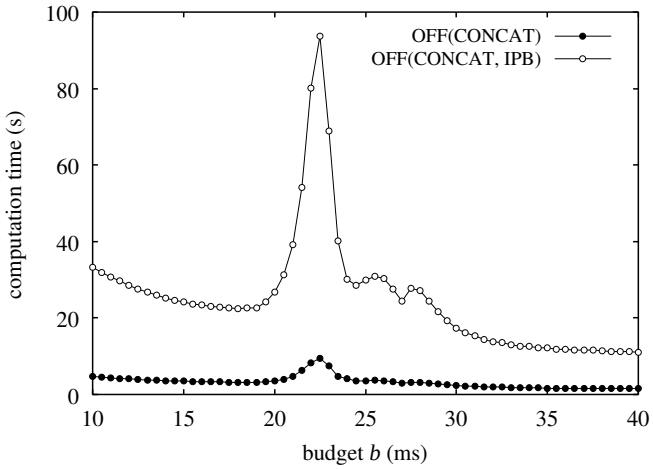


Figure 5.12. The time needed by successive approximation to compute policies for strategies OFF(CONCAT) and OFF(CONCAT, IPB), as a function of the budget.

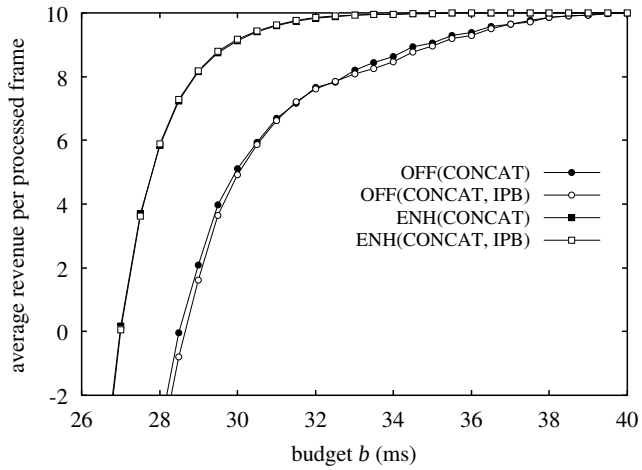


Figure 5.13. The average revenue for strategies OFF(CONCAT), OFF(CONCAT, IPB), ENH(CONCAT), and ENH(CONCAT, IPB) applied in simulations on trace CONCAT, as a function of the budget.

by a factor of three, but over all 61 budgets we experience an average increase in computation time by a factor of 7.7.

The only reason to distinguish between MPEG-2 frame types in the MDP model is to obtain a higher average revenue at run time, by taking advantage of the specific processing-time statistics of I-, P-, and B-frames. To study if it is useful to distinguish between MPEG-2 frame types in the MDP model, we applied the strategies OFF(CONCAT), OFF(CONCAT, IPB), ENH(CONCAT), and ENH(CONCAT, IPB) each in 61 simulations on trace CONCAT, for budgets 10 ms, 10.5 ms, ..., 40 ms. Figure 5.13 shows the average revenue per frame for each strategy, as measured in the different simulations, as a function of the budget. The graphs for strategies OFF(CONCAT) and OFF(CONCAT, IPB) roughly overlap, as well as the graphs of strategies ENH(CONCAT) and ENH(CONCAT, IPB). Hence, we do not see a clear benefit of using MPEG-2 frame types in the MDP model. On the one hand, the assumption that a frame type can be predicted from the type of the preceding frame only is probably too simple for MPEG-2. This assumption was needed to make the model memoryless. On the other hand, as we have seen in Section 5.4.1, the performance of ENH(CONCAT) is already reasonably close to optimum. Hence, there is not much performance to be won.

# 6

# On-line solution approach

In this chapter we present an on-line solution approach to the QoS control problem, based on the Q-learning algorithm. First, in Section 6.1 we revisit the finite MDP model, to discretize continuous state components differently in the on-line case. Next, in Section 6.2 we discuss the process of learning action values. To reduce the run-time overhead of learning, without reducing the accuracy of learning, in Section 6.3 we introduce a technique called state compression. In Section 6.4 we discuss how action values that are not learned can be approximated from learned action values. Based on all this, in Section 6.5 we present an on-line control strategy. Finally, in Section 6.6 we benchmark this strategy against other strategies by means of simulation experiments.

## 6.1 State discretization revisited

In Section 4.1, to obtain a finite set of states for the MDP model, we divided the interval $[1, \delta]$ of all possible progress values at the start point of a frame into a finite set of subintervals $\Lambda = \{\lambda_1, \ldots, \lambda_{n_\Lambda}\}$. We had to discretize the progress using intervals, to be able to compute state-transition probabilities and expected revenues for the MDP model. These state-transition probabilities and expected revenues are needed when the MDP model is solved off line. Solving the MDP model off line, for the given set of states an optimal state-value function is computed, from

which an optimal static policy is derived. The optimal state-value function is a piecewise-constant approximation of the optimal state-value function for the situation in which there are infinitely many progress intervals, i.e., the situation in which the progress is not discretized. The larger the number $n_\Lambda$ of progress intervals is chosen, the better the approximation becomes.

The Q-learning algorithm is a reinforcement learning algorithm that learns a policy at run time while using the learned policy at the same time for control. The algorithm does not require pre-determined state-transition probabilities and expected revenues. As a result, we can follow a different approach to discretize continuous state components. The idea of this approach is as follows. We define a set $S$ of states that can have both continuous and discrete state components. For each continuous state component we define a finite set of strictly increasing values in the range of all values that the state component can attain, the so-called gridpoints. We define the finite set $S_G$ of gridpoint states as the subset of states from $S$ for which every continuous state component is a gridpoint. We let the Q-learning algorithm only learn action values for the finite set $S_G$ of gridpoint states and for a finite set $A$ of actions. To approximate the action value of a state $s \in S \setminus S_G$ and an action $a \in A$, we apply linear interpolation on the learned action values for the set of gridpoint states and action $a$. The optimal action-value function for the finite set $S_G$ of gridpoint states and the finite set $A$ of actions hence gives a piecewise-linear approximation of the optimal action-value function for the infinite set $S$ of states and set $A$ of actions. The more gridpoint states are defined, the better the approximation becomes. Because a piecewise-constant function approximation is less accurate than a piecewise-linear function approximation, we can define gridpoints at a much lower granularity than intervals to obtain the same accuracy of the approximation; see Figure 6.1.

We now define the set $S$ of states and the set $A$ of actions that we use for Q-learning. Again, we model the set $A$ of actions by the finite set $Q$ of quality levels. We define the state of the SVA at the start point of a frame $f$ as follows. As a first component of the state we use progress $\lambda_\alpha(f)$. Based on the results in Section 5.4.3, we no longer use frame type $\phi(f)$ as a state component. To implement the penalty on quality-level changes, as a second (finite) state component we use the previous quality level. Finally, to take care of dependencies in the processing times of successive frames, as a third state component we use scaled budget $\upsilon(f)$. In Chapter 5 we could not use the scaled budget (or, equivalently, the running complexity factor) as a state component, because the resulting set of states would become too large to solve the MDP model in reasonable time. Because we can now discretize continuous state components at a much lower granularity, we use the scaled budget directly as a state component. We come back to this issue in Section 6.6.3.

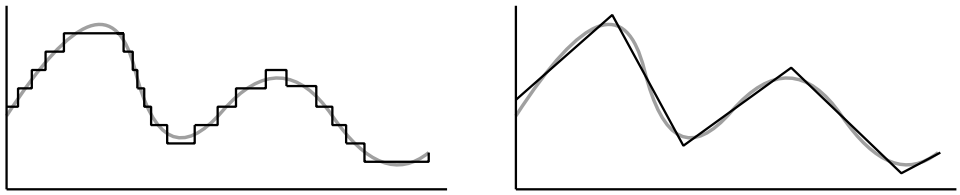

Figure 6.1. Examples of a piecewise-constant function approximation (left figure) and a piecewise-linear function approximation (right figure).

The state given by a progress $\lambda$, a previous quality level $q$, and a scaled budget $\upsilon$ is denoted by $(\lambda, q, \upsilon)$. For a state $s \in S$, we denote the corresponding progress, previous quality level, and scaled budget components by $\lambda_s$, $q_s$, and $\upsilon_s$, respectively. Both the progress and the scaled budget are continuous state components. For the progress, we denote the number of gridpoints by $n_\Lambda$, and we denote the set of increasing gridpoints by $\Lambda = \{\lambda_1, \ldots, \lambda_{n_\Lambda}\}$[1]. We define $\lambda_1 = 1$ and $\lambda_{n_\Lambda} = \delta$, i.e., the lowest and highest possible progress of a frame at its start point. For the scaled budget, we denote the number of gridpoints by $n_\Upsilon$, and we denote the set of increasing gridpoints by $\Upsilon = \{\upsilon_1, \ldots, \upsilon_{n_\Upsilon}\}$.

## 6.2 Learning action values

Let $f$ and $g$ be any pair of frames that are processed in succession. Furthermore, let $s(f) \in S$ be the state of the SVA at the start point of frame $f$, and let $s(g) \in S$ be the state of the SVA at the start point of frame $g$. Recall that the processing time and the revenue of frame $f$ are denoted by $\mu(f)$ and $r(f)$, respectively. In a straightforward control strategy based on the Q-learning algorithm, at the start point of frame $g$ the controller would update action value $Q(s(f), q(f))$ using update rule

$$Q(s(f), q(f)) := (1-\psi)Q(s(f), q(f)) + \psi(r(f) + \gamma \max_{q \in Q} Q(s(g), q)). \qquad (6.1)$$

As mentioned above, we let the Q-learning algorithm only learn action values for the finite set of gridpoint states $S_G$ and the finite set of quality levels $Q$. In (6.1), states $s(f)$ and $s(g)$ are generally not gridpoint states. The $n_Q$ action values $Q(s(g), q)$ can be approximated by interpolating linearly between the learned action values, as will be discussed in Section 6.4. However, it is not possible to update action value $Q(s(f), q(f))$ directly.

Although we cannot directly update action value $Q(s(f), q(f))$, we can update the action value indirectly, via the set of learned action values. More specifically,

[1] To keep notation simple, we reuse symbols that we used earlier to define progress intervals.

using processing time $\mu(f)$ we can update action value $Q(s, q(f))$ for any gridpoint state $s \in S_G$. To this end, we determine the effect of processing frame $f$ at quality level $q(f)$, starting from state $s$ at the start point of frame $f$. We compute the resulting state $s' \in S$ at the start point of frame $g$, and use it to update action value $Q(s, q(f))$.

Before we continue, a short note on the deadline miss approach is in place. Applying the aborting approach, if the deadline of frame $f$ is missed, then the processing of the frame is aborted. In that case, the processing time of frame $f$ is not available at the start point of frame $g$. As a result, it is not possible to update any learned action value. For this reason, in this chapter we only consider the skipping approach.

To compute state $s'$, we have to use a processing time for frame $f$ that matches scaled budget $\upsilon_s$. The actual running complexity factor at the start point of frame $f$ is given by $b/\upsilon_{s(f)}$. This means that frame $f$ is roughly $b/\upsilon_{s(f)}$ times harder than expected. Assuming state $s$ at the start point of frame $f$, the frame is considered to be roughly $b/\upsilon_s$ times harder than expected. To compensate for this difference, for computing the progress component of state $s'$ we use a processing time for frame $f$ given by

$$\mu'(f) = \mu(f) \cdot \frac{\upsilon_{s(f)}}{\upsilon_s}. \tag{6.2}$$

Next, given progress $\lambda_s$, processing time $\mu'(f)$, and budget $b$, we can compute progress component $\lambda_{s'}$ using (2.1)–(2.5). The corresponding number of deadline misses for frame $f$ is given by

$$ndm'(f) = \begin{cases} \lceil \mu'(f)/b - \lambda_s \rceil & \text{if } \lambda_s - \mu'(f)/b < 0 \\ 0 & \text{if } \lambda_s - \mu'(f)/b \geq 0. \end{cases}$$

The previous quality level component of state $s'$ is given by $q_{s'} = q(f)$. Finally, we need the scaled budget component of state $s'$. The scaled budgets of two successive start points are roughly the same, because they are the successive outputs of the same low-pass filter. Hence, we estimate the scaled budget component of state $s'$ by $\upsilon_{s'} = \upsilon_s$. Given state $s' = (\lambda_{s'}, q_{s'}, \upsilon_{s'})$, we can update action value $Q(s, q(f))$ using rule

$$Q(s, q(f)) := (1 - \psi) Q(s, q(f)) + \psi(r'(f) + \gamma \max_{q \in Q} Q(s', q)), \tag{6.3}$$

where revenue $r'(f)$ is given by

$$r'(f) = R_{\text{ql}}(q(f)) - ndm'(f) \cdot P_{\text{dm}} - P_{\text{qlc}}(q_s, q(f)). \tag{6.4}$$

Using this update rule, at the start point of frame $g$ action value $Q(s, q(f))$ can be updated for *all* gridpoint states $s$. We can even go one step further. By estimating the processing time of frame $f$ at different quality levels $q'(f) \in Q$,

action value $Q(s, q'(f))$ can be updated for *all* gridpoint states $s$ and for *all* quality levels $q'(f)$. Recall that $\bar{\mu}(q)$ denotes the expected processing time of a frame processed at quality level $q$. Using pre-determined $\bar{\mu}(q)$-values for the different quality levels $q \in Q$, we can estimate the processing time of frame $f$ at quality level $q'(f) \in Q$ by

$$\tilde{\mu}(f) = \mu(f) \cdot \frac{\bar{\mu}(q'(f))}{\bar{\mu}(q(f))}.$$

By replacing $\mu(f)$ by $\tilde{\mu}(f)$ in (6.2), and by replacing $q(f)$ by $q'(f)$ in (6.3) and (6.4), the update of action value $Q(s, q'(f))$ is given by

$$Q(s, q'(f)) := (1-\psi)\, Q(s, q'(f)) + \psi\,(r'(f) + \gamma \max_{q \in Q} Q(s', q)), \tag{6.5}$$

where revenue $r'(f)$ is given by

$$r'(f) = R_{\mathrm{ql}}(q'(f)) - ndm'(f) \cdot P_{\mathrm{dm}} - P_{\mathrm{qlc}}(q_s, q'(f)).$$

Using this update rule, at the start point of each frame $g > 1$ the controller can update *all* learned action values. As compared to the original Q-learning algorithm, in which only one action value is updated per time step, this will drastically speed up the convergence of learning. Moreover, in the Q-learning algorithm there is no longer the need to take exploring (non-greedy) actions, but the controller can fully exploit what has been learned.

## 6.3 State compression

As mentioned above, at the start point of each frame $g > 1$ the controller can update all learned action values. Doing so, we can make a trade-off between the accuracy of learning and the run-time overhead of the controller by varying the number of gridpoint states. To reduce the number of gridpoint states by a factor $n_Q$, without reducing the accuracy of learning, the following technique can be applied, which we call state compression. The main idea is that the contribution of quality-level changes to action values need not be learned, but can be computed separately.

Therefore, continuing in the line of Section 6.2, we define revenue $r''(f)$ by

$$r''(f) = r'(f) + P_{\mathrm{qlc}}(q_s, q'(f)) = R_{\mathrm{ql}}(q'(f)) - ndm'(f) \cdot P_{\mathrm{dm}}, \tag{6.6}$$

and, for a state $s \in S$ and a quality level $q \in Q$, we define action value $Q'(s, q)$ by

$$Q'(s, q) = Q(s, q) + P_{\mathrm{qlc}}(q_s, q). \tag{6.7}$$

Using these definitions, we derive that

$$\begin{aligned} Q'(s, q'(f)) &= \{\text{using (6.7)}\} \\ &\quad Q(s, q'(f)) + P_{\text{qlc}}(q_s, q'(f)) \\ &:= \{\text{using (6.5)}\} \\ &\quad (1-\psi)Q(s, q'(f)) \\ &\quad +\psi(r'(f) + \gamma \max_{q \in Q} Q(s', q)) + P_{\text{qlc}}(q_s, q'(f)) \\ &= (1-\psi)(Q(s, q'(f)) + P_{\text{qlc}}(q_s, q'(f))) \\ &\quad +\psi(r'(f) + P_{\text{qlc}}(q_s, q'(f)) + \gamma \max_{q \in Q} Q(s', q)) \\ &= \{\text{using (6.6), (6.7), and } q_{s'} = q'(f)\} \\ &\quad (1-\psi)Q'(s, q'(f)) \\ &\quad +\psi(r''(f) + \gamma \max_{q \in Q}(Q'(s', q) - P_{\text{qlc}}(q'(f), q))). \end{aligned}$$

In this update rule, the penalty on quality-level changes is still present, but it only depends on the chosen action $q'(f)$ and the maximization iterator $q$. Hence, to implement this penalty, there is no longer the need to use the previous quality level as a state component. Furthermore, for learning action values $Q'$ we also do not need the previous quality level.

We now define the compressed state of the SVA at the start point of a frame $f \geq 1$ as the combination of progress $\lambda_\alpha(f)$ and scaled budget $\upsilon(f)$. The set of all compressed states is denoted by $\hat{S}$. Next, we define the finite set $\hat{S}_G$ of compressed gridpoint states as the subset of compressed states from $\hat{S}$ for which the progress component and the scaled budget component are both gridpoints. To indicate that a state is a compressed state, we put a hat sign on top of it, i.e., we denote a compressed state $s$ by $\hat{s}$. The compressed state given by a progress $\lambda$ and a scaled budget $\upsilon$ is denoted by $(\lambda, \upsilon)$. For a compressed state $\hat{s} \in \hat{S}$, we denote the corresponding progress and scaled budget components by $\lambda_{\hat{s}}$ and $\upsilon_{\hat{s}}$, respectively.

To reduce the run-time overhead, we let the controller only learn action values $Q'(\hat{s}, q)$ for the finite set $\hat{S}_G$ of compressed gridpoint states and the finite set $Q$ of quality levels. To express the difference in domain, we denote these action values $Q'(\hat{s}, q)$ by $\hat{Q}(\hat{s}, q)$. Assuming state $s \in S_G$ at the start point of frame $f$, state $s' \in S$ at the start point of frame $g$, and quality level $q'(f)$ for frame $f$, we can apply update rule

$$\begin{aligned} \hat{Q}((\lambda_s, \upsilon_s), q'(f)) &:= (1-\psi)\,\hat{Q}((\lambda_s, \upsilon_s), q'(f)) \\ &\quad +\psi\, r''(f) \\ &\quad +\psi\gamma \max_{q \in Q}(\hat{Q}((\lambda_{s'}, \upsilon_{s'}), q) - P_{\text{qlc}}(q'(f), q)). \end{aligned} \quad (6.8)$$

Given the learned action values $\hat{Q}$, the action value $Q(s, q)$ for any gridpoint state $s \in S_G$ is computed by

$$Q(s, q) = \hat{Q}((\lambda_s, \upsilon_s), q) - P_{\text{qlc}}(q_s, q).$$

## 6.4 Approximating action values

To approximate the action value $\hat{Q}(\hat{s}, q)$ of a compressed state $\hat{s} \in \hat{S} \setminus \hat{S}_G$ and a quality level $q \in Q$, we apply linear interpolation on the learned action values $\hat{Q}$ for the set $\hat{S}_G$ of compressed gridpoint states and the set $Q$ of quality levels. More specifically, we first determine integers $i$ $(1 \leq i < n_\Lambda)$ and $j$ $(1 \leq j < n_\Upsilon)$ for which $\lambda_i \leq \lambda_{\hat{s}} \leq \lambda_{i+1}$ and $\upsilon_j \leq \upsilon_{\hat{s}} \leq \upsilon_{j+1}$, respectively. Using $i$ and $j$, we compute fractions $x$ and $y$ by

$$x = \frac{\lambda_{\hat{s}} - \lambda_i}{\lambda_{i+1} - \lambda_i} \quad \text{and} \quad y = \frac{\upsilon_{\hat{s}} - \upsilon_j}{\upsilon_{j+1} - \upsilon_j}, \tag{6.9}$$

respectively. In case $\upsilon_{\hat{s}} < \upsilon_1$ we select $j = 1$ and $y = 0$, and in case $\upsilon_{\hat{s}} > \upsilon_{n_\Upsilon}$ we select $j = n_\Upsilon - 1$ and $y = 1$. The approximation of action value $\hat{Q}(\hat{s}, q)$ is now given by

$$\begin{aligned} \hat{Q}_{\text{LI}}(\hat{s}, q) \quad = \quad & (1-x)\,(1-y)\,\hat{Q}((\lambda_i, \upsilon_j), q) + x\,(1-y)\,\hat{Q}((\lambda_{i+1}, \upsilon_j), q) \\ & + (1-x)\,y\,\hat{Q}((\lambda_i, \upsilon_{j+1}), q) + xy\,\hat{Q}((\lambda_{i+1}, \upsilon_{j+1}), q). \end{aligned} \tag{6.10}$$

We illustrate this interpolation process in Figure 6.2. The figure shows the learned action values for a particular set of compressed gridpoint states and for a particular quality level $q$. The set of compressed gridpoint states is defined by the cross product of sets $\Lambda = \{1, 1.5, 2, 2.5, 3\}$ and $\Upsilon = \{10\,\text{ms}, 20\,\text{ms}, 30\,\text{ms}, 40\,\text{ms}\}$. To approximate the action value of compressed state $\hat{s} = (2.25, 27\,\text{ms})$ (the black dot in the figure), we interpolate linearly between the learned action values for compressed gridpoint states $(2, 20\,\text{ms})$, $(2.5, 20\,\text{ms})$, $(2, 30\,\text{ms})$, and $(2.5, 30\,\text{ms})$. Using (6.9) and (6.10), we approximate the action value of state $\hat{s}$ and quality level $q$ by $\hat{Q}_{\text{LI}}(\hat{s}, q) = 0.5 \cdot 0.3 \cdot 13 + 0.5 \cdot 0.3 \cdot 24 + 0.5 \cdot 0.7 \cdot 27 + 0.5 \cdot 0.7 \cdot 38 = 28.3$.

## 6.5 On-line strategy

Figure 6.3 shows pseudo-code for a control strategy based on the Q-learning algorithm, according to the approach described in Sections 6.1 to 6.4. We call this control strategy the *on-line strategy*. The only data that the strategy needs in advance are $\bar{\mu}(q)$-values for the different quality levels $q \in Q$. These values can be derived from a suitable trace.

An important issue is the run-time overhead of the on-line strategy. At each start point of a frame $f > 1$, the strategy updates action value $\hat{Q}(\hat{s}, q)$ for each compressed gridpoint state $\hat{s} \in \hat{S}_G$ and for each quality level $q \in Q$, using (6.8). In

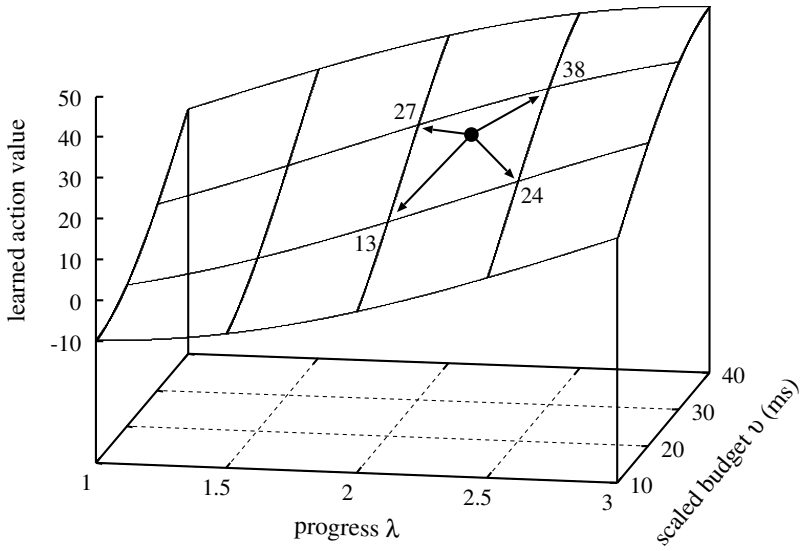


Figure 6.2. The learned action values for a set of compressed gridpoint states defined by the cross product of sets $\Lambda = \{1, 1.5, 2, 2.5, 3\}$ and $\Upsilon = \{10\,\text{ms}, 20\,\text{ms}, 30\,\text{ms}, 40\,\text{ms}\}$, and for a particular quality level. The action value of a compressed state is approximated by interpolating linearly between the learned action values for the set of compressed gridpoint states.

this update rule $n_Q$ values are computed, of which only the highest value is used to update $\hat{Q}(\hat{s}, q)$. Hence, the time complexity of the on-line strategy at the start point of a frame $f > 1$ is given by $\mathcal{O}(|\hat{S}_G| \cdot |Q|^2)$. To obtain a low run-time overhead, the number of compressed gridpoint states should not be chosen too large. We get back on this issue in the Section 6.6.3.

---

{*start point of frame* $f = 1$}
**input:**
**output:** quality level sel_ql for frame $f$

**for** each $\hat{s} \in \hat{S}_G$ **do**
    **for** each $q \in Q$ **do**
        initialize $\hat{Q}(\hat{s}, q)$ arbitrarily;
$\upsilon := b$;
sel_ql := Random$\{q_1, \ldots, q_{n_Q}\}$;
prev_ql := sel_ql;

---

{*start point of a frame* $f > 1$}
**input:** progress $\lambda_\alpha(f)$, processing time $\mu$ of the just-processed frame
**output:** quality level sel_ql for frame $f$

**for** each $\hat{s} \in \hat{S}_G$ **do**
    **for** each $q \in Q$ **do**
    **begin**
        $\lambda := \lambda_{\hat{s}} - (\mu/b) \cdot (\bar{\mu}(q)/\bar{\mu}(\text{prev\_ql})) \cdot (\upsilon/\upsilon_{\hat{s}})$;
        **if** $\lambda < 0$
        **then begin**
                $r := R_{\text{ql}}(q) - P_{\text{dm}} \cdot \lceil -\lambda \rceil$;
                $\lambda := \lambda + \lceil -\lambda \rceil$;
            **end**
        **else** $r := R_{\text{ql}}(q)$;
        $\lambda := \lambda + 1$;
        **if** $\lambda > \delta$
        **then** $\lambda := \delta$;
        $\hat{Q}(\hat{s}, q) := (1 - \psi)\hat{Q}(\hat{s}, q) + \psi(r + \gamma \max_{q' \in Q}(\hat{Q}_{\text{LI}}((\lambda, \upsilon_{\hat{s}}), q') - P_{\text{qlc}}(q, q')))$;
    **end**;
update scaled budget $\upsilon$ using processing time $\mu$;
sel_ql := $\arg\max_{q \in Q}(\hat{Q}_{\text{LI}}((\lambda_\alpha(f), \upsilon), q) - P_{\text{qlc}}(\text{prev\_ql}, q))$;
prev_ql := sel_ql;

---

Figure 6.3. Pseudo-code for the on-line strategy.

## 6.6 Simulation experiments

To evaluate the on-line strategy, we have run simulation experiments. These experiments are a continuation of the experiments described in Sections 4.4 and 5.4.

### Settings

Given the default parameter settings defined in Sections 4.4 and 5.4, for the on-line strategy we add the following two default parameter settings.

- We use a learning rate $\psi = 0.01$, i.e., action values are updated using a weight of 1% for every new experience, and a weight of 99% for built-up experience. We evaluate other learning rates in Section 6.6.2.
- We use a set of 63 compressed gridpoint states, given by the cross product of set $\Lambda = \{1, 1.25, \ldots, 3\}$ and set $\Upsilon = \{10\,\text{ms}, 15\,\text{ms}, \ldots, 40\,\text{ms}\}$; other sets of compressed gridpoint states are evaluated in Section 6.6.3.

We use ON($\tau$) as a shorthand notation for the on-line strategy, using trace $\tau$ as statistics trace. Trace $\tau$ is used before run time to compute values for the expected processing time per frame at the different quality levels. These $\bar{\mu}(q)$-values, which comprise the only off-line information needed by ON($\tau$), are used at run time to estimate the processing times of frames at different quality levels, and to compute the complexity factors of completed frames. For any change in the parameter settings with respect to the default settings, we extend the shorthand notation. For example, if we use a learning rate of 0.1 instead of 0.01, we denote the strategy by ON($\tau, \psi = 0.1$).

### Experiment overview

First, in Section 6.6.1 we study the behavior of the on-line strategy for the default settings, and we benchmark the strategy against the off-line strategy, the enhanced strategy, strategy Q4, and the clairvoyant strategy. Next, in Sections 6.6.2 and 6.6.3 we vary learning rate $\psi$ and the set of compressed gridpoint states, respectively. In Section 6.6.4 we study the effect of using a simulation trace that differs from the applied statistics trace. Finally, in Section 6.6.5 we study the creation of gain time by the SVA for the various control strategies.

### 6.6.1 Results for the default settings

To study the behavior of the on-line strategy for the default settings, and to benchmark the strategy against other strategies, we applied the strategies Q4, OFF(CONCAT), ENH(CONCAT), ON(CONCAT), and CLV each in 61 simulations on trace CONCAT, for budgets 10 ms, 10.5 ms, ..., 40 ms. Figures 6.4 and 6.5 show the average revenue per frame and the number of deadline misses, respectively, that we measured in the different simulations, as a function of the budget.

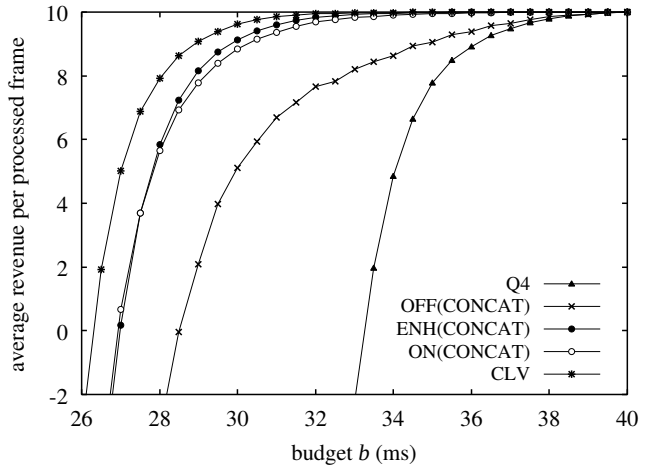


Figure 6.4. The average revenue for strategies Q4, OFF(CONCAT), ENH(CONCAT), ON(CONCAT), and CLV applied in simulations on trace CONCAT, as a function of the budget.

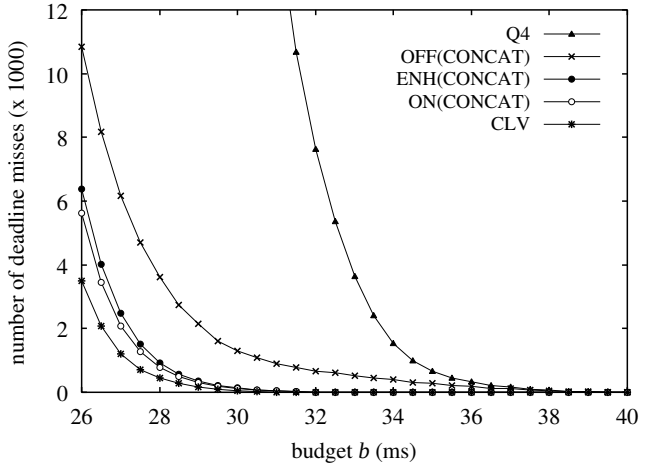


Figure 6.5. The number of deadline misses for strategies Q4, OFF(CONCAT), ENH(CONCAT), ON(CONCAT), and CLV applied in simulations on trace CONCAT, as a function of the budget.

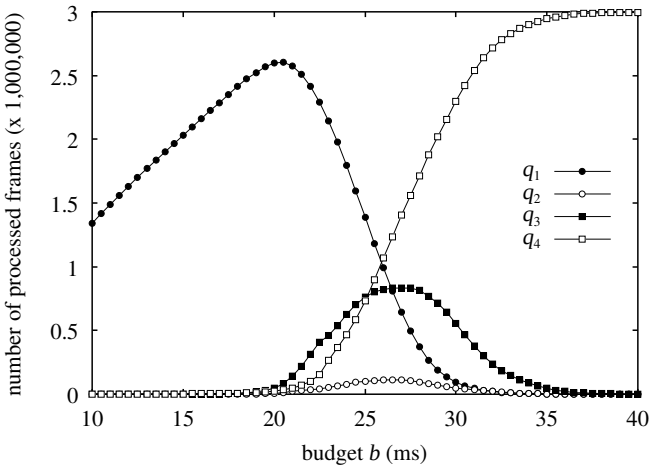


Figure 6.6. The number of frames processed at each quality level, as a function of the budget, for strategy ON(CONCAT) applied in simulations on trace CONCAT.

Table 6.1. The budgets required to attain different values of the average revenue, for strategies CLV, OFF(CONCAT), ENH(CONCAT), and ON(CONCAT) applied in simulations on trace CONCAT.

| average revenue | CLV | OFF(CONCAT) | ENH(CONCAT) | ON(CONCAT) |
|---|---|---|---|---|
| 0.0 | 26.3 ms | 28.5 ms (+2.2 ms) | 27.0 ms (+0.7 ms) | 26.9 ms (+0.6 ms) |
| 1.0 | 26.4 ms | 28.7 ms (+2.3 ms) | 27.1 ms (+0.7 ms) | 27.1 ms (+0.7 ms) |
| 2.0 | 26.5 ms | 29.0 ms (+2.5 ms) | 27.3 ms (+0.8 ms) | 27.2 ms (+0.7 ms) |
| 3.0 | 26.7 ms | 29.2 ms (+2.5 ms) | 27.4 ms (+0.7 ms) | 27.4 ms (+0.7 ms) |
| 4.0 | 26.8 ms | 29.5 ms (+2.7 ms) | 27.6 ms (+0.8 ms) | 27.6 ms (+0.8 ms) |
| 5.0 | 27.0 ms | 30.0 ms (+3.0 ms) | 27.8 ms (+0.8 ms) | 27.8 ms (+0.8 ms) |
| 6.0 | 27.3 ms | 30.5 ms (+3.2 ms) | 28.1 ms (+0.8 ms) | 28.1 ms (+0.8 ms) |
| 7.0 | 27.6 ms | 31.3 ms (+3.7 ms) | 28.4 ms (+0.8 ms) | 28.5 ms (+0.9 ms) |
| 8.0 | 28.0 ms | 32.7 ms (+4.7 ms) | 28.9 ms (+0.9 ms) | 29.2 ms (+1.2 ms) |
| 9.0 | 28.9 ms | 34.8 ms (+5.9 ms) | 29.8 ms (+0.9 ms) | 30.3 ms (+1.4 ms) |
| 9.9 | 31.4 ms | 38.4 ms (+7.0 ms) | 32.6 ms (+1.2 ms) | 34.0 ms (+2.6 ms) |

Next, Figure 6.6 shows the number of frames that were processed at each quality level, as a function of the budget, for strategy ON(CONCAT).

We observe that the on-line strategy performs roughly the same as the enhanced strategy, and that it clearly outperforms strategies Q4 and OFF(CONCAT). For the on-line strategy, the number of deadline misses drops to zero at a budget of 34 ms. For budgets smaller than 34 ms, the on-line strategy produces slightly fewer deadline misses than the enhanced strategy, with some minor exceptions for small budgets. In Figure 6.6 we see that the on-line strategy uses the different quality levels smoothly balanced over the budget spectrum. If we compare this figure with Figure 5.10, then we observe that the on-line strategy is a little more conservative than the enhanced strategy, in the sense that it selects lower quality levels more often for a given budget. As a result, despite the smaller number of deadline misses, the average revenue for the on-line strategy is slightly lower than the average revenue for the enhanced strategy, for budgets larger than 27 ms.

Finally, Table 6.1 shows the various budgets that are required to attain different values of the average revenue, for strategies CLV, OFF(CONCAT), ENH(CONCAT), and ON(CONCAT). These values were computed using linear interpolation between the simulation results for the 61 budgets. Again we observe that the on-line strategy performs very close to the enhanced strategy. To attain a high average revenue, the enhanced strategy performs a little better than the on-line strategy. Nevertheless, the on-line strategy still performs a lot better than the off-line strategy. Apparently, the on-line strategy learns sufficiently fast.

### 6.6.2 Varying learning rate $\psi$

To study the influence of learning rate $\psi$ on the results we obtained for the default settings, we defined a set $\Psi$ with twelve candidate learning rates: $0.001, 0.01, 0.05, 0.1, 0.15, \ldots, 0.5$. For each $\psi \in \Psi$, we applied strategy ON(CONCAT, $\psi$) in 61 simulations on trace CONCAT, for budgets 10 ms, 10.5 ms, ..., 40 ms. Note that strategy ON(CONCAT, $\psi = 0.01$) is the same as strategy ON(CONCAT). Figure 6.7 shows the average revenue that we measured in the simulations for the different strategies, as a function of $\psi$, for budgets between 27 ms and 31 ms.

For small budgets the average revenue is strongly negative for every $\psi \in \Psi$. As the budget increases, the average revenue turns positive at $b = 27$ ms, for values of $\psi$ in the range $[0.001, 0.3]$. In Figure 6.7, the distance between successive budget lines becomes smaller, which is due to the convergence of the average revenue to 10 as the budget increases. The budget lines, which are pretty flat, show maximum average revenues for values of $\psi$ in the range $[0.01, 0.05]$.

We came to the default setting $\psi = 0.01$ in a similar way as we determined $\theta$ in Section 5.4.2. Let $x(b, \psi)$ denote the average revenue that we measured in the

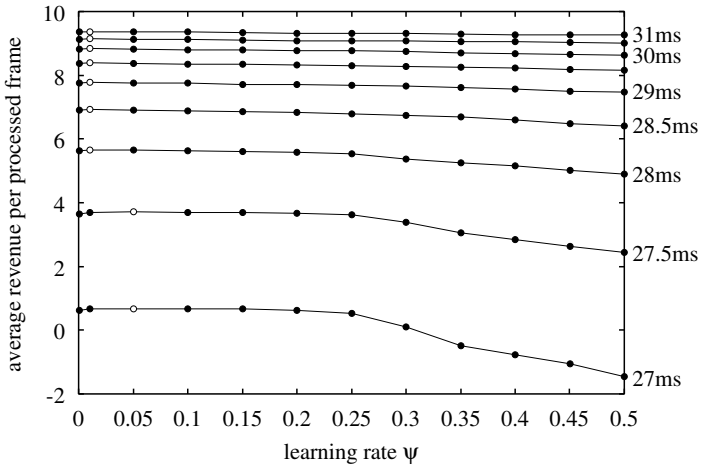


Figure 6.7. The average revenue for different strategies ON(CONCAT, ψ) applied in simulations on trace CONCAT, as a function of learning rate ψ, for budgets between 27 ms and 31 ms. For each budget we indicate the highest average revenue over all twelve values of ψ by means of a white dot.

Table 6.2. Error measure $e(B, \psi)$ for different budgets sets $B$, and for each $\psi \in \Psi$.

| $\psi$ | $e(B(10, 40), \psi)$ | $e(B(27, 40), \psi)$ | $e(B(27, 31), \psi)$ |
|---|---|---|---|
| 0.001 | 10.21 | 0.43 | 0.25 |
| 0.01 | 9.33 | 0.03 | 0.03 |
| 0.05 | 7.91 | 0.13 | 0.07 |
| 0.1 | 4.64 | 0.33 | 0.20 |
| 0.15 | 7.48 | 0.54 | 0.35 |
| 0.2 | 18.15 | 0.80 | 0.55 |
| 0.25 | 90.62 | 1.17 | 0.86 |
| 0.3 | 365.36 | 2.18 | 1.83 |
| 0.35 | 859.35 | 3.42 | 3.02 |
| 0.4 | 1485.46 | 4.26 | 3.82 |
| 0.45 | 2189.32 | 5.21 | 4.73 |
| 0.5 | 2909.38 | 6.14 | 5.62 |

simulation for budget $b$ and learning rate $\psi$. Furthermore, let $B(i, j)$ denote the subset of budgets from set $\{10\,\text{ms}, 10.5\,\text{ms}, \ldots, 40\,\text{ms}\}$ in the range $[i\,\text{ms}, j\,\text{ms}]$. For a set of budgets $B$ we define error measure $e(B, \psi)$ by

$$e(B, \psi) = \sum_{b \in B} (\max_{\psi' \in \Psi} x(b, \psi') - x(b, \psi)).$$

Table 6.2 shows $e(B, \psi)$ for the different values $\psi \in \Psi$, and for the budget sets $B(10, 40)$, $B(27, 40)$, and $B(27, 31)$. The smaller the error measure for a given set of budgets $B$, the better. Because we are interested in a positive average revenue, we focus on budgets of 27 ms and larger. Hence, based on the results in Table 6.2, we have selected $\psi = 0.01$ as default learning rate.

### 6.6.3 Varying the set of compressed gridpoint states

To study the influence of the set of compressed gridpoint states on the results we obtained for the default settings, we defined five sets of compressed gridpoint states, named CGS1 to CGS5; see Table 6.3. For each set CGS of compressed gridpoint states, we applied strategy ON(CONCAT, CGS) in 61 simulations on trace CONCAT, for budgets 10 ms, 10.5 ms, ..., 40 ms. Note that strategy ON(CONCAT, CGS4) is the same as strategy ON(CONCAT). Figure 6.8 shows the average revenue that we measured in the simulations for the different strategies, as a function of the budget.

Table 6.3. The different sets of compressed gridpoint states.

| set | progress gridpoints $\Lambda$ | scaled budget gridpoints $\Upsilon$ | number of states |
|---|---|---|---|
| CGS1 | $\{1, 3\}$ | $\{10\,\text{ms}, 40\,\text{ms}\}$ | 4 |
| CGS2 | $\{1, 2, 3\}$ | $\{10\,\text{ms}, 25\,\text{ms}, 40\,\text{ms}\}$ | 9 |
| CGS3 | $\{1, 1.5, \ldots, 3\}$ | $\{10\,\text{ms}, 17.5\,\text{ms}, \ldots, 40\,\text{ms}\}$ | 25 |
| CGS4 | $\{1, 1.25, \ldots, 3\}$ | $\{10\,\text{ms}, 15\,\text{ms}, \ldots, 40\,\text{ms}\}$ | 63 |
| CGS5 | $\{1, 1.1, \ldots, 3\}$ | $\{10\,\text{ms}, 11\,\text{ms}, \ldots, 40\,\text{ms}\}$ | 651 |

As expected, a larger set of compressed gridpoint states results in a better performance of the on-line strategy. Moreover, we observe that we can indeed define gridpoints at a much lower granularity than intervals to obtain the same accuracy of the approximation. Although sets CGS1 and CGS2 are clearly too small to be useful, for set CGS3 the on-line strategy already gives an acceptable result. For set CGS4 the on-line strategy performs roughly the same as the enhanced strategy, and reasonably close to the optimum given by strategy CLV. For set CGS5 the on-line strategy even outperforms the enhanced strategy, for budgets between 19.5 ms and 30.5 ms. It is unlikely that a set of compressed gridpoint states larger than CGS5 will further improve the performance of the on-line strategy significantly.

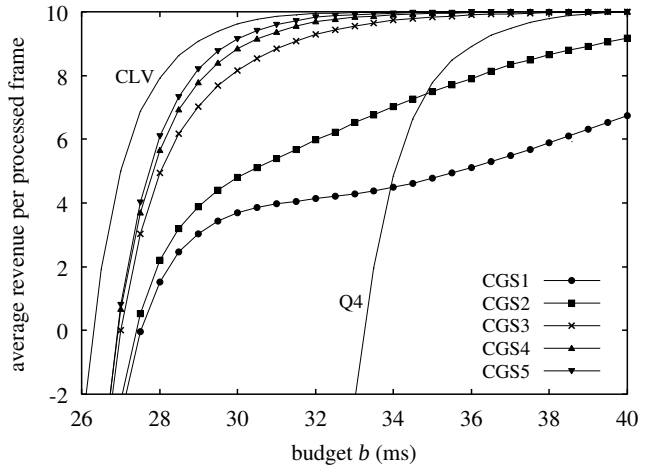


Figure 6.8. The average revenue for strategies ON(CONCAT, CGS1) to ON(CONCAT, CGS5) applied in simulations on trace CONCAT, as a function of the budget.

The larger the used set of compressed gridpoint states, the higher is the run-time overhead of the on-line strategy. Hence, we have to make a trade-off between the performance of the on-line strategy and its run-time overhead. As mentioned in Section 6.5, the time complexity of the on-line strategy at the start point of a frame $f > 1$ is given by $\mathcal{O}(|\hat{S}_G| \cdot |Q|^2)$. For set CGS5 we have $|\hat{S}_G| \cdot |Q|^2 = 10,416$. Given a video signal with a picture resolution of $720 \times 576$ pixels, for each frame there are 414,720 individual pixels to be computed. Hence, even for set CGS5 we consider the run-time overhead of the on-line strategy to be acceptable. Because strategy ON(CONCAT, CGS5) performs only slightly better than strategy ON(CONCAT, CGS4), but at the cost of an over times times higher run-time overhead, we have selected set CGS4 as the default set of compressed gridpoint states.

### 6.6.4 Cross-trace experiments

So far, in simulation experiments we always used trace CONCAT as statistics trace for the off-line strategy, the enhanced strategy, and the on-line strategy. Moreover, we always used either trace ARTIFICIAL or trace CONCAT as simulation trace. In this section we study the effect of using different statistics traces and different simulation traces on the performance of the strategies. To this end, we applied each of the strategies CLV, Q4, the 18 strategies OFF(A) to OFF(R), the 18 strategies ENH(A) to ENH(R), and the 18 strategies ON(A) to ON(R) in simulations on traces A to R, for budgets 10 ms, 10.5 ms, ..., 40 ms. Table 6.4 gives the aver-

age revenues for all simulations that were carried out. Because we are interested in both a positive average revenue and a smaller than worst-case budget, for each simulation trace we first identified a 'sensible' budget range, which is given by the subset of budgets from set $\{10\,\text{ms}, 10.5\,\text{ms}, \ldots, 40\,\text{ms}\}$ for which strategy CLV attains an average revenue in the range $[0, 9.9]$. The average revenues in Table 6.4 are computed over the specified budget ranges.

As expected, strategy Q4 performs poorly on all 18 simulation traces. The different off-line strategies consistently perform better than Q4. However, as can be seen from the large deviations in the average revenue, the off-line strategy is very sensitive to the used statistics trace. We observe that choosing the statistics trace the same as the simulation trace is not always preferable. For example, using statistics trace H, in simulations on trace H the off-line strategy results in an average revenue of 1.89, whereas most other statistics traces result in a higher average revenue in simulations on trace H. This may be explained as follows. If we average over all 18 simulation traces, then the off-line strategy performs the best using statistics trace J, and the strategy performs the worst using statistics trace P. Of all traces A to R, trace J is the trace with the highest average processing times for the different quality levels, and trace P is the trace with the lowest average processing times; see Table 4.3. Hence, using statistics trace J, the off-line strategy expects on average low processing times. As a result, if the processing times in a simulation are lower, then the strategy selects the quality levels for frames too low. This is an advantage in structurally hard scenes, because fewer deadlines than normal are missed, and deadline misses are heavily penalized. Similarly, using statistics trace P, the off-line strategy expects on average low processing times. As a result, if the processing times in a simulation are higher, then the strategy selects the quality levels for frames too high. This is a disadvantage in structurally hard scenes, because more deadlines than normal are missed.

The 18 enhanced strategies perform much closer to optimum. For each simulation trace, the deviations in the average revenues for the different statistics traces are much smaller than for the off-line strategy. The same holds for the 18 on-line strategies. On average, the on-line strategy performs slightly worse than the enhanced strategy. We observe that the enhanced strategy and the on-line strategy are quite insensitive to using the statistics of a different trace. This is a favorable result, as in practice the strategies are applied to unknown video content.

For the on-line strategy, the cross relations between the statistics traces and the simulation traces are negligible. Since the on-line strategy learns statistics at run time, the only 'statistics' used are the expected processing times for the different quality levels. Because the on-line strategy performs roughly the same as the enhanced strategy, we again conclude that the on-line strategy learns sufficiently fast.

Table 6.4. The average revenues for the cross-trace experiments.

| simulation trace:<br>sensible budget range: | A<br>27–32 ms | B<br>27–32 ms | C<br>26.5–31.5 ms | D<br>25–30 ms | E<br>25–29.5 ms | F<br>24–28 ms |
|---|---|---|---|---|---|---|
| CLV | 8.21 | 8.87 | 8.19 | 8.77 | 8.72 | 8.43 |
| Q4 | −279.27 | −335.50 | −399.34 | −523.20 | −337.80 | −272.02 |
| OFF(A) | 0.67 | 3.85 | 1.80 | **7.55** | **6.63** | 5.46 |
| OFF(B) | 0.69 | 3.86 | 1.96 | 7.38 | 6.46 | 4.86 |
| OFF(C) | −1.27 | 2.70 | 1.06 | 7.36 | 6.43 | 4.86 |
| OFF(D) | −53.71 | −43.33 | −40.99 | 5.53 | 1.79 | 5.64 |
| OFF(E) | −66.37 | −58.91 | −54.39 | 4.23 | −0.78 | 5.20 |
| OFF(F) | −128.93 | −132.97 | −124.89 | −10.71 | −24.65 | 0.91 |
| OFF(G) | −83.52 | −79.94 | −72.80 | 2.20 | −1.57 | 4.77 |
| OFF(H) | −118.40 | −120.26 | −112.53 | −4.52 | −16.27 | 1.59 |
| OFF(I) | −17.50 | −10.09 | −11.06 | 7.13 | 5.65 | **6.27** |
| OFF(J) | **3.80** | **5.91** | **4.09** | 6.31 | 5.36 | 3.60 |
| OFF(K) | −24.99 | −16.23 | −15.79 | 6.81 | 5.34 | 5.02 |
| OFF(L) | −62.46 | −53.45 | −50.74 | 4.11 | −1.43 | 4.71 |
| OFF(M) | −65.72 | −58.17 | −54.25 | 3.90 | −1.53 | 5.11 |
| OFF(N) | −72.55 | −65.74 | −59.99 | 3.08 | −3.26 | 4.66 |
| OFF(O) | −68.65 | −61.04 | −55.42 | 4.28 | −0.97 | 4.66 |
| OFF(P) | **−272.69** | **−326.63** | **−381.70** | **−426.21** | **−261.71** | **−142.26** |
| OFF(Q) | −88.93 | −83.94 | −82.33 | 0.18 | −7.62 | 2.22 |
| OFF(R) | −77.15 | −70.40 | −65.98 | 2.55 | −4.04 | 3.87 |
| OFF(CONCAT) | −6.06 | −1.40 | −4.78 | 7.12 | 5.56 | 6.01 |
| AVG(OFF(A)…OFF(R)) | −66.54 | −64.71 | −65.22 | −20.49 | −15.90 | −3.82 |
| STDEV(OFF(A)…OFF(R)) | 63.50 | 75.25 | 85.33 | 98.50 | 60.15 | 33.60 |
| ENH(A) | **5.93** | **6.93** | **5.35** | **7.62** | **7.08** | **6.85** |
| ENH(B) | 5.75 | 6.86 | 5.14 | 7.69 | 7.04 | 6.80 |
| ENH(C) | 5.71 | 6.84 | 5.11 | 7.74 | 7.03 | 6.80 |
| ENH(D) | **4.43** | **5.81** | **3.73** | 7.73 | **6.41** | **6.42** |
| ENH(E) | 5.05 | 6.35 | 4.39 | 7.76 | 6.70 | 6.60 |
| ENH(F) | 4.84 | 6.20 | 4.17 | 7.66 | 6.64 | 6.51 |
| ENH(G) | 4.90 | 6.22 | 4.14 | 7.76 | 6.63 | 6.46 |
| ENH(H) | 5.16 | 6.48 | 4.50 | 7.68 | 6.79 | 6.62 |
| ENH(I) | 5.47 | 6.62 | 4.74 | 7.65 | 6.87 | 6.67 |
| ENH(J) | 5.52 | 6.64 | 4.77 | 7.71 | 6.92 | 6.65 |
| ENH(K) | 5.14 | 6.45 | 4.44 | **7.81** | 6.77 | 6.55 |
| ENH(L) | 5.48 | 6.65 | 4.74 | 7.66 | 6.90 | 6.69 |
| ENH(M) | 5.43 | 6.60 | 4.66 | 7.69 | 6.90 | 6.65 |
| ENH(N) | 5.29 | 6.55 | 4.58 | 7.77 | 6.82 | 6.64 |
| ENH(O) | 4.74 | 6.05 | 4.03 | 7.77 | 6.56 | **6.42** |
| ENH(P) | 5.21 | 6.49 | 4.48 | 7.69 | 6.79 | 6.64 |
| ENH(Q) | 5.54 | 6.68 | 4.85 | 7.69 | 6.98 | 6.74 |
| ENH(R) | 5.38 | 6.59 | 4.68 | 7.66 | 6.88 | 6.70 |
| ENH(CONCAT) | 5.50 | 6.65 | 4.78 | 7.74 | 6.98 | 6.69 |
| AVG(ENH(A)…ENH(R)) | 5.28 | 6.50 | 4.58 | 7.71 | 6.82 | 6.63 |
| STDEV(ENH(A)…ENH(R)) | 0.37 | 0.28 | 0.40 | 0.05 | 0.17 | 0.12 |
| ON(A) | 5.72 | 6.59 | 5.10 | **7.57** | **6.94** | **6.67** |
| ON(B) | 5.69 | 6.46 | 4.98 | 7.38 | 6.90 | 6.57 |
| ON(C) | 5.67 | 6.43 | 4.96 | 7.41 | 6.91 | 6.59 |
| ON(D) | 5.78 | 6.61 | **5.19** | 7.28 | 6.75 | 6.59 |
| ON(E) | 5.74 | 6.55 | 5.10 | 7.12 | 6.67 | 6.50 |
| ON(F) | 5.39 | 6.03 | 4.50 | 6.04 | 6.08 | 5.89 |
| ON(G) | 5.73 | 6.61 | 5.13 | 7.53 | 6.87 | 6.63 |
| ON(H) | 5.19 | 5.74 | 4.15 | 5.75 | 5.67 | 5.49 |
| ON(I) | 5.75 | 6.58 | 5.15 | 7.27 | 6.75 | 6.57 |
| ON(J) | 5.63 | 6.31 | 4.77 | 6.57 | 6.30 | 6.14 |
| ON(K) | 5.70 | 6.46 | 4.97 | 7.41 | 6.84 | 6.62 |
| ON(L) | 5.46 | 6.16 | 4.69 | 6.20 | 6.28 | 6.09 |
| ON(M) | 5.38 | 5.97 | 4.36 | 5.81 | 5.96 | 5.74 |
| ON(N) | 5.72 | 6.55 | 5.07 | 6.99 | 6.46 | 6.41 |
| ON(O) | **5.79** | **6.65** | **5.19** | 7.18 | 6.65 | 6.51 |
| ON(P) | **5.11** | 5.71 | 4.28 | 6.88 | 6.54 | 6.14 |
| ON(Q) | 5.13 | **5.62** | **4.01** | **5.65** | **5.57** | **5.37** |
| ON(R) | 5.31 | 5.90 | 4.29 | 5.74 | 5.91 | 5.72 |
| ON(CONCAT) | 5.76 | 6.60 | 5.13 | 7.12 | 6.66 | 6.49 |
| AVG(ON(A)…ON(R)) | 5.55 | 6.27 | 4.77 | 6.77 | 6.45 | 6.24 |
| STDEV(ON(A)…ON(R)) | 0.23 | 0.35 | 0.39 | 0.69 | 0.43 | 0.42 |

Table 6.4. (continued)

| simulation trace:<br>sensible budget range: | G<br>24.5–28.5 ms | H<br>24.5–28.5 ms | I<br>26–30.5 ms | J<br>29.5–34.5 ms | K<br>25–30.5 ms | L<br>25–29.5 ms |
|---|---|---|---|---|---|---|
| CLV | 8.43 | 9.17 | 8.46 | 8.29 | 8.24 | 8.78 |
| Q4 | −480.99 | −218.75 | −270.73 | −227.00 | −663.58 | −265.24 |
| OFF(A) | **6.44** | 6.90 | 5.43 | −30.54 | 5.82 | **6.29** |
| OFF(B) | 6.07 | 6.40 | 5.46 | −27.61 | **6.01** | 6.16 |
| OFF(C) | 6.06 | 6.40 | 5.24 | −50.51 | 5.90 | 6.13 |
| OFF(D) | 5.18 | 6.65 | −11.90 | −164.01 | −2.68 | 1.49 |
| OFF(E) | 3.47 | 5.85 | −17.34 | −163.29 | −6.75 | −0.99 |
| OFF(F) | −13.21 | −0.01 | −64.13 | −198.42 | −61.88 | −16.78 |
| OFF(G) | 2.88 | 5.38 | −27.36 | −183.05 | −8.53 | −1.52 |
| OFF(H) | −6.44 | 1.89 | −54.95 | −195.69 | −41.09 | −11.98 |
| OFF(I) | 6.38 | **7.36** | 1.44 | −101.26 | 3.56 | 5.02 |
| OFF(J) | 4.77 | 4.95 | **5.55** | **−1.53** | 5.62 | 5.07 |
| OFF(K) | 5.83 | 6.36 | −1.03 | −119.30 | 4.26 | 4.61 |
| OFF(L) | 2.74 | 5.55 | −15.24 | −159.72 | −7.53 | −1.51 |
| OFF(M) | 2.72 | 5.63 | −17.56 | −169.88 | −9.28 | −1.51 |
| OFF(N) | 2.34 | 5.46 | −23.90 | −179.85 | −12.33 | −3.02 |
| OFF(O) | 3.21 | 5.79 | −18.71 | −175.80 | −7.65 | −0.72 |
| OFF(P) | **−292.83** | **−144.63** | **−254.82** | **−226.99** | **−547.53** | **−203.23** |
| OFF(Q) | −1.57 | 3.31 | −31.81 | −172.69 | −19.31 | −7.72 |
| OFF(R) | 0.57 | 4.64 | −23.77 | −175.98 | −14.18 | −3.52 |
| OFF(CONCAT) | 6.20 | 7.27 | 2.44 | −32.64 | 3.30 | 4.88 |
| AVG(OFF(A)…OFF(R)) | −14.19 | −3.12 | −29.97 | −138.67 | −39.31 | −12.10 |
| STDEV(OFF(A)…OFF(R)) | 67.75 | 34.37 | 57.86 | 65.52 | 124.43 | 46.77 |
| ENH(A) | 6.64 | **8.09** | **6.52** | 5.88 | 5.79 | **6.70** |
| ENH(B) | 6.71 | 8.06 | 6.42 | 5.99 | 5.86 | 6.50 |
| ENH(C) | 6.65 | 8.05 | 6.37 | 5.98 | 5.81 | 6.45 |
| ENH(D) | 6.66 | **7.59** | **5.37** | 5.84 | 5.61 | **5.46** |
| ENH(E) | 6.74 | 7.84 | 5.93 | 5.88 | 5.65 | 5.86 |
| ENH(F) | **6.60** | 7.79 | 5.73 | **5.83** | **5.49** | 5.75 |
| ENH(G) | 6.77 | 7.77 | 5.73 | 5.95 | 5.71 | 5.78 |
| ENH(H) | 6.64 | 7.87 | 5.93 | 5.90 | 5.66 | 5.95 |
| ENH(I) | 6.62 | 7.96 | 6.16 | 5.92 | 5.76 | 6.23 |
| ENH(J) | **6.88** | 7.95 | 6.13 | **6.18** | **6.05** | 6.33 |
| ENH(K) | 6.87 | 7.89 | 5.90 | 6.07 | 5.95 | 6.03 |
| ENH(L) | 6.69 | 7.96 | 6.16 | 5.92 | 5.78 | 6.23 |
| ENH(M) | 6.76 | 7.93 | 6.20 | 5.96 | 5.86 | 6.21 |
| ENH(N) | 6.81 | 7.94 | 5.97 | 6.04 | 5.92 | 5.97 |
| ENH(O) | 6.73 | 7.70 | 5.57 | 5.97 | 5.71 | 5.68 |
| ENH(P) | 6.71 | 7.89 | 6.03 | 5.95 | 5.72 | 6.09 |
| ENH(Q) | 6.77 | 8.02 | 6.24 | 5.96 | 5.91 | 6.36 |
| ENH(R) | 6.64 | 7.94 | 6.15 | 5.90 | 5.70 | 6.18 |
| ENH(CONCAT) | 6.74 | 7.99 | 6.20 | 6.01 | 5.90 | 6.28 |
| AVG(ENH(A)…ENH(R)) | 6.72 | 7.90 | 6.03 | 5.95 | 5.77 | 6.10 |
| STDEV(ENH(A)…ENH(R)) | 0.08 | 0.13 | 0.29 | 0.08 | 0.13 | 0.31 |
| ON(A) | 6.34 | **7.96** | 6.26 | **5.86** | 5.61 | 6.48 |
| ON(B) | **6.38** | 7.92 | 6.24 | 5.62 | 5.67 | **6.51** |
| ON(C) | **6.38** | 7.91 | 6.22 | 5.56 | 5.68 | 6.49 |
| ON(D) | 5.78 | 7.86 | 6.25 | 5.56 | 5.19 | 6.31 |
| ON(E) | 5.98 | 7.77 | 6.10 | 5.28 | 5.08 | 6.30 |
| ON(F) | 4.98 | 7.34 | 5.77 | 4.98 | 4.38 | 5.97 |
| ON(G) | 6.28 | 7.95 | **6.37** | **5.86** | 5.58 | 6.46 |
| ON(H) | 4.47 | 7.06 | 5.45 | 4.65 | 4.03 | **5.76** |
| ON(I) | 5.78 | 7.86 | 6.24 | 5.51 | 5.18 | 6.40 |
| ON(J) | 5.63 | 7.57 | 5.91 | 5.38 | 4.99 | 6.16 |
| ON(K) | 6.36 | 7.95 | 6.26 | 5.67 | **5.72** | 6.50 |
| ON(L) | 5.31 | 7.47 | 5.87 | 5.14 | 4.57 | 6.09 |
| ON(M) | 4.80 | 7.21 | 5.65 | 4.75 | 4.18 | 5.90 |
| ON(N) | 5.69 | 7.71 | 6.01 | 5.27 | 4.90 | 6.08 |
| ON(O) | 5.72 | 7.80 | 6.17 | 5.48 | 5.14 | 6.25 |
| ON(P) | 5.91 | 7.57 | 5.68 | 4.88 | 5.36 | 6.21 |
| ON(Q) | **4.37** | **6.96** | **5.37** | **4.53** | **3.97** | **5.76** |
| ON(R) | 4.71 | 7.18 | 5.61 | 4.72 | 4.14 | 5.91 |
| ON(CONCAT) | 6.02 | 7.78 | 6.13 | 5.33 | 5.08 | 6.21 |
| AVG(ON(A)…ON(R)) | 5.61 | 7.61 | 5.97 | 5.26 | 4.97 | 6.20 |
| STDEV(ON(A)…ON(R)) | 0.66 | 0.32 | 0.30 | 0.41 | 0.59 | 0.25 |

Table 6.4. (continued)

| simulation trace:<br>sensible budget range: | M<br>25–28 ms | N<br>24.5–29.5 ms | O<br>24.5–29.5 ms | P<br>20.5–23 ms | Q<br>23.5–27.5 ms | R<br>24.5–28 ms |
|---|---|---|---|---|---|---|
| CLV | 8.70 | 8.15 | 8.17 | 7.61 | 7.89 | 8.48 |
| Q4 | −169.16 | −485.95 | −412.54 | −382.60 | −235.46 | −134.54 |
| OFF(A) | **6.11** | **6.18** | 6.11 | **1.42** | 4.19 | **5.21** |
| OFF(B) | 5.65 | 5.88 | 5.80 | **1.42** | 3.65 | 4.77 |
| OFF(C) | 5.61 | 5.89 | 5.79 | **1.42** | 3.65 | 4.72 |
| OFF(D) | 2.50 | 4.19 | 4.51 | **1.42** | 3.63 | 2.48 |
| OFF(E) | 0.43 | 1.82 | 2.36 | **1.42** | 2.65 | 0.74 |
| OFF(F) | −12.04 | −12.70 | −12.85 | 3.06 | −6.64 | −8.83 |
| OFF(G) | −0.32 | 1.06 | 0.69 | **1.42** | 2.52 | −0.04 |
| OFF(H) | −8.35 | −8.47 | −7.44 | 3.36 | −4.42 | −6.25 |
| OFF(I) | 5.67 | 5.95 | **6.19** | **1.42** | **4.75** | 4.94 |
| OFF(J) | 4.43 | 4.71 | 4.59 | **1.42** | 2.67 | 3.61 |
| OFF(K) | 3.94 | 5.55 | 5.49 | **1.42** | 3.65 | 3.76 |
| OFF(L) | 0.26 | 0.84 | 2.08 | 2.08 | 1.59 | 0.74 |
| OFF(M) | 0.62 | 1.26 | 2.45 | 2.40 | 1.75 | 0.84 |
| OFF(N) | −0.99 | 0.55 | 0.96 | 1.75 | 1.77 | −0.48 |
| OFF(O) | 0.70 | 1.88 | 2.40 | 1.75 | 2.16 | 0.70 |
| OFF(P) | **−136.97** | **−330.98** | **−284.46** | 2.63 | **−95.46** | **−100.70** |
| OFF(Q) | −4.33 | −4.87 | −3.67 | **3.61** | −2.90 | −3.59 |
| OFF(R) | −1.10 | −0.94 | 0.93 | 2.72 | 0.08 | −0.79 |
| OFF(CONCAT) | 5.46 | 5.69 | 5.99 | 1.42 | 4.59 | 4.78 |
| AVG(OFF(A)…OFF(R)) | −7.12 | −17.34 | −14.34 | 2.01 | −3.93 | −4.90 |
| STDEV(OFF(A)…OFF(R)) | 31.85 | 76.23 | 65.70 | 0.74 | 22.40 | 23.54 |
| ENH(A) | **7.14** | 6.24 | 6.52 | **4.82** | **5.72** | **6.67** |
| ENH(B) | 6.94 | 6.30 | 6.60 | 4.97 | 5.64 | 6.44 |
| ENH(C) | 6.92 | 6.35 | 6.57 | 5.00 | 5.53 | 6.41 |
| ENH(D) | **5.80** | 6.16 | 6.56 | 4.95 | **4.31** | **5.28** |
| ENH(E) | 6.44 | 6.25 | 6.57 | 4.96 | 4.84 | 5.81 |
| ENH(F) | 6.16 | 6.22 | **6.48** | 4.93 | 4.65 | 5.64 |
| ENH(G) | 6.20 | **6.14** | 6.64 | 5.06 | 4.75 | 5.60 |
| ENH(H) | 6.44 | 6.27 | 6.52 | 5.00 | 4.93 | 5.91 |
| ENH(I) | 6.83 | 6.24 | 6.54 | 4.97 | 5.17 | 6.20 |
| ENH(J) | 6.78 | 6.27 | **6.69** | **5.07** | 5.36 | 6.17 |
| ENH(K) | 6.35 | 6.23 | 6.65 | 5.06 | 5.01 | 5.80 |
| ENH(L) | 6.75 | 6.28 | 6.54 | 4.97 | 5.25 | 6.17 |
| ENH(M) | 6.81 | 6.23 | 6.59 | 5.00 | 5.17 | 6.17 |
| ENH(N) | 6.50 | **6.40** | 6.61 | 5.05 | 5.11 | 5.88 |
| ENH(O) | 6.17 | 6.22 | 6.59 | 5.05 | 4.48 | 5.52 |
| ENH(P) | 6.68 | **6.14** | 6.58 | 4.95 | 5.03 | 6.05 |
| ENH(Q) | 6.87 | 6.28 | 6.59 | 4.97 | 5.38 | 6.29 |
| ENH(R) | 6.80 | 6.21 | 6.52 | 4.94 | 5.18 | 6.16 |
| ENH(CONCAT) | 6.80 | 6.31 | 6.63 | 5.07 | 5.29 | 6.19 |
| AVG(ENH(A)…ENH(R)) | 6.59 | 6.25 | 6.57 | 4.98 | 5.08 | 6.01 |
| STDEV(ENH(A)…ENH(R)) | 0.34 | 0.07 | 0.05 | 0.06 | 0.37 | 0.35 |
| ON(A) | 6.87 | **6.28** | **6.34** | **4.47** | 5.60 | 6.46 |
| ON(B) | 6.85 | 6.15 | 6.25 | 3.96 | **5.68** | 6.49 |
| ON(C) | 6.85 | 6.17 | 6.25 | 3.99 | 5.67 | 6.49 |
| ON(D) | 6.86 | 6.09 | 5.97 | 3.79 | 5.62 | 6.44 |
| ON(E) | 6.73 | 5.89 | 5.82 | 3.42 | 5.53 | 6.35 |
| ON(F) | 6.56 | 4.84 | 5.08 | 2.78 | 5.27 | 6.22 |
| ON(G) | **6.91** | **6.28** | 6.31 | 4.33 | 5.60 | 6.48 |
| ON(H) | 6.21 | 4.54 | 4.73 | 2.14 | 5.00 | 5.79 |
| ON(I) | 6.90 | 6.07 | 5.95 | 3.75 | 5.58 | 6.44 |
| ON(J) | 6.71 | 5.35 | 5.67 | 3.20 | 5.49 | 6.37 |
| ON(K) | 6.87 | 6.14 | 6.31 | 4.07 | **5.68** | **6.51** |
| ON(L) | 6.66 | 5.02 | 5.25 | 3.24 | 5.36 | 6.36 |
| ON(M) | 6.48 | 4.64 | 4.88 | 2.66 | 5.19 | 6.17 |
| ON(N) | 6.59 | 5.83 | 5.71 | 3.12 | 5.52 | 6.24 |
| ON(O) | 6.75 | 6.01 | 5.93 | 3.52 | 5.54 | 6.38 |
| ON(P) | 6.28 | 5.68 | 5.75 | 2.79 | 5.55 | 5.85 |
| ON(Q) | **6.09** | **4.43** | **4.63** | **2.02** | **4.92** | **5.65** |
| ON(R) | 6.42 | 4.58 | 4.79 | 2.54 | 5.16 | 6.14 |
| ON(CONCAT) | 6.70 | 5.95 | 5.85 | 3.35 | 5.54 | 6.33 |
| AVG(ON(A)…ON(R)) | 6.64 | 5.56 | 5.65 | 3.32 | 5.44 | 6.27 |
| STDEV(ON(A)…ON(R)) | 0.25 | 0.67 | 0.58 | 0.71 | 0.23 | 0.25 |

In Table 6.4 we also present the results for strategies OFF(CONCAT), ENH(CONCAT), and ON(CONCAT) applied in simulations on traces A to R. For each strategy we observe that statistics trace CONCAT performs well on all 18 simulation traces. With only one minor exception, for strategy OFF(CONCAT) applied in simulations on trace P, statistics trace CONCAT consistently results in a higher average revenue over the specified budget range than the average taken over the results for statistics traces A to R individually. Based on this result we conclude that CONCAT is an appropriate statistics trace to derive parameter settings for the different control strategies.

### 6.6.5 Gain-time creation

In Chapter 2 we briefly mentioned gain time [Audsley *et al.*, 1994], which is the difference between the budget assigned to a task and the budget used by the task. The gain time that is generated by a task is made available to other tasks. The amount of gain time that a task will receive from other tasks in a given time period cannot be guaranteed, and is only know afterwards. For this reason, we want a task to generate as little gain time as possible.

In this section we study the creation of gain time by the SVA for various control strategies. Figure 6.9 shows a graph of the average budget usage per period, as a function of the budget assigned to the SVA, for strategy ON(CONCAT) applied in simulations on trace CONCAT. Up to a budget of 20 ms the SVA produces hardly any gain time, i.e., the SVA almost fully consumes its budget. For increasing values of the budget, the average budget usage converges to 26.7 ms, which is the average processing time per frame for trace CONCAT at quality level $q_4$.

Next, Table 6.5 shows the average budget usage per period to attain different values of average revenue, for strategies Q4, OFF(CONCAT), ENH(CONCAT), and ON(CONCAT) applied in simulations on trace CONCAT. The values in Table 6.5 were computed using linear interpolation between the simulation results. Unfortunately, we have no data for the clairvoyant strategy CLV. When comparing these results with the results in Table 6.1, we observe overall a significant difference between the budget that has to be assigned to the SVA to attain a certain average revenue, and the average budget usage. As mentioned, this difference is the gain time, and we observe that it is smaller for strategies ENH(CONCAT) and ON(CONCAT) than for strategies Q4 and OFF(CONCAT). By optimizing the QoS measure, as we try to do with our control strategies, a significant reduction of gain time can be achieved. For example, to attain an average revenue of 8, using strategy ON(CONCAT) the SVA requires a budget of 29.2 ms, while its average budget usage is only 25.5 ms. This results in an average gain time of 3.7 ms, approximately 13% of the budget. For comparison, using strategy Q4, which does not try to optimize the QoS measure, the gain time is approximately 32% of the budget. Hence,

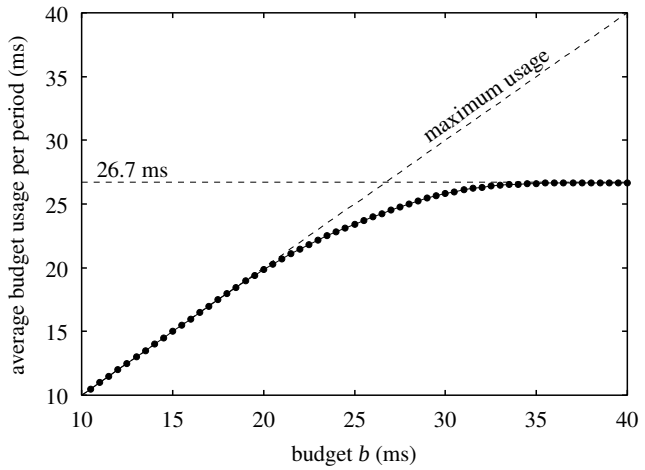


Figure 6.9. The average budget usage per period, as a function of the budget assigned to the SVA, for strategy ON(CONCAT) applied in simulations on trace CONCAT.

Table 6.5. The average budget usage per period to attain different values of the average revenue, for strategies Q4, OFF(CONCAT), ENH(CONCAT), and ON(CONCAT) applied in simulations on trace CONCAT.

| average revenue | Q4 | OFF(CONCAT) | ENH(CONCAT) | ON(CONCAT) |
|---|---|---|---|---|
| 0.0 | 26.6 ms | 26.0 ms | 25.3 ms | 24.5 ms |
| 1.0 | 26.6 ms | 26.1 ms | 25.4 ms | 24.5 ms |
| 2.0 | 26.6 ms | 26.1 ms | 25.5 ms | 24.6 ms |
| 3.0 | 26.6 ms | 26.2 ms | 25.5 ms | 24.7 ms |
| 4.0 | 26.6 ms | 26.3 ms | 25.6 ms | 24.8 ms |
| 5.0 | 26.6 ms | 26.3 ms | 25.7 ms | 24.9 ms |
| 6.0 | 26.7 ms | 26.4 ms | 25.8 ms | 25.1 ms |
| 7.0 | 26.7 ms | 26.5 ms | 25.9 ms | 25.3 ms |
| 8.0 | 26.7 ms | 26.6 ms | 26.0 ms | 25.5 ms |
| 9.0 | 26.7 ms | 26.7 ms | 26.3 ms | 25.9 ms |
| 9.9 | 26.7 ms | 26.7 ms | 26.6 ms | 26.5 ms |

QoS control can be an effective approach to reduce gain time. The remaining gain time can be put to good use by applying gain time reclamation.

# 7

# Conclusion

In this final chapter we recapitulate the work presented in this thesis. First, in Section 7.1 we discuss experiments that we set up to assess the various control strategies based on user perception. Unfortunately, these experiments were not very successful. Next, in Section 7.2 we come back to a number of simplifying assumptions that were made in the processing model. Finally, in Section 7.3 we summarize the main results of this thesis, and we present directions for future research.

## 7.1 User-perception experiments

So far, in this thesis we have assessed the various control strategies only by means of simulation experiments. Of course, the ultimate goal is to assess the strategies in a real system, based on user perception. Unfortunately, the research platform that we used to derive the traces could not be applied, because it was still under development. To still be able to do some user-perception experiments, we followed a different approach to assess the strategies. The idea of this approach is that, for a given SVA, we first generate a number of processing-time traces for various video sequences. We use these traces to run simulation experiments, similar to the ones described in Chapters 4, 5, and 6. However, for each simulation we now use the SVA to generate the true video output that corresponds to the imaginary video

output of the simulation. The generated output sequence can be shown afterwards in real-time on a display, and used for perception experiments.

A video algorithm that was eligible for this approach was natural motion. The natural motion algorithm doubles the frame rate of a video sequence by inserting a new frame between each pair of successive input frames. The new frame is computed based on the two surrounding input frames. This computation consists of a motion estimation step, in which the positions of objects in the new frame are predicted, and a motion compensation step, in which the new frame is generated. Motion estimation is usually implemented by dividing each input frame into smaller blocks of, for example, $8 \times 8$ or $16 \times 16$ pixels. For each block a motion vector is computed, which indicates where the visual content of the block has most probably moved to in the next input frame. A well-known motion estimation algorithm is 3-D Recursive Search (3DRS) [De Haan *et al.*, 1993]. For each block this algorithm considers a particular set of candidate motion vectors. For each candidate motion vector a match error called sum of absolute differences (SAD) is computed. The candidate vector that results in the smallest SAD is applied for motion compensation. The number of SAD computations is a proper measure for the amount of processing time needed by the motion estimation algorithm.

In the natural motion algorithm that we used the 3DRS motion estimation algorithm was already made scalable. This was done by introducing a threshold on the SAD for accepting a candidate motion vector. For a given block, if a candidate motion vector results in an acceptable SAD, then the motion vector is applied for motion compensation, and the SAD computation for the other candidate motion vectors is skipped. A set of quality levels can be defined by introducing multiple threshold values, one for each quality level.

To define quality levels for the scalable algorithm, we considered a set of 53 different threshold values. For each threshold value we ran the natural motion algorithm seven times, on seven input sequences of 20 frames each. In each run we estimated the quality of the output signal using a Philips-proprietary tool. Figure 7.1 shows multiple graphs of the estimated output quality as a function of the resource usage (for the motion estimation part only, expressed in the average number of SAD computations per block), for one of the input sequences. There is a graph for each inserted frame. Each graph connects the measurements for 53 different threshold values. The graphs for the other six input sequences are comparable to the graphs of Figure 7.1. In the figure, the two lowest graphs are due to initialization of algorithm, and should be ignored. We have drawn a black curve in the figure, which is an intuitive approximation for the different graphs. Using the curve, we defined four quality levels, $q_1$ to $q_4$, as indicated in the figure. We also defined an additional quality level, $q_0$, which generates the inserted frame by copying the preceding input frame. This quality level, which requires no active processing of

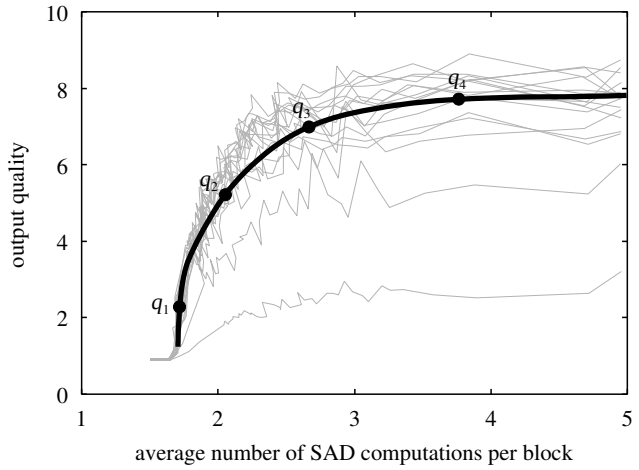


Figure 7.1. Multiple graphs of the output quality of a natural motion algorithm as a function of the resource usage of its scalable motion estimation algorithm, for a given input sequence of 20 frames. The different graphs correspond to the frames that were inserted by natural motion. Each graph connects the measurements for 53 different threshold values on the SAD. The black curve gives an intuitive approximation for the different graphs.

the natural motion algorithm, provides the scenario for missing a deadline.

As a next step towards testing the various control strategies in user-perception experiments, we set up an initial user-perception experiment to find proper rewards for the different quality levels. In this experiment we used three video sequences of 100 frames each as input for the natural motion algorithm. We selected these sequences based on clearly visible motion artifacts, which can be reduced by natural motion. We processed each sequence at each of the five different quality levels. This resulted in 15 processed sequences of 199 frames each. Next, we created a set up in which two displays were placed next to each other. Using this setup, we defined a number of paired comparison tests. In each paired comparison test the two displays show the same video sequence simultaneously, but processed at different quality levels. The test subject (a person) has to indicate which of the two sequences he or she prefers, the left one or the right one. The sequences are shown repeatedly, until the subject has made a choice. For each of the three video sequences we defined a set of 20 paired comparison tests, for all combinations of a particular quality level chosen for the left display and another quality level chosen for the right display. This resulted in a set of 60 paired comparison tests.

We performed the experiment on a group of 20 test subjects. For each test sub-

ject we placed the 60 paired comparison tests in random order. When performing a statistical analysis on the obtained test data for the three sequences together, we came to the following, disappointing conclusion. Although quality level $q_4$ was perceived significantly better than $q_3$, the differences in user-perceived quality between $q_1$ and $q_2$ were not significant. Furthermore, the fall-back quality level $q_0$, to be used in case of deadline misses, was not even significantly different from $q_1$ and $q_2$, whereas one normally expects a very poor quality. Furthermore, when performing a statistical analysis on the test data for the three sequences individually, we observed a clear dependence of the applied video sequence on the user perception of the different quality levels. For example, for one sequence the difference between $q_3$ and $q_4$ was not significant, and for another sequence the difference between $q_0$ and $q_3$ was not significant. Based on these results, we could not find a solid basis to proceed with the experiments. Before continuing on this track, either the MPEG-2 decoding algorithm used in the previous chapters should be made more operational, or another algorithm with better perceivable differences should be used.

In addition to the above, we observe that the curve in Figure 7.1 appears to be logarithmic. For an SVA, a more linear relation between the output quality and the resource needs is however preferred, to prevent that different quality levels provide roughly the same user-perceived quality, or that they require roughly the same amount of resources. In other words, there should be something to gain by varying the quality level. In [Radha *et al.*, 2001] a scalable MPEG-4 video encoding algorithm is presented having this desirable property.

## 7.2 Some assumptions revisited

In Chapter 2 we made a number of simplifying assumptions in the processing model. We now come back to these assumptions, and sketch possible solutions for the situation that the assumptions do not hold.

**The input order and output order of frames.** In the processing model we assume that the input order and output order of frames are the same. This assumption is in general valid for all video algorithms, except for the class of MPEG algorithms. To give an impression of the problems that arise, we focus on MPEG-2 decoding. As mentioned in Section 1.1.1, for MPEG-2 the decoding order (i.e., input order) of frames can differ from the display order (i.e., output order) of the frames. For example, for a sequence of frames $I_1P_2B_3B_4I_5B_6B_7$ in decoding order, the display order is given by $I_1B_3B_4P_2B_6B_7I_5$, where $I_1$ denotes that frame 1 is an I-frame, $P_2$ denotes that frame 2 is a P-frame, etcetera. For simplicity, we assume that all frame numbers are in decoding order. A single quality-level change in decoding order can result in multiple quality-level changes in display order. In

the above example, if frames 1 and 2 are processed (decoded) at $q_1$, and frames 3 to 7 are processed at $q_4$, then the single quality-level change between frames 2 and 3 in decoding order maps onto three quality-level changes in display order, viz. between frames 1 and 3, between frames 4 and 2, and between frames 2 and 6. On the other hand, multiple quality-level changes in decoding order can also result in fewer quality-level changes in display order. If frame 1 is processed at $q_1$, frame 2 is processed at $q_4$, frames 3 and 4 are processed at $q_1$, and frames 5 to 7 are processed at $q_4$, then the three quality-level changes in decoding order map onto a single quality-level change in display order, viz between frames 4 and 2. Another point of concern is that the picture quality of a processed frame does not only depend on the quality level at which the frame is processed, but also on the quality levels at which the used reference frames were processed. For example, if a P-frame is processed at $q_3$ using a reference I-frame that was processed at $q_1$, then the picture quality of the processed P-frame is typically lower than if the I-frame was also processed at $q_3$.

We consider two approaches to deal with the above issues. The first approach is to only allow quality-level changes for I-frames, or only for I-frames and P-frames. In other words, the controller is restricted to taking decisions less frequently. In the MDP model this can be implemented by restricting the set of quality levels that can be chosen in a given state. The second approach that we consider is to extend the definition of a state in the MDP model with two extra components: the quality level chosen for the last-processed I- or P-frame, and the quality level chosen for the one-but-last processed I- or P-frame. Using this additional information, in the MDP model it is possible to actively control the quality-level changes in display order, and to prevent that a frame $f$ is processed using a reference frame that has been processed at a quality level lower than $q(f)$.

Another issue of different input and output orders concerns the deadlines of frames. In our model, the deadline of a frame is given by the time at which the processed frame is consumed from the output queue, and we assume that the deadlines of successively processed frames are strictly periodic in time. However, if the decoding order and display order of frames can differ, then the deadlines of successively processed frames may no longer be strictly periodic in time. One approach to deal with this issue is to revisit the definition of a deadline. The deadline of a frame may still be mapped onto the time at which the processed frame is consumed from the output queue. However, the deadline may also represent the time at which the processed frame is needed as a reference frame. For both types of deadlines we can define different deadline miss penalties. The second approach we consider is to revisit the computation of progress in the processing model, to make the progress of a frame dependent on the frame type. So, depending on the frame type, there are different probabilities of missing a deadline.

A final remark is that it may be useful to apply a combination of the skipping approach and the aborting approach to handle deadline misses. Upon a deadline miss, for an I-frame or a P-frame it may be best to apply the skipping approach, as the processed frame may be needed as a reference frame. For a B-frame it may be best to apply the aborting approach, to not reduce the budget available for processing a later, more important, I- or P-frame.

**The input rate and output rate of frames.** In the processing model we assume that the input rate and output rate of frames are the same. This assumption is not valid for the class of scan rate up-conversion and down-conversion algorithms, which increase or decrease the frame rate of a video sequence, respectively. For example, the natural motion algorithm, described in Section 7.1, doubles the frame rate of a video sequence. For each input frame the algorithm produces two output frames. One output frame is copied directly from the input, and requires no active processing. The other output frame is generated by means of motion-compensated interpolation between two successive input frames. Hence, we can still see this as a one-to-one relation between the input and the output of the algorithm. We can also address the issue by redefining input frames or output frames. For example, we can define two successive output frames as a single output frame.

**An integer periodic latency.** In the processing model we assume that the periodic latency $\delta$ is an integer number. This means that the output process tries to consume a frame from the output queue at the very same moment as the input process tries to insert a frame into the input queue. In general, the periodic latency does not have to be an integer number. The only consequence for our model is that the input queue and output queue should each consist of at least $\lceil \delta \rceil$ frame buffers. Hence, the available buffers are used less efficiently. Nevertheless, it may be useful to choose $\delta$ somewhat higher than strictly necessary, to absorb fluctuations in the arrival times and consumption times of frames.

## 7.3 Conclusions

In this final section we summarize the main results of this thesis, and we identify directions for future research. First, in Chapter 2 we presented a processing model for a software video processing task with soft real-time constraints. In this model the task has to process an indefinite sequence of video frames. The task is characterized by highly fluctuating, content-dependent processing times of frames. For each frame to be processed there is a deadline, which is given by the time at which the processed frame is needed for output. We assumed that the deadlines of the successive frames to be processed are strictly periodic in time. In each time period between two successive deadlines a fixed amount of guaranteed processing time is

assigned to the task, called a budget. We assumed that this budget is smaller than the task's worst-case needs for processing video frames. Hence, if no countermeasures are taken, then the deadlines of the most compute-intensive frames will be missed. A deadline miss generally results in a perceivable artifact in the output of the task. We discussed two approaches to recover from a deadline miss, an aborting approach and a skipping approach. Furthermore, we introduced a measure for frames called progress. The progress of a frame indicates how much budget is left for processing the frame, before the frame's deadline is missed.

To prevent deadline misses we applied a combination of two techniques. First, we applied scalable video processing. In particular, we considered a video processing task consisting of a scalable video algorithm (SVA) and a corresponding controller. An SVA can process frames at different quality levels. Each quality level provides a particular trade-off between the time needed for processing a frame, and the resulting picture quality. The controller is used to select the quality level for each frame. By selecting the proper quality levels for frames, deadline misses may be prevented. The second technique that we applied is asynchronous processing. Asynchronous processing allows the task to work ahead, which can be used to even out the fluctuating load of frames in time. The extent to which working ahead can be applied is determined by latency and buffer constraints.

We used the notion of Quality of Service (QoS) to trade-off three aspects of user-perceived output quality in a single measure: the quality levels at which frames are processed, deadline misses, and quality-level changes between successively processed frames. We defined the revenue of a processed frame as a measure indicating to what extent these three aspects are satisfied for the frame. Next, we defined the QoS measure as the average revenue per frame, over all frames that are processed. Given the QoS measure, we defined the QoS control problem as the problem of finding a quality-level selection strategy for the controller that will maximize the QoS measure for an arbitrary sequence of frames to be processed. This problem is an on-line problem, because the controller has to select the quality level for each frame without knowing how complex the frame is, or how complex the frames are that follow.

In Chapter 4 we modeled the QoS control problem as a finite Markov decision process (MDP). In this model, the controller has to select the quality level for each frame based on the state of the SVA. We defined this state by the combination of the progress of the frame, the type of the frame, and the previous quality level. To obtain a finite set of states we discretized the progress using intervals. The state-transition probabilities and expected revenues for the MDP model are estimated based on pre-determined processing-time statistics. The MDP model is solved before run time for one particular value of the budget. The result is given by an optimal policy, a lookup table that indicates which quality level should be chosen

for a frame to be processed, given the state of the SVA. Following an optimal policy will maximize the average revenue, provided that the situation at run time matches the MDP model. We defined the off-line strategy as the strategy for the controller that selects the quality levels for frames using such a pre-computed policy.

In the MDP model, the processing times of successive frames are implicitly assumed to be independent. However, in practice there are often dependencies in the processing times of successive frames, due to the highly related video content of the frames. Therefore, in Chapter 5 we introduced a technique to enhance the off-line strategy, called budget scaling. The idea behind this technique is that we consider the load of the SVA to consist of a content-dependent structural load, around which short-term load fluctuations take place. We introduced a run-time measure for the structural load of the SVA, called the scaled budget. The MDP model is solved multiple times, for different values of the scaled budget. The state-transition probabilities and expected revenues in the MDP model are estimated based on processing-time statistics that only comprise short-term load fluctuations around the structural load. At run time, to select the quality level for a frame, we let the controller apply a policy that was computed for the actual value of the scaled budget. Hence, the controller always applies a policy that was computed to handle short-term load fluctuations around the actual structural load. We called this control strategy the enhanced strategy.

In Chapter 6 we presented an on-line solution approach to the QoS control problem. This approach is based on solving the MDP model at run time, using the Q-learning algorithm. The Q-learning algorithm can learn an optimal policy at run time, based on gained experience, while using the learned policy at the same time for control. The algorithm does not require pre-determined state-transition probabilities and expected revenues. This property allowed us to discretize continuous state components differently, based on linear interpolation instead of intervals. As a result, we could discretize the progress at a much lower granularity than before, while maintaining the same accuracy of the MDP model, and we could integrate the budget scaling technique directly into the MDP model. Next, we introduced a technique to remove the previous quality level component from the state signal, called state compression. The Q-learning algorithm makes use of a set of action values, which represent a policy. The controller starts with randomly chosen action values. Before processing a frame, the controller first updates a number of action values, based on statistics of the just-processed frame. Next, the algorithm uses the set of action values to select the quality level for the frame. Normally, the Q-learning algorithm updates only a single action value per frame. Based on characteristics of the QoS control problem we were able to adapt the Q-learning algorithm. This adaptation allowed us to update all action values per frame, instead of just a single action value, which speeds up learning significantly. We called the

resulting control strategy the on-line strategy.

We assessed the various control strategies in simulation experiments. As input for these experiments we used the processing times for decoding 18 MPEG-2 sequences from DVD. Each sequence was processed at four different quality levels. For benchmarking purposes we introduced a clairvoyant control strategy, which always selects the quality levels for frames optimally based on off-line optimization. The performance of this clairvoyant strategy cannot be met by any run-time strategy. In initial experiments we first fine-tuned the various control strategies. From these experiments we learned that it does not pay off to use MPEG-2 frame types in the MDP model. For the on-line strategy, the run-time overhead of the controller is mainly determined by the number of states in the MDP model. To obtain an acceptable run-time overhead we defined a relatively small set of states for this strategy.

As expected, we observed that the off-line strategy cannot properly deal with dependencies in the processing times of successive frames. In contrast, we observed that the enhanced strategy and the on-line strategy are well able to deal with load dependencies. Both strategies performed roughly equally well, and, in contrast to the off-line strategy, they also performed relatively close to optimum. To attain the same average revenue as the clairvoyant strategy, the two strategies required only a little more budget. Based on this result we conclude that budget scaling is a successful technique to handle load dependencies. Moreover, we conclude that the on-line strategy learns fast, and that it can perform remarkably well using a small set of states.

We also observed that the enhanced strategy and the on-line strategy are quite insensitive to the statistics that are used to derive the strategies. This is a favorable result, as in practice the strategies are applied to unknown video content. We esteem the on-line strategy to be the best strategy. In contrast to the enhanced strategy, it barely needs pre-determined processing-time statistics, which makes it more suitable to be applied to new sequences. Finally, we conclude that reinforcement learning can provide a powerful contribution to QoS control in adaptive real-time systems.

There are a number of directions for future work. First of all, the various control strategies still have to be tested in a real system based on user perception. As described in Section 7.1, our attempt to test the strategies in a realistic setting was not successful, because users often could not see the differences between quality levels. Second, the enhanced strategy and the on-line strategy use expected processing times for the different quality levels, to compute the scaled budgets of completed frames. These expected processing times are computed before run time, and at run time they are not adapted to the actual processing times. It would be interesting to adapt these expected processing times at run time, and to study the

effect on the performance of both control strategies. Third, so far we assumed that a fixed periodic budget is assigned to the task. In practice, the budget may be subject to fluctuations. This is for example the case when the SVA has to share its budget with another task that has a data-dependent load. Using a stochastic budget is also a topic of future research.

# Bibliography


AARTS, E., R. HARWIG, AND M. SCHUURMANS [2002], Ambient intelligence, in: P.J. Denning (ed.), *The Invisible Future: The Seamless Integration of Technology into Everyday Life*, McGraw-Hill, New York, NY, 235–250.

AUDSLEY, N.C., A. BURNS, R.I. DAVIS, K.W. TINDELL, AND A.J. WELLINGS [1995], Fixed priority pre-emptive scheduling: An historical perspective, *Journal of Real-Time Systems* **8**, 173–198.

AUDSLEY, N.C., R.I. DAVIS, AND A. BURNS [1994], Mechanisms for enhancing the flexibility and utility of hard real-time systems, *Proc. 15th IEEE Real-Time Systems Symposium (RTSS)*, San Juan, Puerto Rico, 12–21.

BAICEANU, V., C. COWAN, D. MCNAMEE, C. PU, AND J. WALPOLE [1996], Multimedia applications require adaptive CPU scheduling, *Proc. Workshop on Resource Allocation Problems in Multimedia Systems*, Washington, DC.

BELLMAN, R. [1957], *Dynamic Programming*, Princeton University Press, Princeton, NJ.

BRASPENNING, R.A., G. DE HAAN, AND C. HENTSCHEL [2002], Complexity scalable motion estimation, *Proc. SPIE Conference on Visual Communications and Image Processing (VCIP)*, Vol. 4671, San José, CA, 442–453.

BRIL, R.J. [2004], *Real-time scheduling for media processing using conditionally guaranteed budgets*, Ph.D. thesis, Technische Universiteit Eindhoven, Eindhoven, The Netherlands.

BRIL, R.J., M. GABRANI, C. HENTSCHEL, G.C. VAN LOO, AND E.F.M. STEFFENS [2001a], QoS for consumer terminals and its support for product families, *Proc. International Conference on Media Futures (ICMF)*, Florence, Italy, 299–302.

BRIL, R.J., C. HENTSCHEL, E.F.M. STEFFENS, M. GABRANI, G. VAN LOO, AND J.H.A. GELISSEN [2001b], Multimedia QoS in consumer terminals, *Proc. IEEE Workshop on Signal Processing Systems (SIPS)*, Antwerp, Belgium, 332–343.

BUTTAZZO, G.C. [1997], *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

CHEN, W.Y. [1997], Emerging home digital networking needs, *Proc. 4th Interna-*

*tional Workshop on Community Networking*, Atlanta, GA, 7–12.

COMBAZ, J., J.-C. FERNANDEZ, T. LEPLEY, AND J. SIFAKIS [2005a], Fine grain QoS control for multimedia application software, *Proc. Conference and Exhibition on Design, Automation and Test in Europe (DATE'05)*, Vol. 2, Munich, Germany, 1038–1043.

COMBAZ, J., J.-C. FERNANDEZ, T. LEPLEY, AND J. SIFAKIS [2005b], QoS control for optimality and safety, *Proc. 5th ACM International Conference on Embedded Software (EMSOFT'05)*, Jersey City, NJ, 90–99.

FENG, X., AND A.K. MOK [2002], A model of hierarchical real-time virtual resources, *Proc. 23rd IEEE Real-Time Systems Symposium (RTSS)*, Austin, TX, 26–35.

HAAN, G. DE [2000], *Video Processing for Multimedia Systems*, University Press Eindhoven, Eindhoven, The Netherlands.

HAAN, G. DE, P.W.A.C. BIEZEN, H. HUIJGEN, AND O.A. OJO [1993], True-motion estimation with 3-D recursive search block matching, *IEEE Transactions on Circuits and Systems for Video Technology* **3**, 368–379.

HAMANN, C.-J., J. LÖSER, L. REUTHER, S. SCHÖNBERG, J. WOLTER, AND H. HÄRTIG [2001], Quality-assuring scheduling: Using stochastic behavior to improve resource utilization, *Proc. 22nd IEEE Real-Time Systems Symposium (RTSS)*, London, UK, 119–128.

HASKELL, B.G., A. PURI, AND A.N. NETRAVALI [1997], *Digital Video: An Introduction to MPEG-2*, Chapman & Hall, New York, NY.

HENTSCHEL, C., R. BRASPENNING, AND M. GABRANI [2001a], Scalable algorithms for media processing, *Proc. International Conference on Image Processing (ICIP)*, Vol. 3, Thessaloniki, Greece, 342–345.

HENTSCHEL, C., R.J. BRIL, Y. CHEN, R. BRASPENNING, AND T.-H. LAN [2003], Video quality-of-service for consumer terminals: A novel system for programmable components, *IEEE Transactions on Consumer Electronics* **49**, 1367–1377.

HENTSCHEL, C., R.J. BRIL, M. GABRANI, L. STEFFENS, K. VAN ZON, AND S. VAN LOO [2001b], Scalable video algorithms and dynamic resource management for consumer terminals, *Proc. International Conference on Media Futures (ICMF)*, Florence, Italy, 193–196.

ISOVIĆ, D., AND G. FOHLER [2004], Quality aware MPEG-2 stream adaptation in resource constrained systems, *Proc. 16th Euromicro Conference on Real-Time Systems (ECRTS)*, Catania, Italy, 23–32.

ISOVIĆ, D., G. FOHLER, AND L. STEFFENS [2003], Timing constraints of MPEG-2 decoding for high-quality video: Misconceptions and realistic assumptions, *Proc. 15th Euromicro Conference on Real-Time Systems (ECRTS)*, Porto, Portugal, 73–82.

ITU-T [1994], *Recommendation E.800*, Technical report, International Telecommunication Union, http://www.itu.int/home/index.html.

JARNIKOV, D., P. VAN DER STOK, AND C.C. WÜST [2004], Predictive control of video quality under fluctuating bandwidth conditions, *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, Taipei, Taiwan, 1051–1054.

KLEIN, M.H., T. RALYA, B. POLLAK, R. OBENZA, AND M. GONZÁLEZ HARBOUR [1993], *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer Academic Publishers, Dordrecht, The Netherlands.

LAFRUIT, G., L. NACHTERGAELE, K. DENOLF, AND J. BORMANS [2000], 3D computational graceful degradation, *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, Vol. 3, Geneva, Switzerland, 547–550.

LAN, T., Y. CHEN, AND Z. ZHONG [2001], MPEG2 decoding complexity regulation for a media processor, *Proc. Fourth IEEE Workshop on Multimedia Signal Processing (MMSP)*, Cannes, France, 193–198.

LEE, C., J. LEHOCZKY, R. RAJKUMAR, AND D. SIEWIOREK [1999], On quality of service optimization with discrete QoS options, *Proc. Fifth IEEE Real-Time Technology and Applications Symposium (RTAS)*, Vancouver, Canada, 276–286.

LEHOCZKY, J.P., AND S.R. THUEL [1995], Scheduling periodic and aperiodic tasks using the slack stealing algorithm, in: S.H. Son (ed.), *Advances in Real-Time Systems*, Prentice Hall, Inc., Englewood Cliffs, NJ, 175–198.

LIU, C.L., AND J.W. LAYLAND [1973], Scheduling algorithms for multiprogramming in a real-time environment, *Journal of the ACM* **20**, 46–61.

MERCER, C.W., S. SAVAGE, AND H. TOKUDA [1994], Processor capacity reserves: Operating system support for multimedia applications, *Proc. International Conference on Multimedia Computing and Systems (ICMCS)*, Boston, MA, 90–99.

MIETENS, S., P.H.N. DE WITH, AND C. HENTSCHEL [2004], Computational-complexity scalable motion estimation for mobile MPEG encoding, *IEEE Transactions on Consumer Electronics* **50**, 281–291.

MITCHELL, J.L., W.B. PENNEBAKER, C.E. FOGG, AND D.J. LEGALL [1997], *MPEG Video Compression Standard*, Chapman & Hall, New York, NY.

OPPENHEIM, A.V., AND R.W. SCHAFER [1975], *Digital Signal Processing*, Prentice Hall, Inc., Englewood Cliffs, NJ.

OTERO PÉREZ, C.M., AND I. NITESCU [2002], Quality of service resource management for consumer terminals: Demonstrating the concepts, *Proc. Work-in-Progress Session 14th Euromicro Conference on Real-Time Systems (ECRTS)*, Vienna, Austria, 29–32.

OTERO PÉREZ, C.M., L. STEFFENS, P. VAN DER STOK, S. VAN LOO, A. ALONSO, J.F. RUÍZ, R.J. BRIL, AND M. GARCÍA VALLS [2003], QoS-based resource management for ambient intelligence, in: T. Basten, M. Geilen, and H. de Groot (eds.), *Ambient Intelligence: Impact on Embedded System Design*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 159–182.

PENG, S. [2001], Complexity scalable video decoding via IDCT data pruning, *Digest of Technical Papers IEEE International Conference on Consumer Electronics (ICCE)*, Los Angeles, CA, 74–75.

PRASAD, D., A. BURNS, AND M. ATKINS [2003], The valid use of utility in adaptive real-time systems, *Real-Time Systems* **25**, 277–296.

PUTERMAN, M.L. [1994], *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons Inc., New York, NY.

RADHA, H.M., M. VAN DER SCHAAR, AND Y. CHEN [2001], The MPEG-4 fine-grained scalable video coding method for multimedia streaming over IP, *IEEE Transactions on Multimedia* **3**, 53–68.

RAJKUMAR, R., K. JUVVA, A. MOLANO, AND S. OIKAWA [1998], Resource kernels: A resource-centric approach to real-time and multimedia systems, *Proc. SPIE Conference on Multimedia Computing and Networking (MMCN)*, Vol. 3310, San José, CA, 150–164.

RATHNAM, S., AND G. SLAVENBURG [1996], An architectural overview of the programmable multimedia processor, TM-1, *Proc. IEEE COMPCON*, Santa Clara, CA, 319–326.

SHA, L., J.P. LEHOCZKY, AND R. RAJKUMAR [1986], Solutions for some practical problems in prioritized preemptive scheduling, *Proc. 7th IEEE Real-Time Systems Symposium (RTSS)*, New Orleans, LA, 181–191.

STEINMETZ, R. [1996], Human perception of jitter and media synchronization, *IEEE Journal on Selected Areas in Communications* **14**, 61–72.

SUTTON, R.S. [1988], Learning to predict by the methods of temporal differences, *Machine Learning* **3**, 9–44.

SUTTON, R.S., AND A.G. BARTO [1998], *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA.

WAL, J. VAN DER [1981], *Stochastic Dynamic Programming: Successive Approximations and Nearly Optimal Strategies for Markov Decision Processes and Markov Games*, Mathematical Centre Tracts 139, Mathematisch Centrum, Amsterdam, The Netherlands.

WATKINS, C.J.C.H. [1989], *Learning from Delayed Rewards*, Ph.D. thesis, King's College, University of Cambridge, Cambridge, UK.

WHITE, D.J. [1993], *Markov Decision Processes*, John Wiley & Sons Ltd., Chich-

ester, UK.

WÜST, C.C., R.J. BRIL, C. HENTSCHEL, L. STEFFENS, AND W.F.J. VERHAEGH [2004a], QoS control challenges for multimedia consumer terminals, *Proc. First International Workshop on Probabilistic Analysis Techniques for Real-Time and Embedded Systems (PARTES)*, Pisa, Italy, http://www.cs.york.ac.uk/rts/partes04/final/08_wust.pdf.

WÜST, C.C., E.F.M. STEFFENS, AND W.F.J. VERHAEGH [2003], Adaptive QoS control for real-time video processing, *Proc. Work-in-Progress Session 15th Euromicro Conference on Real-Time Systems (ECRTS)*, Porto, Portugal, 49–52.

WÜST, C.C., L. STEFFENS, R.J. BRIL, AND W.F.J. VERHAEGH [2004b], QoS control strategies for high-quality video processing, *Proc. 16th Euromicro Conference on Real-Time Systems (ECRTS)*, Catania, Italy, 3–12.

WÜST, C.C., L. STEFFENS, W.F.J. VERHAEGH, R.J. BRIL, AND C. HENTSCHEL [2005], QoS control strategies for high-quality video processing, *Real-Time Systems* **30**, 7–29.

WÜST, C.C., AND W.F.J. VERHAEGH [2004c], Dynamic control of scalable media processing applications, in: W. Verhaegh, E. Aarts, and J. Korst (eds.), *Algorithms in Ambient Intelligence*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 259–276.

WÜST, C.C., AND W.F.J. VERHAEGH [2004d], Quality control for scalable media processing applications, *Journal of Scheduling* **7**, 105–117.

ZHAO, W., C. CHEW LIM, J.W.S. LIU, AND P.D. ALEXANDER [1995], Overload management by imprecise computation, in: S. Natarajan (ed.), *Imprecise and Approximate Computation*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1–22.

ZHONG, Z., Y. CHEN, AND T.-H. LAN [2002], Signal adaptive processing in MPEG-2 decoders with embedded resizing for interlaced video, *Proc. SPIE Conference on Visual Communications and Image Processing (VCIP)*, Vol. 4671, San José, CA, 434–441.

# Author index

# A

# Appendix: Simulation software

In this appendix we give a high-level overview of the software that was written for the simulation experiments of Sections 4.4, 5.4, and 6.6. The simulation experiments were carried out using a set of C++ programs. In Section A.1 we give a short description of these programs. Next, in Section A.2 we provide data flow graphs of simulation experiments that make use of strategy Q4, the off-line strategy, the enhanced strategy, and the on-line strategy.

## A.1 Programs

**GetAverageProcessingTimes**

*Description:* For a given input trace, this program computes the average processing time per frame for each quality level.

*Input:* a trace $\tau$.

*Output:* the average processing time per frame for each quality level.

**NormalizeTrace**

*Description:* This program uses an IIR filter to transform a source trace into a normalized trace, as described in Section 5.3.3.

*Input:* a trace $\tau$ and an IIR filter step-size parameter $\theta$.

*Output:* a normalized trace $\tau'$.

**ComputePolicies**

*Description:* This programs solves the MDP model of Section 4.1 using successive approximation, for a given statistics trace, and for each budget in a given set of budgets. To derive policies for the enhanced strategy, a normalized statistics trace should be used, and the budgets should correspond to scaled budgets.

*Input:* a statistics trace $\tau$, a set of budgets $b$, a periodic latency $\delta$, a number of progress intervals $n_\Lambda$, the applied deadline miss approach, a set of revenue parameters, and a successive approximation convergence error $\varepsilon$.

*Output:* a monotonic policy for each budget $b$, the computation time of each policy, and the expected average revenue of each policy.

**Simulator**

*Description:* This program simulates the execution of an SVA, according to the processing model of Chapter 2. The simulation is based on the processing times in a given simulation trace. The SVA uses a controller to determine the quality levels at which frames are processed. The controller applies strategy Q4, the off-line strategy, the enhanced strategy, or the on-line strategy.

*Input:* a simulation trace $\tau$, a simulation budget $b$, a periodic latency $\delta$, the applied deadline miss approach, a set of revenue parameters, and the applied control strategy.

*Additional input for the off-line strategy:* a monotonic policy computed for budget $b$.

*Additional input for the enhanced strategy:* a set of monotonic policies computed for different values of the scaled budget, an average processing time for each quality level, and an IIR filter step-size parameter $\theta$.

*Additional input for the on-line strategy:* a set of gridpoint states, a learning rate $\psi$, an average processing time for each quality level, and an IIR filter step-size parameter $\theta$.

*Output:* the number of processed frames, the number of frames processed at each quality level, the number of skipped frames, the number of aborted frames, the number of quality level changes for each pair of quality levels, the number of deadline misses, the average revenue, and the average budget usage per period.

**ClairvoyantStrategy**

*Description:* This program computes the maximum average revenue that can be attained in a simulation on a given simulation trace for a given simulation budget, based on dynamic programming.

*Input:* a simulation budget $b$, a periodic latency $\delta$, a number of progress intervals $n_\Lambda$, the applied deadline miss approach, and a set of revenue parameters.

*Output:* the number of processed frames, the number of frames processed at each quality level, the number of skipped frames, the number of aborted frames, the number of quality level changes for each pair of quality levels, the number of deadline misses, and the average revenue.

## A.2 Data flow graphs

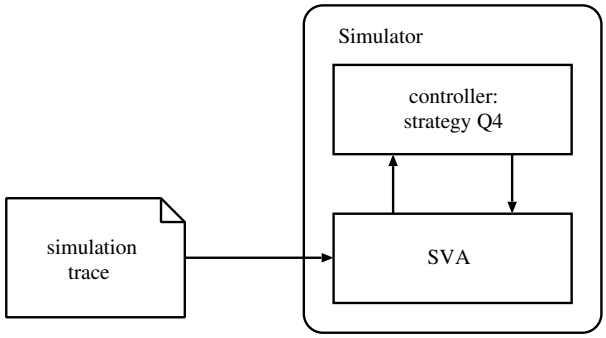


Figure A.1. Data flow graph of a simulation experiment using strategy Q4.

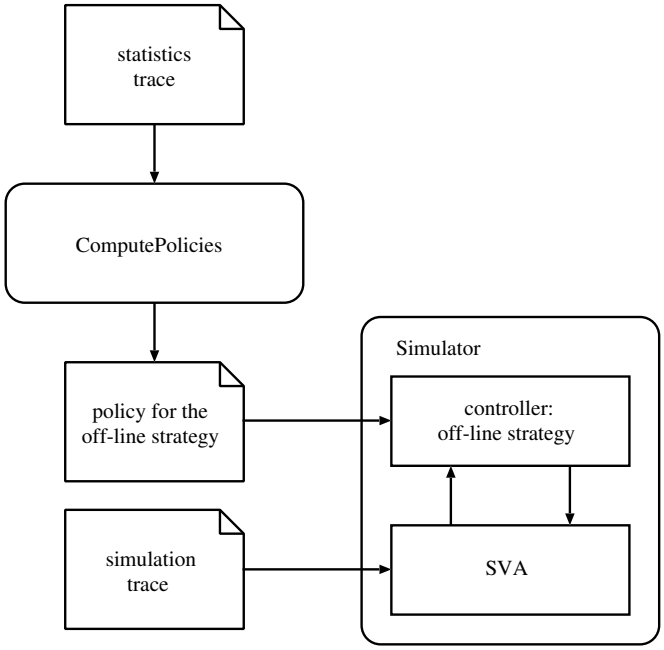


Figure A.2. Data flow graph of a simulation experiment using the off-line strategy.

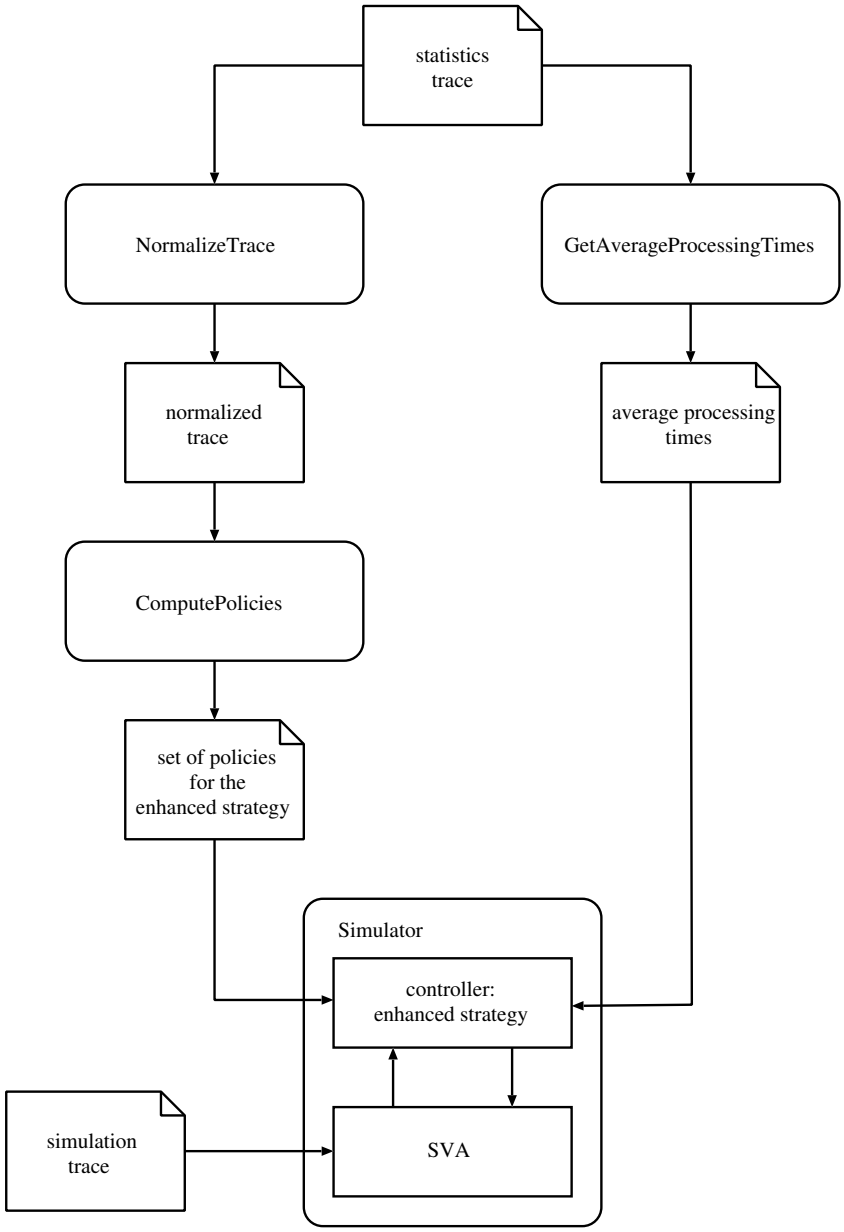


Figure A.3. Data flow graph of a simulation experiment using the enhanced strategy.

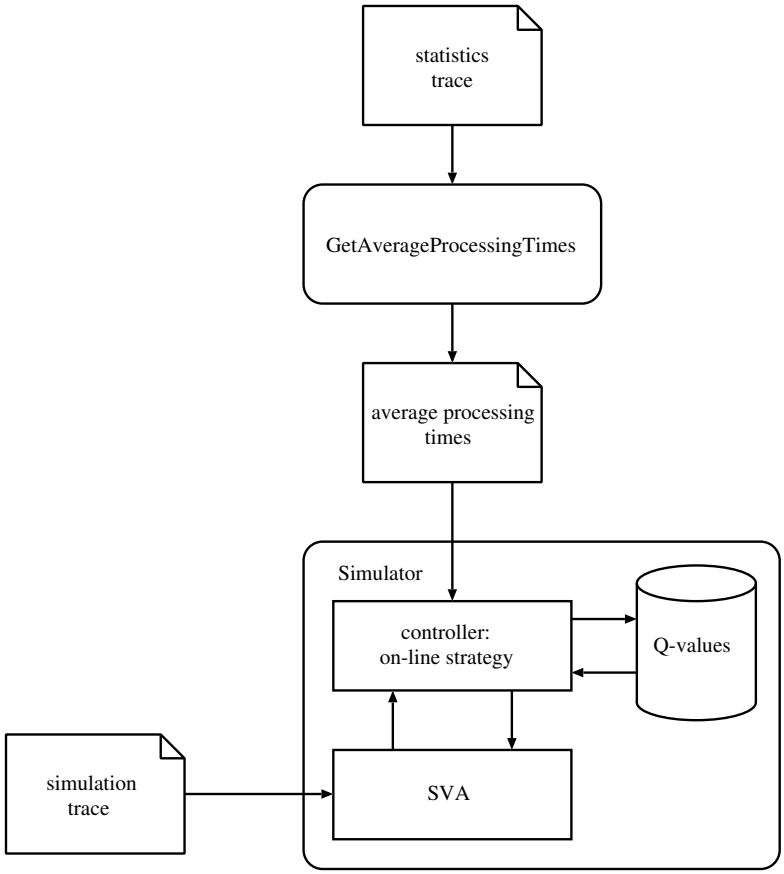


Figure A.4. Data flow graph of a simulation experiment using the on-line strategy.

# B

# Appendix: Resulting output

This appendix contains a list of papers and reports of which the author of this thesis is an author, and a list of patent applications of which the author of this thesis is an inventor. The listed papers, reports, and patent applications are all related to the work described in this thesis.

## B.1 Papers and reports

- WÜST, C., AND W. VERHAEGH [2001], Quality level control for scalable media processing applications having fixed CPU budgets, *Proc. 2nd Philips Workshop on Scheduling and Resource Management (SCHARM'01)*, Eindhoven, The Netherlands, 29–39.
- AARTS, E., E. DEN BOEF, J. KORST, V. PRONK, W. VERHAEGH, AND C. WÜST [2002], Adaptive scheduling and resource management in ambient intelligence, *PT Embedded Systems Research Dossier*, 12–15.
- WÜST, C.C., AND W.F.J. VERHAEGH [2002], Dynamic control of scalable media processing applications, *Proc. 1st Philips Symposium On Intelligent Algorithms (SOIA'02)*, Eindhoven, The Netherlands, 119–131.

- WÜST, C.C., E.F.M. STEFFENS, AND W.F.J. VERHAEGH [2003], Adaptive QoS control for real-time video processing, *Proc. Work-in-Progress Session 15th Euromicro Conference on Real-Time Systems (ECRTS)*, Porto, Portugal, 49–52.
- WÜST, C.C., AND R.H.M. WUBBEN [2003], *Resource Scalability of a Block-Hopping 3-D Recursive Search Motion Estimator*, Philips Research Technical Note PR-TN-2003/00792, Eindhoven, The Netherlands.
- WÜST, C.C., AND W.F.J. VERHAEGH [2004], Dynamic control of scalable media processing applications, in: W. Verhaegh, E. Aarts, and J. Korst (eds.), *Algorithms in Ambient Intelligence*, Kluwer Academic Publishers, Dordrecht, The Netherlands, 259–276.
- WÜST, C.C., AND W.F.J. VERHAEGH [2004], Quality control for scalable media processing applications, *Journal of Scheduling* **7**, 105–117.
- WÜST, C.C., L. STEFFENS, R.J. BRIL, AND W.F.J. VERHAEGH [2004], QoS control strategies for high-quality video processing[1], *Proc. 16th Euromicro Conference on Real-Time Systems (ECRTS)*, Catania, Italy, 3–12.
- WÜST, C.C., R.J. BRIL, C. HENTSCHEL, L. STEFFENS, AND W.F.J. VERHAEGH [2004], QoS control challenges for multimedia consumer terminals, *Proc. 1st International Workshop on Probabilistic Analysis Techniques for Real-Time and Embedded Systems (PARTES)*, Pisa, Italy.
- JARNIKOV, D., P. VAN DER STOK, AND C.C. WÜST [2004], Predictive control of video quality under fluctuating bandwidth conditions, *Proc. IEEE International Conference on Multimedia and Expo (ICME)*, Taipei, Taiwan, 1051–1054.
- WÜST, C.C., L. STEFFENS, W.F.J. VERHAEGH, R.J. BRIL, AND C. HENTSCHEL [2005], QoS control strategies for high-quality video processing, *Real-Time Systems* **30**, 7–29.
- WÜST, C.C., AND W.F.J. VERHAEGH [2005], Intelligent control for scalable video processing, *Proc. Workshop on Resource Management for Media Processing in Networked Embedded Systems (RM4NES)*, Eindhoven, The Netherlands, 51–69.
- BRIL, R.J., W.F.J. VERHAEGH, AND C.C. WÜST [2006], A cognac-glass algorithm for conditionally guaranteed budgets, *Accepted for publication in Proc. 27th IEEE Real-Time Systems Symposium (RTSS)*, Rio de Janeiro, Brazil.


[1] For this paper we received the best paper award of the ECRTS'04.

## B.2 Patent applications

- VERHAEGH, W.F.J., AND C.C. WÜST (inventors), *Method of and system to set a quality of a media frame*, Koninklijke Philips Electronics N.V. (applicant), Publ. nr. WO03051039, Publ. date June 19, 2003, Prio. nr. EP20010204791, Prio. date December 10, 2001.
- VERHAEGH, W.F.J., AND C.C. WÜST (inventors), *Method of and system to set an output quality of a media frame*, Koninklijke Philips Electronics N.V. (applicant), Publ. nr. WO2004095274, Publ. date November 4, 2004, Prio. nr. EP20030076189, Prio. date April 23, 2003.
- VAN DER STOK, P.D.V., C.C. WÜST, AND D. JARNIKOV (inventors), *Method and apparatus for smoothing overall quality of video transported over a wireless medium*, Koninklijke Philips Electronics N.V. (applicant), Publ. nr. WO2005048606, Publ. date May 26, 2005, Prio. nr. US20030519809P, Prio. date November 13, 2003.

# Intelligent Control for Scalable Video Processing

## Summary

In this thesis we study a problem related to cost-effective video processing in software by consumer electronics devices, such as digital TVs. Video processing is the task of transforming an input video signal into an output video signal, for example to improve the quality of the signal. This transformation is described by a video algorithm. At a high level, video processing can be seen as the task of processing a sequence of still pictures, called frames. Video processing in consumer electronic devices is subject to strict time constraints. In general, the successively processed frames are needed periodically in time. If a frame is not processed in time, then a quality reduction of the output signal may be perceived.

Video processing in software is often characterized by highly fluctuating, content-dependent processing times of frames. There is often a considerable gap between the worst-case and average-case processing times of frames. In general, assigning processing time to a software video processing task based on its worst-case needs is not cost effective. We consider a software video processing task to which has been assigned insufficient processing time to process the most compute-intensive frames in time. As a result, a severe quality reduction of the output signal may occur. To optimize the quality of the output signal, given the limited amount of processing time that is available to the task, we do the following. First we use a technique called asynchronous processing, which allows the task to make more effective use of the available processing time by working ahead. Second, we make use of scalable video algorithms. A scalable video algorithm can process frames at different quality levels. The higher the applied quality level for a frame, the higher is the resulting picture quality, but also the more processing time is needed. Due to the combination of asynchronous processing and scalable processing, a larger fraction of the frames can be processed in time, however at the cost of a sometimes lower picture quality.

The problem we consider is to select the quality level for each frame. The objective that we try to optimize reflects the user-perceived quality, and is given by a combination of the number of frames that are not processed in time, the quality levels applied for the processed frames, and changes in the applied quality level

between successive frames. The video signal to be processed is not known in advance, which means that we have to make a quality-level decision for each frame without knowing in which processing time this will result, and without knowing the complexity of the subsequent frames. As a first solution approach we modeled this problem as a Markov decision process. The input of the model is given by the budgetted processing time for the task, and statistics on the processing times of frames at the different quality levels. Solving the Markov decision process results in a Markov strategy that can be used to select a quality level for each frame to be processed, based on the amount of time that is available for processing until the deadline of the frame.

Our first solution approach works well if the processing times of successive frames are independent. In practice, however, the processing times of successive frames can be highly correlated, because successive frames are often very similar. Our second solution approach, which can be seen as an extension of our first approach, takes care of the dependencies in the processing times of successive frames. The idea is that we introduce a measure for the complexity of successively processed frames, based on structural fluctuations in the processing times of the frames. Before processing, we solve the Markov decision process several times, for different values of the complexity measure. During video processing we regularly determine the complexity measure for the frames that have just been processed, and based on this measure we dynamically adapt the Markov policy that is applied to select the quality level for the next frame.

The Markov strategies that we use are computed based on processing-time statistics of a particular collection of video sequences. Hence, these statistics can differ from the statistics of the video sequence that is processed. Therefore we also worked out a third solution approach in which we use a learning algorithm to select the quality levels for frames. The algorithm starts with hardly any processing-time statistics, but it has to learn these statistics from run-time experience. Basically, the learning algorithm implicitly solves the Markov decision process at run time, making use of the increasing amount of information that becomes available. The algorithm also takes care of dependencies in the processing time of successive frames, using the same complexity measure as in our second solution approach.

From computer simulations we learned that our second and third solution approaches perform close to a theoretical upper bound, determined by a reference strategy that selects the quality levels for frames based on complete knowledge of the processing times of all frames to be processed. Although our solutions are successful in computer simulations, they still have to be tested in a real system.

# Acknowledgments

During the work resulting in this thesis I was helped and supported by a great number of people to whom I feel a deep gratitude. First and foremost among them are copromotor Wim Verhaegh, Reinder Bril, and Liesbeth Steffens. Wim has been my daily supervisor for this work. His patience and encouragement helped to create a pleasant and stimulating working atmosphere. Words cannot express my gratitude for his outstanding intellectual support. Reinder and Liesbeth, driving people behind the QoS RM project, both provided many valuable contributions. Reinder's creativity was a great source of inspiration. He offered many useful comments on my thesis. Liesbeth helped me a lot in writing research papers. Thanks to her my work became visible to the outside world.

Many thanks are due to my first promotor, Emile Aarts, who gave me the opportunity to combine work at Philips with doctorate research. His inspiring – not to say ambient – advice guided me into the right directions. Together with Wim we had many fruitful discussions. I also would like to thank my second promotor, Peter de With, and the other members of the PhD committee, Onno Boxma, Gerhard Fohler – how lucky I was to meet him in Florida!, Johan Lukkien, and Gerard Sierksma, my former mathematics professor at the University of Groningen, who greatly stimulated my studies. They all provided valuable commentary that greatly improved the quality of the thesis.

The work described in this thesis was supported by and carried out at the Philips Research Laboratories in Eindhoven. It has been an honor to study in this leading research institute. Thanks go to the research management team, in particular to Maurice Groten, Eelco Dijkstra, and Jean Gelissen. The work reported upon in this thesis has been partially funded by the ITEA/EUROPA project.

Many colleagues helped me in practical ways with the work in progress. Among the others I name: Ralph Braspenning, Maria Gabrani, Christian Hentschel, Dmitri Jarnikov, Clara Otero Pérez, Laurenţiu Păpălău, Peter van der Stok, and Rob Wubben. Special thanks go to Roos Joordens and Franco Oberti, who helped me to set up and analyze the perception experiments. Furthermore, I am indebted in many ways to other friends and colleagues at Philips Research, in particular those from the Media Interaction group: Zharko Aleksovski, Edgar den Boef, Ramon Clout, Sebastian Egner, Kero van Gelder, Gijs Geleijnse,

Edwin Hanegraaf, Jan Korst, Wil Michiels, Steffen Pauws, Verus Pronk, and Dragan Sekulovski. Thank you all for a wonderful time! Frans Schraven did an excellent job in designing the cover of the thesis.

Closer to home, I would like to thank my dearest friends (and family), Frans and Netteke. They have supported my work in many ways. Back in the 1980s, Frans gave me my first computer, a Multitech Microprofessor MPF-III. Learning how to program this machine greatly raised my interest in mathematics and computer science. The final proof of my progress in this field is the present thesis.

I am grateful to all my friends who repeatedly reminded me that there is life beyond a thesis, but especially to Peter and Wilma, Bert and Sandra, Eric and Sylvia, Dolf, Edgar, Frank, Igor, and Robbert.

But most of all I am grateful to my parents, to my sister and her husband, who encouraged me, had patience with me, and offered me a warm and comforting home in all stages of my studies.

Eindhoven, November 2006 Clemens Wüst

# Curriculum Vitae

Clemens Christiaan Wüst was born on August 15, 1974, in Leeuwarden, the Netherlands. After he finished secondary school (atheneum) in 1992, at the Stedelijke Scholengemeenschap in Leeuwarden, he studied mathematics at the Noordelijke Hogeschool Leeuwarden. In 1996 he completed this study and received the BEd and BEng degrees. In parallel, in 1995 he started his study in mathematics at the University of Groningen, from which he received the MSc degree with honors in 1998. His Master's thesis, written under the supervision of prof.dr. G. Sierksma, deals with the adjacency of extreme points on the Hamiltonian cycle polytope.

In September 1998 Clemens joined the Philips Research Laboratories in Eindhoven. At Philips Research, he first participated in a two-years program on computer science. He next worked for several years in the QoS RM and V-QoS projects on the subject of Quality-of-Service resource management for high-quality video processing in software. The results of this research are reported in this thesis. In October 2006 he joined the research department of NXP Semiconductors, the former semiconductor division of Philips Electronics. His current activities are in the field of compiler technology for future generations of microarchitectures.

# Symbol index

The numbers refer to the pages of first occurence.

**Dummy variables**

| | |
|---|---|
| $f, g$ | frame numbers |
| $q, q'$ | quality levels |
| $\phi, \phi'$ | frame types |
| $t$ | time (continuous) or time step (discrete) |
| $s, s'$ | states |
| $\hat{s}$ | compressed state |
| $a, a'$ | actions |
| $\pi, \pi'$ | policies |
| $\lambda, \lambda'$ | progress intervals (Chapter 4) or progress values (Chapter 6) |
| $\tau, \tau'$ | traces |

**Chapter 2**

| | | |
|---|---|---|
| $Q$ | set of quality levels | 16 |
| $n_Q$ | number of quality levels | 16 |
| $q_i$ | quality level $i$ $(1 \leq i \leq n_Q)$ | 16 |
| $q(f)$ | quality level selected for frame $f$ | 16 |
| $\Phi$ | set of frame types | 16 |
| $n_\Phi$ | number of frame types | 16 |
| $\phi_i$ | frame type $i$ $(1 \leq i \leq n_\Phi)$ | 16 |
| $\phi(f)$ | type of frame $f$ | 16 |
| $e(f)$ | entry time of frame $f$ | 17 |
| $d(f)$ | deadline of frame $f$ | 17 |

| | | |
|---|---|---|
| $P$ | time period between successive deadlines | 16 |
| $\delta$ | periodic latency | 17 |
| $b$ | periodic budget | 23 |
| $b_t(f)$ | total amount of budget left at time $t$ until deadline $d(f)$ | 24 |
| $\alpha(f)$ | start point of frame $f$ | 17 |
| $\omega(f)$ | milestone of frame $f$ | 17 |
| $\mu(f)$ | processing time of frame $f$ | 18 |
| $\lambda_t(f)$ | progress of frame $f$ at time $t$ | 24 |
| $\lambda_\alpha(f)$ | progress of frame $f$ at its start point | 24 |
| $\lambda_\omega(f)$ | progress of frame $f$ at its milestone | 24 |
| $ndm(f)$ | number of deadline misses for frame $f$ | 25 |
| $prev(f)$ | the frame that was processed right before frame $f$ | 26 |
| $r(f)$ | revenue of frame $f$ | 26 |
| $R_{\text{ql}}(q)$ | reward for processing a frame at quality level $q$ | 26 |
| $P_{\text{dm}}$ | deadline miss penalty | 26 |
| $P_{\text{qlc}}(q, q')$ | penalty for changing the quality level from $q$ to $q'$ | 26 |

**Chapter 3**

| | | |
|---|---|---|
| $S$ | set of states | 29 |
| $s_t$ | state at time step $t$ | 29 |
| $A$ | set of actions | 29 |
| $A(s)$ | set of actions that can be taken in state $s$ | 30 |
| $a_t$ | action at time step $t$ | 30 |
| $r_t$ | revenue at time step $t$ | 30 |
| $R_t$ | return at time step $t$ | 30 |
| $p^a_{ss'}$ | probability on a transition from state $s$ to state $s'$ under action $a$ | 31 |
| $r^a_{ss'}$ | expected revenue of a transition from state $s$ to state $s'$ under action $a$ | 31 |
| $V^\pi$ | state-value function of policy $\pi$ | 31 |
| $V^\pi(s)$ | value of state $s$ under policy $\pi$ | 31 |

| | | |
|---|---|---|
| $V^*$ | optimal state-value function | 32 |
| $Q^\pi$ | action-value function of policy $\pi$ | 32 |
| $Q^\pi(s, a)$ | value of taking action $a$ in state $s$ under policy $\pi$ | 32 |
| $Q^*$ | optimal action-value function | 32 |
| $\pi$ | policy | 31 |
| $\pi(s,a)$ | probability of taking action $a$ in state $s$ under policy $\pi$ | 31 |
| $\pi(s)$ | action to be taken in state $s$ under policy $\pi$ | 31 |
| $\pi^*$ | optimal policy | 32 |
| $\gamma$ | discount rate | 30 |
| $\varepsilon$ | convergence error | 34 |
| $\psi$ | learning rate | 36 |

**Chapter 4**

| | | |
|---|---|---|
| $\Lambda$ | set of progress intervals | 40 |
| $n_\Lambda$ | number of progress intervals | 40 |
| $\lambda_i$ | progress interval $i$ $(1 \leq i \leq n_\Lambda)$ | 40 |
| $\underline{\lambda}$ | lower bound of progress interval $\lambda$ | 40 |
| $\overline{\lambda}$ | upper bound of progress interval $\lambda$ | 40 |
| $(\lambda, \phi, q)$ | state given by progress interval $\lambda$, frame type $\phi$, and previous quality level $q$ | 40 |
| $\lambda_s$ | progress interval of state $s$ | 40 |
| $\phi_s$ | frame type of state $s$ | 40 |
| $q_s$ | previous quality level of state $s$ | 40 |
| $p^q_{ss'}$ | probability on a transition from state $s$ to state $s'$ under quality level $q$ | 41 |
| $r^q_{ss'}$ | expected revenue of a transition from state $s$ to state $s'$ under quality level $q$ | 45 |
| $r^q_s$ | expected revenue for choosing quality level $q$ in state $s$ | 46 |
| $\Pr(\phi, \phi')$ | probability that a frame of type $\phi'$ follows upon a frame of type $\phi$ | 41 |
| $X_{\phi q}$ | random variable representing the processing time of a frame of type $\phi$ at quality level $q$ | 41 |

| | | |
|---|---|---|
| $F_{\phi q}$ | cumulative distribution function of $X_{\phi q}$ | 41 |
| OFF($\tau$) | off-line strategy based on the statistics of trace $\tau$ | 54 |

**Chapter 5**

| | | |
|---|---|---|
| $c(f)$ | complexity factor of frame $f$ | 71 |
| $c'(f)$ | running complexity factor of frame $f$ | 71 |
| $\mu(f, q)$ | processing time of frame $f$ at quality level $q$ | 77 |
| $c(f, q)$ | complexity factor of frame $f$ at quality level $q$ | 73 |
| $c'(f, q)$ | running complexity factor of frame $f$ at quality level $q$ | 73 |
| $\upsilon(f)$ | scaled budget of frame $f$ | 73 |
| $\bar{\mu}(q)$ | expected processing time of a frame at quality level $q$ | 71 |
| $\theta$ | step-size parameter | 71 |
| Q4 | strategy that only selects quality level $q_4$ | 80 |
| ENH($\tau$) | enhanced strategy based on the statistics of trace $\tau$ | 80 |
| CLV | clairvoyant strategy | 80 |

**Chapter 6**

| | | |
|---|---|---|
| $\Lambda$ | set of progress gridpoints | 93 |
| $n_\Lambda$ | number of progress gridpoints | 93 |
| $\lambda_i$ | progress gridpoint $i$ $(1 \leq i \leq n_\Lambda)$ | 93 |
| $\Upsilon$ | set of scaled budget gridpoints | 93 |
| $n_\Upsilon$ | number of scaled budget gridpoints | 93 |
| $\upsilon_i$ | scaled budget gridpoint $i$ $(1 \leq i \leq n_\Upsilon)$ | 93 |
| $S_G$ | set of gridpoint states | 92 |
| $\hat{S}$ | set of compressed states | 96 |
| $\hat{S}_G$ | set of compressed gridpoint states | 96 |
| $(\lambda, q, \upsilon)$ | state given by progress $\lambda$, previous quality level $q$, and scaled budget $\upsilon$ | 93 |
| $(\lambda, \upsilon)$ | compressed state given by progress $\lambda$ and scaled budget $\upsilon$ | 96 |
| $\lambda_s$ | progress of state $s$ (Chapter 6) | 93 |

| | | |
|---|---|---|
| $\upsilon_s$ | scaled budget of state $s$ | 93 |
| $s(f)$ | state at the start point of frame $f$ | 93 |
| $\hat{Q}(\hat{s}, q)$ | action value of compressed state $\hat{s}$ and quality level $q$ | 96 |
| $\mathrm{ON}(\tau)$ | on-line strategy based on the statistics of trace $\tau$ | 100 |

# List of acronyms

The numbers refer to the pages of first occurence.

| | | |
|---|---|---|
| 3DRS | 3-D Recursive Search | 116 |
| CGB | Conditionally Guaranteed Budget | 13 |
| DVD | Digital Versatile Disc | 2 |
| FIR | Finite Impulse Response (filter) | 71 |
| fps | frames per second | 2 |
| GOP | Group Of Pictures | 4 |
| IDCT | Inverse Discrete Cosine Transform | 51 |
| IIR | Infinite Impulse Response (filter) | 71 |
| MCT | Multimedia Consumer Terminal | 4 |
| MDP | Markov Decision Process | 31 |
| MPEG | Motion Picture Experts Group | 3 |
| PAL | Phase Alternating Line (video standard) | 2 |
| QoS | Quality of Service | 8 |
| RCE | Resource Consuming Entity | 9 |
| SAD | Sum of Absolute Differences | 116 |
| SVA | Scalable Video Algorithm | 6 |

# Subject index