

# Supporting Timeliness and Accuracy in Distributed Real-time Content-based Video Analysis

Viktor S. Wold Eide<sup>1,2</sup>, Frank Eliassen<sup>2</sup>, Ole-Christoffer Granmo<sup>1,2,3</sup>, and Olav Lysne<sup>2\*</sup>

<sup>1</sup>University of Oslo  
P.O. Box 1080 Blindern  
N-0314 Oslo, Norway

<sup>2</sup>Simula Research Laboratory  
P.O. Box 134  
N-1325 Lysaker, Norway

<sup>3</sup>Agder University College  
Grooseveien 36  
N-4876 Grimstad, Norway

viktore,olegr@ifi.uio.no

viktore,frank,olavly@simula.no

ole.granmo@hia.no

## ABSTRACT

Real-time content-based access to live video data requires content analysis applications that are able to process the video data at least as fast as the video data is made available to the application and with an acceptable error rate. Statements as this express quality of service (QoS) requirements to the application. In order to provide some level of control of the QoS provided, the video content analysis application must be scalable and resource aware so that requirements of timeliness and accuracy can be met by allocating additional processing resources.

In this paper we present a general architecture of video content analysis applications including a model for specifying requirements of timeliness and accuracy. The salient features of the architecture include its combination of probabilistic knowledge-based media content analysis with QoS and distributed resource management to handle QoS requirements, and its independent scalability at multiple logical levels of distribution. We also present experimental results with an algorithm for QoS-aware selection of configurations of feature extractor and classification algorithms that can be used to balance requirements of timeliness and accuracy against available processing resources. Experiments with an implementation of a real-time motion vector based object-tracking application, demonstrate the scalability of the architecture.

## Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems—*Distributed applications*; D.2.11 [Software Engineering]: Software Architectures—*Domain-specific architectures*; I.2.10 [Artificial Intelligence]: Vision and Scene Understanding—*Video analysis*

\*Authors are listed alphabetically

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

MM'03, November 2–8, 2003, Berkeley, California, USA.  
Copyright 2003 ACM 1-58113-722-2/03/0011 ...\$5.00.

## General Terms

Algorithms, design, measurement, performance

## Keywords

Real-time video content analysis, parallel processing, task graph scheduling, event-based communication, QoS and resource management

## 1. INTRODUCTION

There is evidence that the need for applications that can analyse concurrently and in real-time the content of multiple media streams, such as audio and video, is increasing. For example, in video content analysis applications including feedback control or interaction such as road traffic control systems, automated surveillance systems, and smart rooms, timeliness is an important aspect [1,6,29]. This requires that content analysis applications are able to process the media streams at least as fast as the data is made available to the application.

Real-time content analysis is an active research field where efficient techniques for e.g. multi-object detection and tracking have been found. In such applications, pattern classification systems which automatically classify media content in terms of high-level concepts have been taken into use. Roughly stated, the goal of such pattern classification systems is to bridge the gap between the low-level features produced through signal processing (filtering and feature extraction) and the high-level concepts desired by the end user.

The above challenges become even more critical when coordinated content analysis of video data from multiple video sources is necessary. Typically, a parallel processing environment is required for real-time performance.

When building a real-time content analysis application, not only must the processing properties of the application be considered, but also the content analysis properties. Such properties we might refer to as Quality of Service (QoS) dimensions and include dimensions such as timeliness and acceptable error rate.

Statements of QoS must generally be expressed by the application user according to some domain specific QoS model that defines the QoS dimensions for the specific application domain. In order to provide some level of control of the QoS provided, the video content analysis application must be scalable so that requirements of timeliness can be met

by allocating additional processing resources. Furthermore, the selection of analysis algorithms must be resource aware, balancing requirements of accuracy against processing time.

In this paper we present a general architecture of video content analysis applications that includes a model for specifying requirements of timeliness and accuracy. The architecture combines probabilistic knowledge-based media content analysis with resource awareness to handle QoS requirements. Based on an application for real-time tracking of a moving object in a video stream, we demonstrate that the architecture is independently scalable at multiple logical levels of distribution by measuring the performance of the application under different distribution configurations. Furthermore, we present experimental results with an algorithm for resource-aware selection of configurations of feature extractor and classification algorithms. In our prototype implementation the algorithm is used to balance requirements of timeliness and accuracy against available processing resources for the same application as above. Although we in this paper focus on video as input, the architecture itself is not limited to video only. It has been designed to handle and exploit input from any combination of different types of sensors in the same application.

Several frameworks for parallel execution of video content analysis tasks have been developed. For instance, in [25] a multi-agent based system for coarse-grained parallelization and distribution of feature extraction is presented. A fine-grained solution for parallel feature extraction is described in [20] where feature extraction is distributed in a hypercube multicomputer network. Other frameworks for parallel and distributed video analysis include [12, 21, 22, 34]. However, these frameworks lack an explicit QoS-model and do not support balancing of accuracy and timeliness against the available processing resources.

The rest of this paper is structured as follows. First, in Section 2 we present a general architecture for (video) content analysis applications. Then we present a QoS model for such applications in Section 3. In Section 4 we discuss architectural requirements that are important for supporting the QoS model. Based on these requirements, the architecture is presented in Section 5. In Section 6 we present empirical results. Lastly, in Section 7 we conclude and provide some pointers to further work.

## 2. CONTENT ANALYSIS

A general approach for building content analysis applications is to combine low-level quantitative video processing into high-level concept recognition. Typically, such applications are logically organized as a hierarchy of logical modules each encapsulating a video processing task, as illustrated in Figure 1. A task is a well-defined algorithm involved in the video analysis process.

We define task categories according to their functionality in the system. At the lowest level of the hierarchy there are tasks representing video streaming sources. At the level above, the video streams are filtered and transformed by filtering tasks. The transformed video streams are then fed to feature extraction tasks as video segments (e.g. video frame regions). Feature extraction tasks operate on the video segments from the transformed video streams, and in the case of a video frame region, calculate features such as color histograms and motion vectors. Finally, results from feature extraction tasks are reported to classification tasks higher

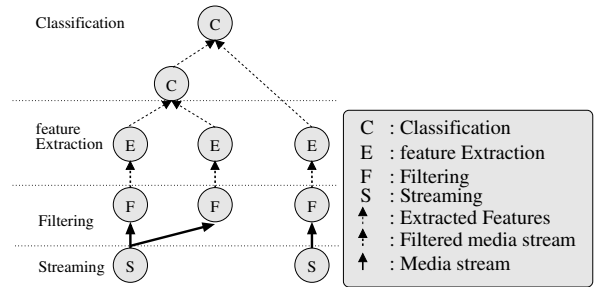


Figure 1: Content analysis hierarchy example.

up in the hierarchy that are responsible for detecting high level domain concepts, such as a moving object in a video stream. In other words, classification is interpretation of extracted features in some application specific context. We will hereafter denote the Streaming, Filtering, feature Extraction, and Classification by the letters S, F, E, and C respectively as seen in Figure 1.

Tasks generally form a directed acyclic graph where the tasks are represented by the nodes in the graph, and the edges represent the directed flows of data between tasks.

Often, the above type of content analysis applications are implemented as monolithic applications making reuse, development, maintenance, and extensions by third parties difficult. Such applications are often executed in single processes, unable to benefit from distributed processing environments.

## 3. QOS MODEL

In this section we present a QoS model for real-time content based video analysis applications. The model consists of QoS dimensions that we believe characterize the timeliness and accuracy requirements of such applications. Other QoS dimensions such as reliability and availability are considered outside the scope of this work.

The QoS model we have adopted in this work includes the following QoS dimensions: *accuracy*, *temporal resolution*, and *latency*.

The *accuracy* of a media content analysis application can be characterized by its estimated error rate, defined as the number of misclassifications divided by the total number of classifications when analysing a set of media streams. Depending on the media content analysis application, various levels of error rate may be acceptable. For instance, misclassifying events when monitoring an airport for security reasons may be more critical than misclassifying events when indexing a baseball video stream.

The *temporal resolution* dimension specifies the minimum temporal length of an event that the content analysis application should be able to detect. According to the Nyquist sampling theorem [31], any function of time (e.g. stream of high-level concepts) whose highest frequency is  $W$  can be completely determined by sampling at twice the frequency,  $2W$ . In other words, if a stream of high-level concepts is sampled at a frequency less than twice the frequency of the finest temporal details in the stream, high-level concepts may be missed. Hence the value of the temporal resolu-

tion dimension determines the required sampling frequency of the media streams to be analysed.

The *latency* dimension specifies the maximum acceptable elapsed time from an event occur in the real world until it is reported by the appropriate classifier algorithm. For real-time video content analysis applications including feedback control (e.g. road traffic control systems) this dimension is important.

In general, different classes of quality of service can also be identified, varying from best effort service to guaranteed service. The latter class requires support from the system in terms of resource reservation and admission control, while the former does not. Although the problem of resource reservation and admission control have been studied for a long time, their solution has not generally been integrated into more general-purpose operating systems and networks. We therefore restrict the class of processing platforms that we consider to general-purpose ones without special real-time processing features. However, we do assume that we have some level of control over the load of competing applications in the processing environment. Furthermore, we believe our results can easily be adapted to take advantage of processing platforms providing real-time scheduling policies.

## 4. ARCHITECTURAL REQUIREMENTS

It seems evident that the resource requirements for the application domain of real-time video content analysis are very challenging and will most likely remain so in the near future. This calls for an architecture that is scalable in the sense that the performance of the application scales well with the amount of processing resources allocated to it. Scalability is required in order to be able to cope with increasing QoS requirements and coordinated analysis of an increasing number of media streams.

A scalable application architecture can generally only be obtained by adopting distribution as its basic principle. Scalability of distributed applications is usually achieved by parallelizing application algorithms and distributing the processing of their parts to different processors.

The relative complexity of streaming, filtering/ transformation, feature extraction, and classification depends on the application. Therefore the architecture should support focusing of processing resources on any given logical level, independently of other logical levels. E.g., if only the filtering is parallelized and distributed, the feature extraction and the classification may become processing bottlenecks.

A scalable interaction mechanism which also supports such independent parallelization is therefore required.

The level of accuracy that can be supported by a video content analysis application depends on the misclassification behavior (error rate) of the selected configuration of feature extractor and classifier algorithms. Hence configurations of such algorithms must be carefully selected based on the desired level of accuracy.

However, selecting configurations of algorithms which give a high accuracy might result in increased processing time since configurations of algorithms with better accuracy usually require more processing cycles than configurations with poorer accuracy. Therefore, algorithms that can be used to decide whether a QoS requirement can be satisfied in a given distributed physical processing environment are needed. This will include search for an appropriate configuration of feature extractor and classifier algorithms that provides the

desired accuracy and that can be allocated to different processors in such a way that the requirements for latency and temporal resolution are fulfilled.

Reduced latency and a smaller temporal resolution may be achieved in a scalable distributed architecture by allocating independent tasks to different processors. A further decrease in temporal resolution may be achieved by deploying *dependent tasks* as pipe-lines also on different processors. E.g., in a maximally distributed video processing pipe-line a frame rate of  $R$  may be sustained if no task in the pipe-line has an average processing time per frame exceeding  $1/R$ .

## 5. ARCHITECTURE

In this section we develop an architecture that supports the three following content analysis application construction steps:

1. Decomposition of a content analysis application into tasks. The tasks form a task graph as discussed in Section 2. The granularity of the decomposition should be a modifiable parameter because what granularity is appropriate depends on the processing environment at hand, and in particular the number of processors available.
2. Fine grained trading of accuracy against latency (and consequently temporal resolution). The starting point is a “brute force” task graph. By a “brute force” task graph we shall mean a task graph that contains the filtering, feature extraction, and classification tasks deemed relevant, without regard of the processing resources available. Tasks are removed iteratively from the task graph so that either the latency/temporal resolution requirement is met (success) or the accuracy falls below the required level (failure).
3. Scalable deployment and execution of the resulting task graph on multiple processors in a distributed processing environment.

In the next subsections we will first describe some example task graphs. These will be used to exemplify the functionality of the different parts of our architecture. Then, we present the ARCAMIDE algorithm which is used in step two above. Finally, we introduce the architectural basis for applying the ARCAMIDE algorithm. This includes the support of step one and three above. With respect to step three, an event-based interaction mechanism is a key factor for achieving flexible parallelization and distribution.

### 5.1 Example Task Graphs

Figure 2 illustrates the functional decomposition of a content analysis application for real-time tracking of a moving object in a video stream, the application henceforth used for illustration purposes. The video stream is filtered by an algorithm doing video stream decoding and color-to-grey level filtering. The filtered video frame is divided into  $m \times n$  blocks (video segments) before a motion estimator calculates motion vectors for the blocks. The block motion vectors are then received by a classification task (a so-called particle filter) and used for object detection and tracking.

We base our example application on motion vector calculation and particle filtering, because these techniques are recent and promising approaches to object/region tracking

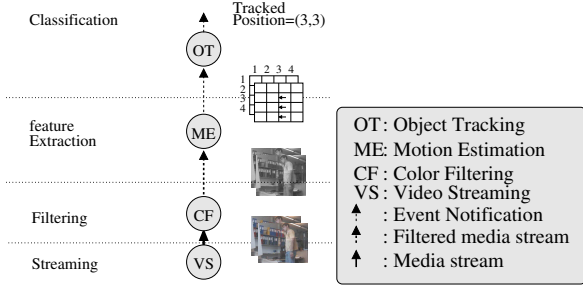


Figure 2: The functional decomposition of the real-time object tracking application.

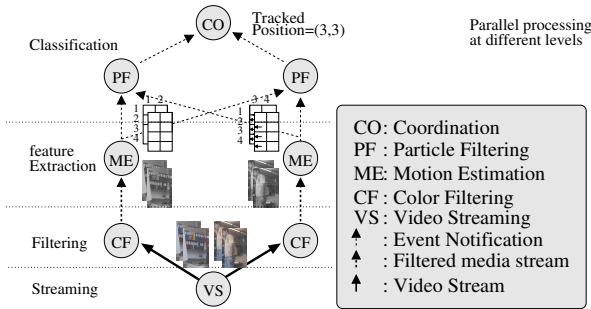


Figure 3: A configuration of the real-time object tracking application where the computation at several levels is parallelized.

in video. To elaborate, calculation of motion vectors (also called optical flow) is a typical pre-processing step in tracking of moving objects/regions [2, 27], and the particle filter is a promising approach to object-tracking which allows e.g. simultaneous tracking and verification [18].

The above content analysis task graph can be executed as a pipeline (each level of the chain is executed in parallel). For instance, the application can be executed on four processors, where the streaming is conducted from one processor, the filtering is executed on a second processor, the motion estimation is conducted on a third processor, and the classification on a fourth processor. Such distribution allows an application to take advantage of a number of processors equal to the depth of the hierarchy.

Figure 3 illustrates a task graph with a finer granularity compared to the task graph in Figure 2. The finer granularity has been achieved by independently decomposing the filtering, feature extraction and classification into pairs of two tasks. Such decomposition opens up for focusing the processing resources on the processing bottlenecks at hand. For instance, in Figure 3 the motion estimation could be conducted on two processors while the classification, i.e. particle filtering and coordination (see Section 5.3.5), can be conducted on three processors.

## 5.2 Determining Accuracy and Latency

In this subsection we describe the ARCAMIDE algorithm [14] for fine grained trading of accuracy against latency (and

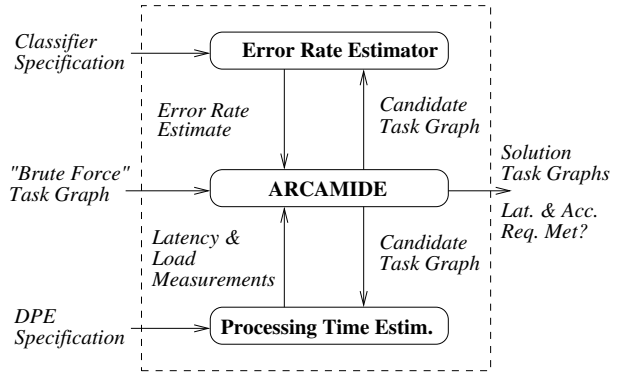


Figure 4: Architecture supporting the accuracy and latency dimension.

hence temporal resolution). Before we go into the details of the ARCAMIDE algorithm, let us provide an overview of the overall ARCAMIDE architecture. As shown in Figure 4, the architecture includes the following functionality:

- An *error rate estimator* which is used to estimate the content analysis (classification) error rate of candidate task graphs, based on a specification of the classification goal (e.g. object tracking). As we will see later, we use so-called dynamic Bayesian networks as a classifier specification language.
- A *processing time estimator* which is used to measure the latency and parallelizability of candidate task graphs, given a specification of the distributed processing environment (DPE) at hand.
- The *ARCAMIDE algorithm* which systematically removes tasks from the task graph in order to reduce the latency of the task graph while at the same time trying to minimize the loss of accuracy.

The output of the architecture is a sequence of solution task graphs ordered so that the estimated latency is decreasing while the estimated error rate is increasing. In this sequence, the first task graph which meets the latency requirement, must also meet the accuracy requirement. Otherwise, the specified DPE will not be able to support the given latency and accuracy requirements. In the following we will first discuss the inputs of the architecture, and then the estimator- and ARCAMIDE functionality in more detail.

### 5.2.1 The Task Graph and the DPE Specification

We assume that the task graph is annotated with the estimated processing time of each task in milliseconds on the available classes of processors. Furthermore, each edge is assumed to be annotated with the size in bytes of the data communicated between the respective tasks. When this is done for the available streaming, filtering, feature extraction, and classification tasks, we get the “brute force” task graph which the ARCAMIDE algorithm takes as input.

The DPE specification consists of the number and class of processors as well as the network latency and bandwidth between each pair of processors. To simplify the estimation of

the latency of a task graph, we make the following assumptions about the communication between each pair of processors: the communication is contention free and the network latency and bandwidth are constant. These assumptions are introduced to avoid the additional complexity caused by communication contention, routing, etc. (which are not the focus of this paper) while still handling a significant class of DPEs (e.g. dedicated homogeneous computers connected in a dedicated switched LAN).

### 5.2.2 Dynamic Bayesian Networks and Error Rate Estimation

Dynamic Bayesian networks (DBNs) [16] represent a particularly flexible class of pattern classifiers that allows statistical inference and learning to be combined with domain knowledge. Indeed, DBNs have been successfully applied to a wide range of video content analysis problems [5, 13, 26]. The successful application of DBNs can be explained by their firm foundation in probability theory, combined with the effective techniques for inference and learning that have been developed.

In order to be able to automatically associate high-level concepts (e.g. object position) to video segments, a DBN can be trained on manually annotated video streams. Generally stated, the training is based on finding a more or less accurate mapping between feature space and high-level concept space, within a hypothesis space of possible mappings.

After training the DBN on manually annotated video streams (the training set), the DBN can be evaluated by estimating the number of misclassifications on another manually annotated video stream not used for training (the test set). We shall by the *estimated error rate* of a DBN mean the number of misclassifications divided by the total number of classifications on the test set. This estimated error rate can be seen as a measure on how accurately the DBN will index novel video streams.

One important reason for basing the classification on DBNs is the fact that DBNs can classify even when features are missing. In contrast, other types of classifiers, like neural networks and decision trees, must be retrained whenever the feature set is changed. Accordingly, by using DBNs we make the exploration of the space of possible task graphs and thereby feature subsets more efficient.

### 5.2.3 The Processing Time Estimator

In this section we describe a simple scheduling algorithm targeting the class of task graphs and DPEs defined in Section 5.2.1. The algorithm is based on generic task graph scheduling principles, as described in [17], adopted to suit the needs of the ARCAMIDE algorithm. That is, we propose measures of latency and parallelizability. These measures are used to guide the ARCAMIDE algorithm.

The goal of the scheduling algorithm is to minimize the estimated latency of the task graph when allocating and scheduling the tasks to the available processors in the DPE. Based on the allocation and scheduling of tasks, the resulting latency and temporal resolution can be estimated. Furthermore, the parallelizability of the task graph can be measured. The respective procedures are summarized in Figure 5 and described below.

We examine the scheduling algorithm first. Let *entry tasks* be tasks without parents and let *exit tasks* be tasks without children in the task graph. We define the *b-level* of a task

---

```

; The scheduling algorithm
schedule(Tasks)
    ; Initially all tasks are unallocated
    A := ∅;
    N := Tasks;

    WHILE #N > 0 DO
        ; Identifies task with largest b-level
        n_max := ARGMAX n IN N: b_level(n);
        ; Identifies the processor which allows
        ; the earliest start time
        p_min := ARGMIN p IN P: start_time(n_max, p);
        ; Allocates n_max to p_min
        allocate(n_max, p_min);
        A := A ∪ {n_max};
        N := N \ {n_max};
        ; Returns the ready time of each processor
        RETURN {ready_time(p1), ..., ready_time(pP)};

; The latency estimation procedure
le(Tasks)
    RETURN MAX(schedule(Tasks));

; The parallelizability measurement procedure
par(Tasks)
    RETURN STD_DEV(schedule(Tasks));

```

---

**Figure 5: The scheduling, latency, and parallelizability procedures.**

to be the length of the longest path from the task to an exit node. Likewise, the *s-level* of a task is the length of the longest path from the task to an entry node. The *length of a path* is simply the sum of the task processing times (as given by the task graph) on the path. If multiple classes of processors are available, the average processing time of a task over the available classes of processors is used. Note that when calculating the b-level or the s-level of a task the task itself is included in the path.

A task is either *allocated* to a processor ( $A$ ) or *not allocated* to a processor ( $N$ ). Initially, none of the tasks are allocated to processors. At each iteration of the scheduling algorithm, the non-allocated task with the largest b-level is allocated to a processor. This means that execution of long task graph paths are prioritized before execution of short task graph paths. The main reason behind this strategy is that the longest task graph paths often determine the latency of the task graphs when multiple processors are available, and accordingly should be executed as early as possible.

When a task is to be allocated to a processor the task is scheduled at the earliest *start time* possible. The task may be started when the processor becomes available after previous processing, and the task receives the data produced by its task graph parents. The scheduled *stop time* of a task is simply the sum of its scheduled start time and its processing time (specified by the task graph). A task receives data from a task graph parent at the scheduled stop time of the parent if the two tasks are located on the same processor. Otherwise, the communication time of the data must be added to the data receive time.

When allocating the non-allocated task with the largest b-level to a processor, the processor which allows the earliest task start time is selected. This corresponds to a greedy

step towards the goal of minimizing the estimated latency of the task graph. Consequently, the processor selection is determined by the location of the tasks parents in the task graph as well as the communication time of the corresponding data.

This iteration continues until all the tasks have been allocated. Finally, the scheduled stop time,  $ready\_time(p_i)$ , of the last task to be executed on each processor is returned.

To conclude, the estimated latency of the task graph corresponds to the largest processor ready time. Also, by taking the standard deviation of the processor ready times, we measure how well the scheduling algorithm was able to balance the processing load on the available processors. These two measurements are used by the ARCAMIDE algorithm to search for task graphs with low latency, and which take advantage of the available processors without considering pipelining.

#### 5.2.4 The ARCAMIDE Algorithm

In this section we describe a heuristic search algorithm, the ARCAMIDE algorithm, which prunes a “brute force” task graph in order to offer tradeoffs between estimated error rate and latency. An important goal in this context is to take advantage of the available processors so that the temporal resolution is maximized.

Note that the ARCAMIDE algorithm does not remove the classification tasks from the task graph. Without classifiers the task graph will only output low-level features and no high-level concepts. This means that the classifier tasks should be treated as a special case. Therefore, we partition the tasks (in the brute force task graph) into Streaming, Filtering and feature Extraction tasks, hereafter denoted SFE-tasks, and classification tasks, hereafter denoted C-tasks.

If the search for a task graph that does not violate the QoS requirements is to be computationally practical, only a very small number of the possible task graphs may be evaluated. E.g. if there are no edges in a task graph containing  $n$  SFE-tasks, there are  $2^n$  possible candidate subgraphs. Consequently, the choice of which subgraphs to evaluate is of paramount importance.

In contrast to evaluating all the possible subgraphs of the “brute force” task graph, the ARCAMIDE algorithm consists of two task selection stages. In both stages of the algorithm, the most inefficient parts of the task graph (when considering estimated error rate and latency) are pruned. Roughly stated, this procedure corresponds to a standard sequential backward feature subset search (see [7]), extended to handle task graphs. The task graph search is performed backwards, rather than forwards, in order to avoid a computationally expensive n-step look-ahead search, made necessary by the task graph data dependencies, and in some cases by complexly interacting features. Obviously, other feature subset search procedures can be applied by the ARCAMIDE algorithm (such as beam search [24], genetic algorithms, or branch and bound search [7]) by extending them to handle task graphs. However, due to its simplicity, computational efficiency and goal-directed behavior this paper applies a sequential backward task graph search procedure.

The ARCAMIDE algorithm (see Figure 6) takes as input a set of SFE-tasks, a set of C-tasks, a task irrelevance threshold  $i$ , and a parallelizability threshold  $c$ . The irrelevance threshold is used for removing tasks irrelevant to the content analysis goal. That is, the error rate of each SFE-task

---

```

; The ARCAMIDE algorithm
arcamide(SFE, C, i, c)
; Stage 1
FOR EACH t IN SFE
; Removes t if irrelevant
IF er({t} ∪ desc(t, SFE)) > i THEN
SFE := SFE \ ({t} ∪ desc(t, SFE));
; Stage 2
WHILE #SFE > 0 DO
; Determines whether critical paths
; need to be shortened
IF par(SFE ∪ C) > c THEN
; Shortens critical paths
SFE' := ARGMAX t IN SFE:s_level(t);
t_max := ARGMAX t IN SFE':ef(t, SFE);
ELSE
t_max := ARGMAX t IN SFE:ef(t, SFE);

; Removes t_max and its
; task graph descendants
SFE := SFE \ ({t_max} ∪ desc(t_max, SFE));

; Outputs SFE ∪ C, error rate, and
; processing time of F
OUTPUT <SFE ∪ C, er(SFE), le(SFE ∪ C)>;

RETURN;

```

---

Figure 6: The ARCAMIDE algorithm.

(and its task graph descendants) is estimated individually. If the resulting estimated error rate is not significantly better than what is achieved with pure guessing,  $i$ , the SFE-task is removed along with its descendants. This completes stage one of the ARCAMIDE algorithm.

The parallelization threshold  $c$  controls whether critical paths are to be shortened or the least efficient tasks are to be removed in stage two of the ARCAMIDE algorithm. We take  $par(SFE \cup C) > c$  as an indication that the scheduling algorithm is not able to take advantage of the available processors due to the length of critical paths. Then, the only way to reduce the latency is to remove SFE-tasks at the end of these paths:

$$SFE' := \underset{t \in SFE}{argmax} s\_level(t).$$

If, on the other hand,  $par(SFE \cup C) \leq c$  we take this as an indication that the scheduling algorithm is able to balance the processing load between the available processors. Accordingly, the least “efficient” SFE-task should be removed from the task graph along with its descendants.

Among the tasks considered for removal, the task  $t$  that maximizes the efficiency function

$$ef(t, SFE) \equiv \frac{er(SFE) - er(SFE \setminus (\{t\} \cup desc(t, SFE)))}{le(\{t\} \cup desc(t, SFE))}$$

is removed from  $SFE$  in addition to its task graph descendants in  $SFE$ ,  $desc(t, SFE)$ . Here,  $er(T)$  denotes the estimated error rate of task set  $T$ , and  $le(T)$  denotes the estimated latency of task set  $T$ . In short, the efficiency function rewards tasks which contribute to maintaining the estimated error rate of  $SFE$  and which in addition are computationally cheap.

When the ARCAMIDE algorithm stops, it has generated a sequence of task graphs with different estimated error rate/latency tradeoffs. These task graphs are sought configured to fully utilize the specified processing environment without considering pipelining. The task graph which best fits the provided QoS requirements (in terms of latency, temporal resolution, and accuracy) is selected for deployment in the actual processing environment, as discussed in the following subsection.

### 5.3 Scalability

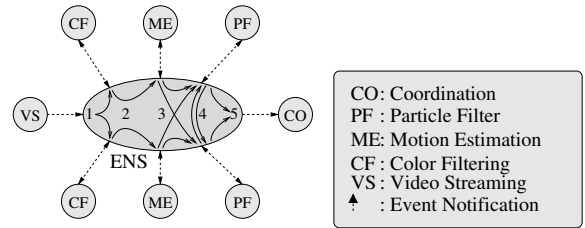
The distributed and parallel processing of one of the task graphs suggested by the ARCAMIDE algorithm enables that QoS requirements can be satisfied by throwing processing resources at the problem. In our architecture components are the unit of deployment. The scalability of the overall application is determined by the scalability of the inter component communication mechanism and the scalability provided at the different levels, i.e. video streaming, filtering/transformation, feature extraction, and classification [8]. We now describe how scalability is achieved in each of these cases.

#### 5.3.1 Event-based Component Interaction

An event-based interaction mechanism enables scalable and independent parallelization of the different levels in the content analysis hierarchy, as described in the following. From Figure 1, 2, and 3, it should be clear that components interact in different ways, such as one to one, one to many (sharing or partitioning of data), many to one (aggregation), and many to many.

In [9], we argue that the requirements for the real-time content analysis application domain fit very well with the publish/subscribe interaction paradigm, leading to an event-based interaction model. Event-based interaction provides a number of distinguishing characteristics, such as asynchronous many to many communication, lack of explicit addressing, indirect communication, and hence loose coupling. Event-based systems rely on some kind of event notification service. A distributed event notification service is realized by a number of cooperating servers. Clients connect to these servers and are either *objects of interest*, *interested parties*, or both. An object of interest publishes event notifications, or just notifications for short. In *content-based* publish/subscribe systems, such as [3, 28, 30, 33], a notification may be a set of type, name, and value tuples. Interested parties subscribe in order to express interest in particular notifications. A subscription is an expression with constraints on the names and values of notifications. The responsibility of the event notification service is routing and forwarding of notifications from objects of interest to interested parties, based on content, i.e. the type, name, and value tuples. The servers jointly form an overlaid network of content-based routers. A survey of the publish/subscribe communication paradigm and the relations to other interaction paradigms are described in e.g. [11].

In our architecture, the different components connect to and communicate through the event notification service, as illustrated in Figure 7 for the application configuration in Figure 3. As a consequence, a component does not need to know if notifications have been generated by a single or a number of components or the location or the identity of the other components. The bindings between components are loose and based on what is produced rather than by whom.



**Figure 7: Inter component communication for the configuration in Figure 3. Components interact through an Event Notification Service, labeled ENS.**

The what- rather than whom-characteristics of event-based communication is a key factor for achieving the flexible parallelization and distribution. As an example, assume that each PF component subscribes to notifications covered by the following subscription:

```
src=vs func=me
```

Assume further that each ME component publishes the calculated motion vectors as notifications:

```
src=vs func=me time=[t,dt] block=[1,1] vector=[ 0 ,0]...
src=vs func=me time=[t,dt] block=[3,2] vector=[-1 ,0]...
src=vs func=me time=[t,dt] block=[4,4] vector=[ 0 ,0]
```

The event notification service is then responsible for forwarding these notifications to the PF components, as indicated by label 3 in Figure 7. As a result, from a ME component's point of view it does not matter if there is a single or a number of components interested in the published notifications. Similarly, from a PF component's point of view it does not matter if the motion vectors have been calculated by a single or a number of ME components. This illustrates that event-based interaction enables independent parallelization of the different levels in the content analysis hierarchy. In Section 6, we present some performance numbers for a scalable distributed content-based event notification service, which is able to take advantage of native IP multicast support.

#### 5.3.2 Video Streaming

Real-time video is quite challenging with respect to processing requirements, the massive amounts of data, and the imposed real-time requirements. A video streaming source which must handle each and every interested component individually will not scale. Therefore, the sender side processing and network bandwidth consumption should be relatively unaffected by the number of receiving components.

Scalable one to many communication is what IP multicast has been designed for. However, sending a full multicast video stream to all receivers wastes both network and receiver processing resources when each receiver only processes some regions in each video frame. In [23], heterogeneous receivers are handled by layered video coding. Each layer encodes a portion of the video signal and is sent to a designated IP multicast address. Each enhancement layer depends on lower layers and improves quality spatially

and/or temporarily. Parallel processing poses a related kind of heterogeneity challenge, but an additional motivation is the distribution of workload by partitioning data. When using an event notification service for video streaming, as described in [4, 9, 32], the video streaming component may send different blocks of each video frame as different notifications. By publishing each video frame as a number of notifications, interested parties may subscribe to only a certain part of a video stream and thereby reduce resolution both spatially and temporally. This strategy, coupled with an event notification service capable of mapping notifications onto IP multicast communication, provides scalable video streaming.

### 5.3.3 Filtering and Transformation

If the representation, the spatial resolutions, or the temporal resolutions offered by a S component is not appropriate for an E component, filtering and transformation is necessary. Filtering and transformation bridge the gap between what a S component offers and an E component can handle. Scalable filtering and transformation requires that an F component may process only some part of a video stream. Therefore, in our approach an F component may subscribe to only some blocks of the video stream, both spatially and temporally, illustrated in Figure 7, labeled 1. The filtered and transformed blocks are then published as notifications, labeled 2 in the same figure. As a result, filtering and transformation is efficiently distributed and parallelized. This is also illustrated in Figure 3, where the left and the right part of each video frame is received by different motion estimation components.

### 5.3.4 Feature Extraction

A feature extraction algorithm operates on video segments from the filtering and transformation level (e.g. video frame blocks) and extracts quantitative information, such as motion vectors and color histograms.

A scalable solution for feature extraction requires that E components may process only some part of a filtered video stream. Some feature extraction algorithms require relatively small amounts of processing, such as a color histogram calculation which may only require a single pass through each pixel in a video frame. But even such simple operations may become costly when applied to a real-time high quality video stream. Additionally, the algorithms may be arbitrarily complex, in general.

Feature extraction algorithms for video, such as calculations of motion vectors, color histograms, and texture roughness, often operate locally on image regions. In our architecture spatial parallelization and distribution of such feature extractors are supported by a block-based approach.

Our implementation of a motion estimation component allows calculation of motion vectors for only some of the blocks in a video frame. In Figure 8, the motion vectors calculated by a single component have been drawn into the video frame. The blocks processed are slightly darker and they also have the motion vectors drawn, pointing from the center of their respective block. The motion vectors indicate that the person is moving to the left.

The calculated motion vectors are published as a notifications. The event notification service forwards each notification to the interested subscribers, as illustrated by label 3 in Figure 7.



Figure 8: Block-based motion estimation example.

### 5.3.5 Classification

The final logical level of our architecture is the classification level. At the classification level each video segment is assigned a content class based on features extracted at the feature extraction level. For instance, if each video frame in a video stream is divided into  $m \times n$  blocks as seen in the previous section, the classification may consist of deciding whether a block contains the center position of a moving object, based on extracted motion vectors.

The classification may become a processing bottleneck due to the complexity of the content analysis task, the required classification rate, and the required classification accuracy. E.g., rough tracking of the position of a single person in a single low rate video stream may be possible using a single processor, but accurately tracking the position of multiple people as well as their interactions (talking, shaking hands, etc.) could require several processors. Multiple video streams may increase the content analysis complexity even further. In short, when the classifier is running on a single processor, the classification may become the processing bottleneck of the content analysis application.

The particle filter (PF) [19] is an approximate inference technique that allows real-time DBN-based video content analysis. In the following we briefly describe our use of the PF in more detail. Then we propose a distributed version of the PF, and argue that the communication and processing properties of the distributed PF allow scalable distributed classification, independent of distribution at the other logical levels.

Our PF is generated from a dynamic Bayesian network specifying the content analysis task. During execution the PF partitions the video stream to be analysed into time slices, where for instance a time slice may correspond to a video frame. The PF maintains a set of particles. A single particle is simply an assignment of a content class to each video segment (e.g. object or background) in the previously analysed time slices, combined with the likelihood of the assignment when considering the extracted features (e.g. motion vectors). Multiple particles are used to handle noise and uncertain feature-content relationships. This



**Figure 9: The center position of the tracked object, calculated by the coordinator, has been drawn as a white rectangle.**

means that multiple feature interpretations can be maintained concurrently in time, ideally until uncertainty can be resolved and noise can be suppressed. When a new time slice is to be analysed, each particle is independently extended to cover new video segments, driven by the dynamic Bayesian network specification. In order to maintain a relevant set of particles, unlikely particles are then systematically replaced by likely particles. Consequently, the particle set is evolved to be a rich summarization of likely content interpretations. This approach has proven effective in difficult content analysis tasks such as tracking of objects. Note that apart from the particle replacement, a particle is processed independently of other particles in the PF procedure.

In order to support scalability, we propose a distributed version of the PF. The particles of the single PF are parted into  $n$  groups which are processed on  $n$  processors. An event based communication scheme maintains global classification coherence. The communication scheme is illustrated in Figure 7 and discussed below.  $n$  PF components and a coordinator (CO) component cooperate to implement the particle filter. Each PF component maintains a local set of particles and executes the PF procedure locally. When a new time slice is to be analysed, the components operate as follows. First,  $m$  locally likely particles are selected and submitted to the other PF components through the event notification service (label 4 in Figure 7). Then, each PF component executes the PF procedure on the locally maintained particles, except that the local particles also can be replaced by the  $(n - 1)m$  particles received from the other PF components. After execution, each PF component submits the likelihood of video segment content classes to the coordinator (label 5 in Figure 7) which estimates the most probable content class of each video segment. E.g., in Figure 9 the output of the CO component has been drawn based on the object position likelihoods produced by the PF components.

In the above communication scheme only  $2n + 1$  messages are submitted per time slice, relying on native multicast support in the event notification service. As shown empirically

in [15], by only submitting a single particle per message no loss of accuracy is detected in the object tracking case.

## 6. EMPIRICAL RESULTS

In this section, we first present some empirical results for the ARCAMIDE algorithm, when applied to our object tracking application. Then we present the measured performance for this application, when executed by a varying number of processors. Lastly, performance numbers are presented for our distributed content-based event notification service, which should reduce an observed bottleneck problem for the first implementation of this application.

### 6.1 Accuracy and Latency

We now modify the object tracking example application from Figure 3 to evaluate the ARCAMIDE algorithm empirically. First of all, we assume that each video frame in the video stream is partitioned into  $8 \times 8$  video frame blocks. The modification consists of adding a color histogram calculation task and a texture calculation task to each video frame block. We assume that the motion estimation in a video frame block depends on color-to-grey level filtering of that block. Finally, the goal of the content analysis application is refined to recognition of whether an object passes from the left to the right or from the right to the left.

Thus, we have five different types of SFE-tasks related to each video frame block: streaming, color-to-grey level filtering, motion estimation, texture calculation, and color histogram calculation. These have different content analysis and processing characteristics. In our simulation, when using motion to detect an object in a video frame block, the probability of false positives and false negatives are assumed to be 0.3. Likewise, when using texture the probability of false positives is assumed to be 0.5 and the probability of false negatives is assumed to be 0.3. When using color the latter probabilities are reversed. Color-to-grey-level filtering only produces intermediate results used in the motion estimation. Obviously, the specified probabilities depend on the environment (e.g. sunshine, darkness) and are here set to reflect rather difficult environment conditions. Finally, the processing time of the streaming task is set to 3 ms, and the processing time of the other tasks are set to 1 ms. The transmission time of a video frame block (either color-to-grey level filtered or not) across the network is considered to be 1 ms.

To evaluate the ARCAMIDE algorithm we trained two 10-state Hidden Markov Models (one for each object passing direction) on 1000 simulated video frame sequences of objects moving from the left to the right and 1000 simulated video frame sequences of objects moving from the right to the left. Here, an object appears on average twice in each video frame block of the two center rows when passing the camera view. We used another independent set of simulated video frame sequences (of identical size) to prune the task graph.

When trading off estimated error rate and latency, the ARCAMIDE algorithm behaves as shown in Figure 10, 11, and 12 respectively for 1 processor, 10 processor, and 100 processor environments. When selecting SFE-tasks for 1 processor, we see from Figure 10 that the estimated latency initially can be reduced with little increase in estimated error rate (while inaccurate SFE-tasks are removed). However, when e.g. SFE-tasks near the first and eighth video

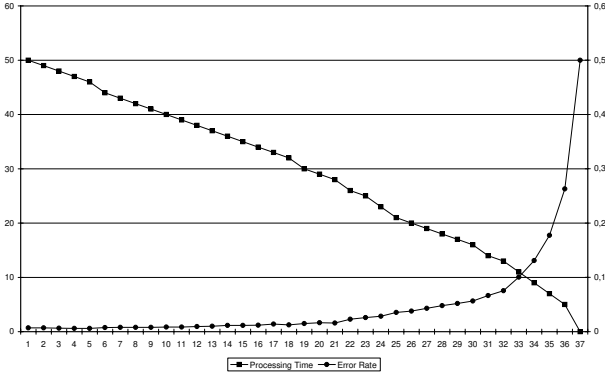


Figure 10: The estimated latency and error rate (y-axis) after each task removal (x-axis) - 1 CPU.

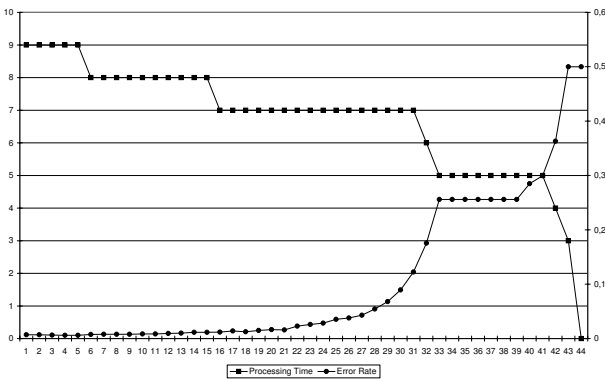


Figure 11: The estimated latency and error rate (y-axis) after each task removal (x-axis) - 10 CPUs.

frame block columns (the object entry regions) must be removed, the estimated content analysis error rate increases more dramatically. When introducing 10 processors, we see in Figure 11 that the estimated latency is reduced in steps — the processing time of all the processors must be reduced before the latency can be reduced. Also note that the data dependencies between color-to-grey level filtering and motion estimation make these tasks the target of removal from removal number 28 and thereafter. When considering 100 processors (Figure 12), initially only the removal of motion SFE-tasks will reduce the processing time due to the dependency of motion SFE-tasks on color-to-grey level filtering tasks. As seen in Figure 12, there are mainly two interesting task graphs; one containing all the relevant SFE-tasks and one containing only texture and color SFE-tasks.

## 6.2 Scalability of Object Tracking Application

In order to examine the scalability of our architecture, five object tracking configurations were used — targeting 1, 2, 4, 8, and 10 processors respectively. A separate computer hosted the video streaming component. The configurable parameters of the motion estimation component (e.g. the search area) and the particle filtering component (e.g. the

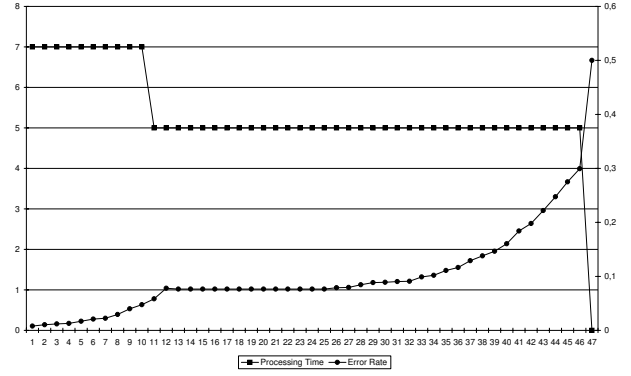


Figure 12: The estimated latency and error rate (y-axis) after each task removal (x-axis) - 100 CPUs.

Table 1: The Measured Number of Frames per Second for Different Configurations of the Real-time Object Tracking Application

	Number of processors				
	1	2	4	8	10
Ideal Frame Rate	Frames per second				
Streaming	2.5	5	10	20	25
Filtering/Feature Extraction	2.5	5	8.5	13.5	16
Classification	2.5	5	10	20	25

number of particles) were set so that they had similar processing resource requirements. The process hosting a motion estimation component also always hosted a video decoding and filtering component.

The first configuration was similar to the configuration in Figure 2. One video decoding and filtering component, one motion estimation component, one particle filter component, and one coordination component were all executed on a single processor. In the second configuration this pipeline was executed on two processors, that is, the filtering and motion estimation components were executed on one processor and the particle filter and coordination component were executed on another processor. In order to take advantage of additional processors, new configurations were created by stepwise adding one motion estimation component (and implicitly also one filtering component) and one particle filter component, each executed by a dedicated processor. The configuration illustrated in Figure 3 was executed on 4 processors. In the 10 processor configuration, five motion estimation components and five particle filtering components were used.

For the experiments, standard 1667MHz dual AMD Athlon PCs running the Linux operating system have been used. The PCs were connected by 100Mbps switched Ethernet. Additionally, standard IP multicast video streaming was used in this implementation.

The achieved frame rate, i.e. the temporal resolution, for each configuration is shown in Table 1. The frame rate increased linearly with the number of processors, except for the filtering and motion estimation part of the computation.

**Table 2: Maximum Number of Notifications Received per Second for Notifications of Different Sizes**

Locality	Notification size in bytes			
	100	500	1000	1450
	Notifications received per second			
Intra LAN	9000	7000	5500	4500
Intra Host	9000	7000	5500	4500
Intra Process	115000	100000	85000	75000

This was caused by the IP multicast based video streaming used in these experiments. Each filtering component had to decode and filter the complete IP multicast video stream, despite the fact that the motion estimation component processed only some blocks of each video frame. The ability of the distributed classifier to handle the full frame rate was tested on artificially generated features.

### 6.3 Event Notification Service

As a solution to the bottleneck observed in the previous experiment, an event-based approach to video streaming has been developed. E.g. a filtering component may register interest in only certain blocks and reduce resolution both spatially and temporally. However, a scalable and high performance event notification service is required in order to provide the necessary throughput. In [10], we present the architecture, the implementation, and the measured performance for our distributed local area network content-based event notification service. The service provides both intra process, intra host, and intra LAN communication.

The throughput for components located on different computers (intra LAN) varied from 4500 notifications per second (1450 bytes each) to 9000 notifications per second (100 bytes each), as illustrated in Table 2. A maximum of approximately 6.5 MBps (MBytes per second) was measured. The performance of the service was unaffected when there were a number of interested parties for the same notifications, due to the fact that the service utilizes native IP multicast support. Additionally, the service is able to isolate different parts of the “event notification space” by mapping notifications to different multicast addresses. As a result, the service is able to handle a number of video streams concurrently.

## 7. CONCLUSION AND FURTHER WORK

In this paper we have presented a general architecture for distributed real-time video content analysis applications. Furthermore, we have proposed a model for specifying requirements of timeliness and accuracy that can be used to deduce the application’s resource requirements from a given QoS specification.

A salient feature of the architecture is its combination of probabilistic knowledge-based media content analysis with QoS and distributed resource management to handle QoS requirements. A further feature is its independent scalability at multiple logical levels of distribution to be able to meet increasing QoS requirements in different QoS dimensions.

This has been achieved by first developing a parallel version of an approximate inference technique known as the particle filter. We demonstrated that the parallel particle filter allows for real-time video content analysis based on dynamic Bayesian networks. Next, we presented an al-

gorithm for balancing the requirements of timeliness and accuracy against available distributed processing resources. We demonstrated its behavior for a real-time motion vector based object tracking application.

Independent scalability at multiple logical levels was primarily achieved through the development of a high performance event notification service as the preferred communication mechanism of the architecture. The scalability was demonstrated through experiments with an implementation of the application mentioned above.

Our future work will include the development of a complete QoS management architecture for real-time video content analysis applications. The work presented here represents steps towards that goal.

## 8. ACKNOWLEDGMENTS

We would like to thank all persons involved in the DMJ (Distributed Media Journaling) project for contributing to the ideas presented in this paper. The DMJ project is funded by the Norwegian Research Council through the DITS program, under grant no. 126103/431.

## 9. REFERENCES

- [1] D. Beymer, P. McLauchlan, B. Coifman, and J. Malik. A Real-time Computer Vision System for Measuring Traffic Parameters. In *Computer Vision and Pattern Recognition (CVPR’97), San Juan, Puerto Rico*, pages 495–501. IEEE, June 1997.
- [2] A. Bors and I. Pitas. Prediction and tracking of moving objects in image sequences. *IEEE Transactions on Image Processing*, 9:1441–1445, August 2000.
- [3] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.
- [4] D. Chambers, G. Lyons, and J. Duggan. Stream Enhancements for the CORBA Event Service. In *Proceedings of the ACM Multimedia (SIGMM) Conference, Ottawa*, October 2001.
- [5] S.-F. Chang and H. Sundaram. Structural and Semantic Analysis of Video. In *Multimedia and Expo 2000 IEEE*, volume 2, pages 687–690, 2000.
- [6] C. Chen, Z. Jia, and P. Varaiya. Causes and Cures of Highway Congestion. *Control Systems Magazine, IEEE*, 21(6):26–32, December 2001.
- [7] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3), 1997.
- [8] V. S. W. Eide, F. Eliassen, O.-C. Granmo, and O. Lysne. Scalable Independent Multi-level Distribution in Multimedia Content Analysis. In *Proceedings of the Joint International Workshop on Interactive Distributed Multimedia Systems and Protocols for Multimedia Systems (IDMS/PROMS), Coimbra, Portugal*, LNCS 2515, pages 37–48. Springer-Verlag, November 2002.
- [9] V. S. W. Eide, F. Eliassen, O. Lysne, and O.-C. Granmo. Real-time Processing of Media Streams: A Case for Event-based Interaction. In *Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS’02), Vienna, Austria*, pages 555–562. IEEE Computer Society, July 2002.

- [10] V. S. W. Eide, F. Eliassen, O. Lysne, and O.-C. Granmo. Extending Content-based Publish/Subscribe Systems with Multicast Support. Technical Report 2003-03, Simula Research Laboratory, July 2003.
- [11] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys (CSUR)*, 35:114–131, June 2003.
- [12] A. R. François and G. G. Medioni. A Modular Software Architecture for Real-Time Video Processing. In *Proceedings of the Second International Workshop on Computer Vision Systems (ICVS), Vancouver, Canada*, volume 2095 of *Lecture Notes in Computer Science*, pages 35–49. Springer, July 2001.
- [13] A. Garg, V. Pavlovic, J. Rehg, and T. Huang. Integrated Audio/Visual Speaker Detection using Dynamic Bayesian Networks. In *Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000*, pages 384–390, 2000.
- [14] O.-C. Granmo. Automatic Resource-aware Construction of Media Indexing Applications for Distributed Processing Environments. In *Proceedings of the 2nd International Workshop on Pattern Recognition in Information Systems (PRIS2002)*, pages 124–139. ICEIS Press, April 2002.
- [15] O.-C. Granmo, F. Eliassen, O. Lysne, and V. S. W. Eide. Techniques for Parallel Execution of the Particle Filter. In *Proceedings of the 13th Scandinavian Conference on Image Analysis (SCIA2003)*, volume 2749 of *Lecture Notes in Computer Science*, pages 983–990. Springer, June 2003.
- [16] F. V. Jensen. *Bayesian Networks and Decision Graphs*. Series for Statistics for Engineering and Information Science. Springer Verlag, 2001.
- [17] Y.-K. Kwok and I. Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381–422, 1999.
- [18] B. Li and R. Chellappa. A generic approach to simultaneous tracking and verification in video. *IEEE Transactions on Image Processing*, 11:530–544, May 2002.
- [19] J. S. Liu and R. Chen. Sequential Monte Carlo methods for Dynamic Systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998.
- [20] V. Manian and R. Vasquez. A Computational Framework for Analyzing Textured Image Classification. In *IEEE Conference on Intelligent Systems for the 21st Century*, volume 1, pages 717–723. IEEE, 1995.
- [21] L. Marcenaro, F. Oberti, G. L. Foresti, and C. S. Regazzoni. Distributed Architectures and Logical-Task Decomposition in Multimedia Surveillance Systems. *Proceedings of the IEEE*, 89(10):1419–1440, October 2001.
- [22] J. Martínez, E. Costa, P. Herreros, X. Sánchez, and R. Baldrich. A modular and scalable architecture for PC-based real-time vision systems. *Real-Time Imaging*, 9:99–112, April 2003.
- [23] S. McCanne, M. Vetterli, and V. Jacobson. Low-complexity video coding for receiver-driven layered multicast. *IEEE Journal of Selected Areas in Communications*, 15(6):983–1001, August 1997.
- [24] T. M. Mitchell. *Machine Learning*. Computer Science Series. McGraw-Hill International Editions, 1997.
- [25] Y. Nakamura and M. Nagao. Parallel Feature Extraction System with Multi Agents -PAFE-. In *11th IAPR International Conference on Pattern Recognition*, volume 2, pages 371–375. IEEE, 1992.
- [26] M. Naphade and T. Huang. Extracting semantics from audio-visual content: the final frontier in multimedia retrieval. *IEEE Transactions on Neural Networks*, 13(4):793–810, 2002.
- [27] R. Okada, Y. Shirai, and J. Miura. Object tracking based on optical flow and depth. In *Proceedings of the International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 565–571. IEEE, December 1996.
- [28] L. Opyrchal, M. Astley, J. Auerbach, G. Banavar, R. Strom, and D. Sturman. Exploiting IP Multicast in Content-Based Publish-Subscribe Systems. In *Proceedings of Middleware*, pages 185–207, 2000.
- [29] B. Ozer and W. Wolf. Video Analysis for Smart Rooms. In *Internet Multimedia Networks and Management Systems, ITCOM, Denver Colorado USA*, volume 4519, pages 84–90. SPIE, July 2001.
- [30] P. R. Pietzuch and J. M. Bacon. Hermes: A distributed event-based middleware architecture. In *Proceedings of 1st International Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria*. IEEE Computer Society, July 2002.
- [31] W. K. Pratt. *Digital Image Processing*. Wiley-Interscience. John Wiley & Sons, Inc., 1991.
- [32] T. Qian and R. Campbell. Extending OMG Event Service for Integrating Distributed Multimedia Components. In *Proceedings of the Fourth International Conference on Intelligence in Services and Networks, Como, Italy*. Lecture Notes in Computer Science by Springer-Verlag, May 1997.
- [33] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with Elvin4. In *Proceedings of AUUG2K, Canberra, Australia*, June 2000.
- [34] W. Zhou, A. Vellaikal, and S. Dao. Cooperative Content Analysis Agents for Online Multimedia Indexing and Filtering. In *Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications*, pages 118–122, 2001.