Configuring Trees of Classifiers in Distributed Multimedia Stream Mining Systems

Brian Foo, Deepak S. Turaga, *Member, IEEE,* Olivier Verscheure, Mihaela van der Schaar, *Fellow, IEEE,* and Lisa Amini

***Abstract*—Multimedia stream mining applications require the identification of several different attributes in data content, and hence rely on a set of cascaded statistical classifiers to filter and process the data dynamically. In this paper, we introduce a novel methodology for configuring such cascaded classifier topologies, specifically binary classifier trees, in resource-constrained, distributed stream mining systems. Instead of traditional load shedding, our approach configures classifiers with optimized operating points after jointly considering the misclassification cost of each end-to-end class of interest in the tree, the resource constraints for every classifier, and the confidence level of each data object that is classified. The proposed approach allows for both intelligent load shedding as well as data replication based on available resources dynamically. We evaluate the algorithm on a sports video concept detection application and identify huge cost savings over load shedding alone. Additionally, we propose several distributed algorithms that enable each classifier in the tree to reconfigure itself based on local information exchange. We analyze the associated tradeoffs between convergence time, information overhead, and the cost efficiency of results achieved by each classifier for each of these algorithms.**

***Index Terms*—Binary classifier tree, networked classifiers, resource constrained stream mining.**

I. Introduction

THE PROCESSING and classification of continuous, high volume data streams is of paramount importance for many applications, including online financial analysis, real-time manufacturing process control, search engines, spam filters, fraud detection, online photo and video streaming sites, security, medical services [1]–[5], and so on. Distributed stream mining systems have been recently developed to support stream processing applications that require complex topologies of distributed operators [6]–[9]. Decomposing applications as topologies of distributed processing operators has merits that transcend the scalability, reliability, and performance objectives of large-scale, real-time stream mining systems [9]–[12]. Specifically, many stream classification and mining applications implement topologies (ensembles such as trees or cascades) of low-complexity binary classifiers to jointly accomplish the task of complex classification [3], [13]. Such a structure enables the successive identification of multiple attributes in the data, as well as provides significant advantages in terms of reduced resource consumption through appropriate dynamic data filtering, based on the incrementally identified attributes. It has been shown that using a tree of binary classifiers can achieve better performance compared to other techniques such as support vector machines (SVMs), rule-based techniques, and neural nets for some applications [4], [14], [15]. Furthermore, using classifiers operating in series with the same model (boosting [10]) or classifiers operating in parallel with multiple models (bagging [11]) has resulted in improved classification performance.

A key challenge in distributed real-time stream mining systems arises from the need to cope effectively with system overload due to large data volumes and limited system resources. For instance, consider a sample stream mining application that is used to analyze real-time sports video to detect different concepts, as shown in Fig. 1. The application is represented as a classifier tree based on the natural hierarchy of concepts, where each classifier identifies a different characteristic of the sports scene. Likewise, similar hierarchies exist in different types of video content (e.g., news, entertainment) as well as other types of multimedia. There is a large computational cost incurred by each classifier (proportional to the data rate) that limits the rate at which the application can handle input video, and in situations with overload it is non-trivial to decide the optimal course of action. Commonly used approaches to dealing with this problem in resource constrained stream mining are based on load-shedding, where algorithms determine when, where, what, and how much data to discard given the observed data characteristics, e.g., burst, desired quality of service (QoS) requirements [16]–[22], data value or delay constraints [23]. While naive load-shedding performs well for simple data management jobs such as aggregation, this is generally not the case for jobs involving sophisticated data classification. Recent work on intelligent and QoS driven load-shedding [24] attempts to maximize certain local quality of decision measures based on the predicted distribution of

Manuscript received March 7, 2008; revised July 13, 2008, December 20, 2008, and April 3, 2009. Date of publication July 26, 2010; date of current version March 23, 2011. This paper was recommended by Associate Editor W. Gao.

B. Foo was with the University of California, Los Angeles, CA 90066 USA. He is now with the Advanced Technology Center, Lockheed Martin Space Systems Company, Denver, CO USA (e-mail: brian.foo@gmail.com).

D. S. Turaga is with the Department of Exploratory Stream Processing, IBM T. J. Watson Research Center, Hawthorne, NY 10532 USA.

O. Verscheure is with the IBM T. J. Watson Research Center, Hawthorne, NY 10532 USA.

M. van der Schaar is with the Department of Electrical Engineering, University of California, Los Angeles, CA 90066 USA.

L. Amini is with the Exploratory Stream Processing Research Group, IBM T. J. Watson Research Center, Hawthorne, NY 10532 USA.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2010.2057012

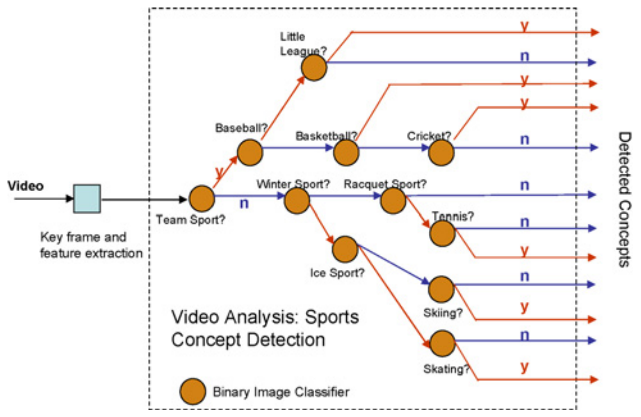


Fig. 1. Video stream mining: real-time sports scene concept detection.

feature values in future time units. The performance of such local load-shedding can be highly suboptimal in terms of end-to-end performance, as data discarded at one stage may be essential for a downstream (later stage) classifier. There has also been some work on operator scheduling for minimizing memory/resource requirements [25]–[27] and resource aware data mining [28], [29].

An alternate approach to resource constrained stream mining involves constructing topologies of classifiers based on hierarchical semantic concepts, and allowing individual classifiers in the topology to operate at different performance levels given the resources allocated to them. The performance level is determined by a classifier operating point that corresponds to the selected tradeoff between probability of detection and probability of false alarm. Examples of operating points include decision thresholds for likelihood ratio tests, or for SVM normalized scores, and so on. Hence, instead of deciding on what fraction of the data to process, as in load-shedding based approaches, such an approach determines *how* the available data should be processed given the underlying resource allocation. A solution, based on this approach, for configuring filtering applications that employ binary classifier chains (linearly cascaded classifiers) was proposed in [30]. Nevertheless, general binary tree topologies go significantly beyond linearly cascaded classifiers by providing greater flexibility in data processing, while also posing different challenges in terms of resource constrained configuration. Specifically, while excess load can be easily handled within the optimization framework for a binary classifier chain, using a single operating point for each classifier in a tree generates two output streams with a total sum output rate that is fixed. Hence, it may not be possible to simultaneously meet tight processing resource constraints for downstream classifiers along both output edges when using only one operating point.

In order to address this limitation, we propose configuring each classifier with multiple operating points, one for each output, e.g., with multiple overlapping and non-overlapping decision thresholds for the different classes. This directly enables intelligent discard of low-confidence data across output edges of each classifier when resources are scarce. Furthermore, this also allows the intelligent replication of low-confidence data across both positive and negative edges when excess resources are available, which can significantly reduce the number of classification misses. This paper makes the following contributions.

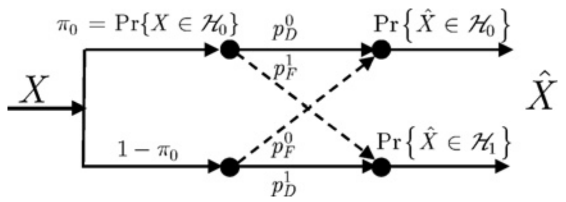
1) We define an end-to-end utility function for a tree of binary classifiers based on application-specific misclassification costs for each class of interest.
2) We formulate the problem of configuring individual classifiers with multiple operating points to maximize the end-to-end utility while satisfying resource constraints imposed by deployment on a distributed set of processing nodes.
3) We propose an algorithm based on sequential quadratic programming for configuring these classifiers with multiple operating points—to enable intelligent data discard and replication.
4) We prescribe several simplified variations of our algorithm, and analyze their performance on a real application for classification of sports video into different contexts.
5) We present distributed configuration algorithms that enable optimization to be performed separately by each classifier based on local information exchanges. We measure experimentally the performance, converge time, and information overhead of each algorithm.

Note that in this paper we focus on configuring individual classifiers with multiple operating points, but do not modify the underlying classification scheme. As mentioned earlier, operating points represent tradeoffs between the probability of detection and false alarm, available as configurable parameters for a majority of classification schemes, such as support vector machines, k-nearest neighbors, maximum likelihood, random decision trees, and so on. This paper is organized as follows. In Sections II and III, we discuss our model for binary classifiers and classifier trees. We formulate the end-to-end utility maximization problem and present a centralized solution in Section IV. We also include a discussion on the convergence of this algorithm, and a proof of the assertion on multiple operating point configurations outperforming single operating point configurations. In Section V, we introduce several distributed algorithms for reconfiguring the classifier tree. We present experimental results in Section VI and conclude in Section VII with directions for future research.

II. Modeling Binary Classifiers: Accuracy and Resource Consumption

A. Model for a Binary Classifier

A binary classifier partitions input data objects into two classes: a "yes" class, labeled $\mathcal{H}^0$ and a "no" class, labeled $\mathcal{H}^1$. An arriving data object is modeled by a random variable X, where $X \in \mathcal{H}^0$ if the object belongs to class $\mathcal{H}^0$ and $X \in \mathcal{H}^1$ if the object belongs to $\mathcal{H}^1$. The output of any classifier $\hat{X}$ may be modeled as a probabilistic function of the input X. The proportion of correctly classified samples in class $\mathcal{H}^0$ is captured by the probability of correct detection, $p_D^0 = Pr\left\{\hat{X} \in \mathcal{H}^0 | X \in \mathcal{H}^0\right\}$. The proportion of samples that are falsely classified in $\mathcal{H}^0$ can be given by the false alarm probability, $p_F^0 = Pr\left\{\hat{X} \in \mathcal{H}^0 | X \notin \mathcal{H}^0\right\}$. These probabilities



$\pi_0 = \Pr\{X \in \mathcal{H}_0\}$
p_D^0
$\Pr\{\hat{X} \in \mathcal{H}_0\}$
p_F^1
X
$\hat{X}$
$1 - \pi_0$
p_F^0
$\Pr\{\hat{X} \in \mathcal{H}_1\}$
p_D^1

Fig. 2. Model for a single classifier.

can similarly be expressed for data from class $\mathcal{H}^1$ as p_D^1 and p_F^1, respectively.[1]

Classifier performance is characterized by a detection error tradeoff (DET) curve, i.e., the set of operating points (p_F^e, p_D^e) with $e = 0, 1$ obtained by tuning its parameters. For example, in a Maximum Likelihood based classification scheme, different operating points are obtained by modifying the threshold. DET curves are concave and lie on or above the line $p_D^e = p_F^e$, between (0, 0) and (1, 1). Note that a DET curve allows us to fully parameterize p_D^e as a function of p_F^e.

The ground truth of the input data (i.e., whether it belongs to class $\mathcal{H}^0$ or $\mathcal{H}^1$) is given by the *a priori* probability π^0 for class $\mathcal{H}^0$ and $\pi^1 = 1 - \pi^0$ for class $\mathcal{H}^1$. The classifier labels data as either $\mathcal{H}^0$ or $\mathcal{H}^1$ and outputs two data streams, one per class. We consider two different measures, the outgoing throughput and goodput in each of these output streams. These correspond to the resulting total data rate, and *good* (correctly labeled) data rate in each stream, respectively. Given a unit rate of input into the classifier, the throughputs t^0, t^1 and goodputs g^0, g^1 for each output stream may thus be determined as

$$\begin{aligned} t^0 &= \pi^0 p_D^0 + \pi^1 p_F^0 \\ t^1 &= \pi^1 p_D^1 + \pi^0 p_F^1 \\ g^0 &= \pi^0 p_D^0 \\ g^1 &= \pi^1 p_D^1. \end{aligned} \tag{1}$$

Data mining applications have a cost associated with each error made in classification. This misclassification cost can be modeled as a linear function of the cost of missed detection per unit rate (for each class) (c_M^0, c_M^1), and the corresponding cost of a false alarm (c_F^0, c_F^1) [31]. Hence, the average cost c incurred by an application per unit data rate entering it can be computed by weighting the *a priori* probability with the probability of misses and false alarms, together with the various costs, that is

$$\begin{aligned} c &= c_M^0 \pi^0 (1 - p_D^0) + c_F^0 \pi^1 p_F^0 \\ &\quad + c_M^1 \pi^1 (1 - p_D^1) + c_F^1 \pi^0 p_F^1 \\ &= c_M^0 (\pi^0 - g^0) + c_F^0 (t^0 - g^0) \\ &\quad + c_M^1 (\pi^1 - g^1) + c_F^1 (t^1 - g^1). \end{aligned} \tag{2}$$

Finally, we model computational resource requirements for the classifier as being directly proportional to the rate of data entering it. This is a reasonable model as classifiers often perform the same set of operations on each data object. Hence, for unit data rate, the resources r consumed by the classifier may be computed as $r = \alpha$, where α is the resource consumption factor.

[1]Note that when the classifier uses one operating point to label each data item as $\mathcal{H}^0$ or $\mathcal{H}^1$ we have $p_D^1 = 1 - p_F^0$ and $p_F^1 = 1 - p_D^0$. This coupling is removed when we have multiple operating points, which we discuss in Section II-B.

B. Discussion: Use of Multiple Operating Points per Classifier

Before moving on to topologies of classifiers, we make an important remark about the operating points of classifiers. If a classifier uses a single threshold or operating point to label each data item into *exactly* one class i.e., as $\mathcal{H}^0$ or $\mathcal{H}^1$, we have $p_D^1 = 1 - p_F^0$ and $p_F^1 = 1 - p_D^0$, since what is not detected in one class is falsely detected in the other class. However, depending on the cost function, it may be beneficial to set different thresholds for each output class. This enables flexibility in terms of allowing data to be intelligently discarded or replicated across both branches based on the cost functions. For example, suppose a classifier uses thresholding on the resulting data prediction scores and forwards all data with scores above 0 across the "yes" branch, and all data with scores below d across the "no" branch. If $d < 0$, all data between d and 0 is dropped from both branches, leading to load shedding. If $d > 0$, all data between 0 and d is transmitted across both branches, leading to replication. The use of independent operating points for each edge outperforms arbitrary load shedding or replication, as it ensures that the data forwarded is more likely to be correctly classified, or the data dropped is less likely to be correctly classified (a sketch of the proof will be provided in Section IV-C). In fact, it can be shown that in extreme cases, forwarding or shedding the entire load across both edges can be optimal.

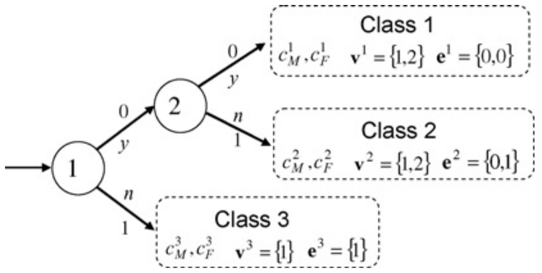
1) When complete replication is optimal: Consider a classifier system where the cost coefficients are identical for each class, with $c_M^k = 1$ and $c_F^k = 0$. Since there is no cost for false alarm, it is better to allow all data objects into all classes, such that the probability of miss is zero.
2) When complete shedding is optimal: Consider the case $c_M^k = 0$ and $c_F^k = 1$. Since each miss incurs no penalty, it is best to shed all load at the input, as this prevents any chance of false alarms in either class.

More importantly, using a single threshold naturally implies that the sum of output rates along each output edge of the classifier is equal to the input rate entering it. As we will see later for cascaded classifiers under tight system resource constraints (i.e., when both downstream classifiers are severely constrained), such a strategy may not be feasible, and may effectively require the downstream classifiers to discard data using "arbitrary" (or random) load shedding. Random load shedding is often sub-optimal compared to intelligently filtering low confidence data. Due to the apparent benefits of using multiple thresholding, for the rest of our formulations, we will assume that a classifier can have different configurations for each output.

III. Modeling Binary Classifier Trees

A. Accuracy and Utility

Consider now a set of I binary classifiers numbered $1, \cdots, I$ cascaded to construct a tree topology, an example of which, for $I = 2$ is shown in Fig. 3.



Class 1
c_M^1, c_F^1 $\mathbf{v}^1 = \{1,2\}$ $\mathbf{e}^1 = \{0,0\}$
Class 2
c_M^2, c_F^2 $\mathbf{v}^2 = \{1,2\}$ $\mathbf{e}^2 = \{0,1\}$
Class 3
c_M^3, c_F^3 $\mathbf{v}^3 = \{1\}$ $\mathbf{e}^3 = \{1\}$
1
2
0
y
n
1
0
y
n
1

Fig. 3. Example of a depth-2 tree of classifiers with three terminal classes.

TABLE I
NOTATIONS AND DEFINITIONS

$k, k \in \{1, 2, ..., K\}$	End-to-end class index
c_F^k, c_M^k	Cost of false alarms and misses For each data object in class k
$i, i \in \{1, 2, ..., I\}$	Classifier id
$v_n^k, n \in \{1, 2, ..., N_k\}$	id of the nth classifier In path from source to class k
$e_n^k, e_n^k \in \{0, 1\}$	Value of the output edge of v_n^k leading To class k (0 for "yes," 1 for "no")
t_n^k	Throughput (rate) across output Edge e_n^k of classifier v_n^k
g_n^k	Goodput (rate of correctly classified data) Across output edge e_n^k of classifier v_n^k
ϕ_n^k	Conditional *a priori* probability that input To classifier v_n^k belongs to output edge e_n^k
$\mathbf{T}_{v_n^k}^{e_n^k}$	Matrix for throughput-goodput calculation Of data passing through classifier v_n^k
$\left(p_F^{e_n^k}\right)_{v_n^k}$	False alarm configuration for output Edge e_n^k of classifier v_n^k
$\left(p_D^{e_n^k}\right)_{v_n^k}$	Probability of detection for output Edge e_n^k of classifier v_n^k
$\mathcal{P}_F$	Set of (false alarm) configurations For all classifiers and output edges
$\mathcal{P}_F^k$	Set of configurations for all classifiers On the path to class k
$\mathcal{P}_{F,i}$	Set of configurations for all classifiers Connected to classifier i
$\mathbf{p}_F$	Vector of (false alarm) configurations For all classifiers
r_i	Resource consumption of classifier i,
α_i	Resource consumption per unit input data rate
$\mathbf{y} = (y_1, ..., y_M)$	Resource constraints for Processing nodes $\{1, 2, ..., M\}$
$\mathbf{A}$	$M \times I$ placement matrix/mapping for Classifiers to processing nodes

The topology of classifiers in Fig. 3 may be used to filter and identify data from three (in general K) end-to-end classes of interest. Each end-to-end class is thus determined after a set of cascaded local classifications. Class k is characterized by N^k—the number of classifiers that data from that class need to pass through, $\mathbf{v}^k = \left(v_1^k, ..., v_{N_k}^k\right)$—the sequence of classifier indices in the path from source to class k, $\mathbf{e}^k = \left(e_1^k, ..., e_{N_k}^k\right)$—the sequence of branch "types" in the path (i.e., the binary vector, where e_n^k indicates whether the corresponding output branch of classifier v_n^k is a "yes" or "no"), c_M^k, c_F^k—cost of miss and false alarm, $\bar{t}^k$, $\bar{g}^k$—the throughput/rate and goodput of data entering class k, respectively, and $\bar{\pi}^k$, the *a priori* probability of source data belonging to class k.

For each classifier v_n^k, we define $(p_F^{e_n^k})_{v_n^k}$ and $(p_D^{e_n^k})_{v_n^k}$ to be the probability of false alarm and detection along its output edge e_n^k. Note that because classifiers have a DET curve that completely parametrizes $(p_D^{e_n^k})_{v_n^k}$ in terms of $(p_F^{e_n^k})_{v_n^k}$, we can denote the *configuration* of classifier v_n^k along output edge e_n^k solely by $(p_F^{e_n^k})_{v_n^k}$. Hence, we can define the entire (ordered) set of configurations for classifiers along the path to class k by

$$\mathcal{P}_F^k = \left\{ \left(p_F^{e_1^k}\right)_{v_1^k}, \left(p_F^{e_2^k}\right)_{v_2^k}, \cdot, \left(p_F^{e_1^k}\right)_{v_{N_k}^k} \right\} \tag{3}$$

and the entire set of configurations for all classifiers in the tree as

$$\mathcal{P}_F = \bigcup_{k=1}^{K} \mathcal{P}_F^k. \tag{4}$$

Note that v_n^k corresponds to a classifier indexed by integer $i \in 1, ..., I$ and occurs in the path of at least two end-to-end classes of interest (e.g., classifier 1 in Fig. 3). The configuration of classifier i should not depend on the class k of an incoming data object, since it is not known prior to processing what class the object belongs to. Hence, for notational convenience, the elements in $\mathcal{P}_F$ can also be expressed as a $2I$-dimensional vector $\mathbf{p}_F$

$$\mathbf{p}_F = \left[\ (p_F^0)_1 \quad (p_F^1)_1 \quad \cdots \quad (p_F^0)_I \quad (p_F^1)_I \ \right]^T. \tag{5}$$

Consider now two successive classifiers along the path to class k, classifier v_{n-1}^k and classifier v_n^k, where n denotes the index of the element in the set. The impact of filtering at v_{n-1}^k on the received data stream at v_n^k may be modeled using the conditional probability functions linking the two classifiers[2]

$$\phi_n^k = Pr\left\{ X \in \mathcal{H}_{v_n^k}^{e_n^k} | X \in \mathcal{H}_{v_{n-1}^k}^{e_{n-1}^k} \right\} \tag{6}$$

where $\left(e_{n-1}^k, e_n^k\right)$ represents one of the four possible combinations (0, 0), (0, 1), (1, 0), (1, 1) corresponding to the "yes" and "no" answers along the two successive branches. For the very first classifier in the tree, we define $\phi_1^k = Pr\left\{ X \in \mathcal{H}_{v_1^k}^{e_1^k} \right\}$.

The throughput for output data from classifier v_n^k to classifier v_{n+1}^k, t_n^k, and the corresponding goodput g_n^k, can be computed from t_{n-1}^k and g_{n-1}^k using a set of recursive relationships. We denote the source stream rate as t_0, and set the initial conditions to be $g_0^k = t_0^k = t_0$. In other words, prior to entering through the tree, each data object in the source stream is considered correctly "classified" by the source as belonging to one of the K classes. For analytical purposes, in the rest of this paper we will normalize the throughput to be $t_0 = 1$. The goodput for both the "yes" as well as the "no" output branches can be computed by

$$g_n^k = \phi_n^k g_{n-1}^k \cdot (p_D^{e_n^k})_{v_n^k}. \tag{7}$$

[2]For a tree topology only one data stream enters each classifier. Hence, we do not need to include v_n^k while parameterizing ϕ.

As opposed to the goodput, the throughput is computed differently for two different types of edges. When an edge consists of the forwarding from the "yes" output of classifier v_n^k, (i.e., $e_n^k = 1$), the throughput can be computed as

$$t_n^k = t_{n-1}^k \cdot \left(p_F^0\right)_{v_n^k} + \phi_n^k g_{n-1}^k \left(\left(p_D^0\right)_{v_n^k} - \left(p_F^0\right)_{v_n^k}\right). \tag{8}$$

Note that in the above expressions we use classifier exclusivity [32] to simplify computations.[3] Now, the throughput for a "no" output edge can be computed as

$$\begin{aligned} t_n^k &= \left(p_D^1\right)_{v_n^k}(t_{n-1}^k - g_{n-1}^k) + (1-\phi_n^k)g_{n-1}^k\left(p_F^1\right)_{v_n^k} \\ &\quad + \phi_n^k g_{n-1}^k \left(p_D^1\right)_{v_n^k}. \end{aligned} \tag{9}$$

The throughput expression in (9) consists of three separate terms that correspond to different types of data: bad data from classifier v_{n-1}^k that is correctly rejected by classifier v_n^k, good data from classifier v_{n-1}^k that is falsely rejected by classifier v_n^k, and good data from classifier v_{n-1}^k that is correctly rejected by classifier v_n^k. Note that ϕ_n^k here denotes the conditional *a priori* probability of data being *rejected* by classifier v_n^k, since class k lies on a path that uses the "no" output branch. Hence, for two classes j and k that share classifier v_i^k, but use opposite output branches, $\phi_n^j = 1 - \phi_n^k$.

We can rewrite the above expressions more concisely in matrix vector notation based on a set of transfer matrices

$$\begin{bmatrix} t_n^k \\ g_n^k \end{bmatrix} = \mathbf{T}_{v_n^k}^{e_n^k} \begin{bmatrix} t_{n-1}^k \\ g_{n-1}^k \end{bmatrix} \tag{10}$$

where $\mathbf{T}_{v_n^k}^{e_n^k}$ is given by

$$\mathbf{T}_{v_n^k}^0 = \begin{bmatrix} \left(p_F^0\right)_{v_n^k} & \phi_n^k\left(\left(p_D^0\right)_{v_n^k} - \left(p_F^0\right)_{v_n^k}\right) \\ 0 & \phi_n^k\left(p_D^0\right)_{v_n^k} \end{bmatrix}$$

$$\mathbf{T}_{v_n^k}^1 = \begin{bmatrix} \left(p_D^1\right)_{v_n^k} & \left(1-\phi_n^k\right)\left(\left(p_F^1\right)_{v_n^k} - \left(p_D^1\right)_{v_n^k}\right) \\ 0 & \phi_n^k\left(p_D^1\right)_{v_n^k} \end{bmatrix}.$$

Using the above relationship, the throughput and goodput $\bar{t}^k = t_{N_k}^k$, $\bar{g}^k = g_{N_k}^k$ for class k can be defined as

$$\begin{bmatrix} \bar{t}^k \\ \bar{g}^k \end{bmatrix} = \left(\prod_{n=1}^{N_k} \mathbf{T}_{v_n^k}^{e_n^k}\right) \begin{bmatrix} t_0 \\ g_0 \end{bmatrix}. \tag{11}$$

Similarly, the *a priori* probability $\bar{\pi}^k$ that data belongs to class k obeys the following relation:

$$\bar{\pi}^k = \prod_{n=1}^{N^k} \phi_n^k. \tag{12}$$

The average end-to-end cost of misclassification for class k can thus be computed as

$$c_{tot}^k = c_M^k(\bar{\pi}^k - \bar{g}^k) + c_F^k(\bar{t}^k - \bar{g}^k). \tag{13}$$

[3]We assume exclusivity in the classifiers, i.e., $Pr\left\{\hat{X} \in \mathcal{H}_{v_n^k}^{e_n^k} | X \notin \mathcal{H}_{v_{n-1}^k}^{e_{n-1}^k}\right\} = 0$. See [32] for details.

Finally, the end-to-end cost of misclassification, per unit data rate, for the entire tree may be computed as

$$C = \sum_{k=1}^{K} c_{tot}^k. \tag{14}$$

B. Resource Consumption and Constraints

Due to the different high complexity operations that need to be performed by each classifier on each data object, limited computational resources in the system impose a heavy constraint on the performance of the classifier tree when the volume of the incoming data stream is large. As mentioned in Section II, we model computational resource requirements for each individual classifier as being directly proportional to the rate of data entering it. In other words, each classifier v_n^k has a resource consumption factor $\alpha_{v_n^k}$, i.e., the amount of resources required per unit rate. The input rate corresponds to the throughput rate t_{n-1}^k from classifier v_{n-1}^k. Hence, the resources consumed by classifier v_n^k is given by $r_{v_n^k} = \alpha_{v_n^k} t_n^k$. Note that this notation implies that $r_{v_n^k} = r_{v_{n'}^{k'}}$ if $v_n^k = v_{n'}^{k'}$, since due to the tree structure, there is only a single input stream per classifier that is shared by both end classes.

Note also that a significant portion of the computational complexity is associated with feature extraction and preprocessing. Using domain-specific features (e.g., motion vector field model [33]) will lead to better performance (i.e., higher DET curves) and therefore better utility that directly affects the outcome of the configuration. However, as more and more specialized features are used—potentially by each different classifier, it is wasteful to gather all the feature extraction into one common process. Instead, the feature extraction may be distributed across the tree, separately for each classifier (domain independent features can still be extracted jointly). Hence, when domain-specific feature extraction is required, each classifier incurs an additional computational complexity based on the selected features. This complexity can be factored into the complexity coefficients $\alpha_{v_n^k} t_n^k$ without affecting the framework or the solution. We can now write the resource consumption vector as

$$\mathbf{r}\left(\mathcal{P}_F\right) = \begin{bmatrix} r_1 & \cdots & r_I \end{bmatrix}^T.$$

Suppose now that each classifier is uniquely placed on one of M different processing nodes. Let $\mathbf{A}_{M\times I}$ be the binary node assignment matrix that maps each classifier onto a processing node, with

$$A_{m,i} = \begin{cases} 1 & \text{if classifier } i \text{ is placed on node } m, \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

Given that processing node m has y_m available resources, any feasible configuration of classifiers in the tree needs to satisfy the constraint

$$\mathbf{A}\mathbf{r}\left(\mathcal{P}_F\right) \le \mathbf{y} \tag{16}$$

where $\mathbf{y} = \begin{bmatrix} y_1 & \cdots & y_M \end{bmatrix}^T$.

IV. Configuring Classifier Topologies

In this section, we present and solve the problem of configuring the classifier tree with multiple operating points per classifier to minimize the end-to-end cost of misclassification while meeting the system imposed resource constraints.

A. Problem Formulation

The objective of the classifier system is to minimize the total cost of misclassification subject to system resource constraints. By negating the total cost from (13) and (14) and considering resource constraints, we obtain the following utility maximization problem:

$$\begin{aligned} \max_{\mathbf{p}_F} \mathcal{U}(\mathbf{p}_F) = & \sum_{k=1}^{K} \underbrace{c_M^k\left(\bar{g}^k\right) - c_F^k\left(\bar{t}^k - \bar{g}^k\right)}_{u^k(\mathcal{P}_F^k)} \\ \text{s.t.} \quad & \mathbf{Ar}(\mathcal{P}_F) \le \mathbf{y} \text{ and } \mathbf{0} \le \mathbf{p}_F \le \mathbf{1} \end{aligned} \tag{17}$$

where we drop the term $\sum_{k=1}^{K} c_M^k \bar{\pi}^k$ in the objective function as it depends only on stream characteristics and application cost functions, and not on the classifier configurations. Note that the terms $\bar{t}^k$ and $\bar{g}^k$ can be computed using the recursive relationships defined in (11).

B. Solution via Convex Programming

In this section, we present a solution to (17) using sequential quadratic programming (SQP). SQP is a well-known, iterative optimization technique that models the nonlinear optimization problem as an approximate quadratic programming subproblem at each iteration [34], and ultimately converges to a locally optimal solution. SQP requires calculating the Lagrangian of the primal problem. The Lagrangian $\mathcal{L}(\mathbf{p}_F, \lambda, \xi, \nu)$ is given by

$$\begin{aligned} \mathcal{L}(\mathbf{p}_F, \lambda, \xi, \nu) = & \,\mathcal{U}(\mathbf{p}_F) - \xi^T(\mathbf{p}_F - 1) \\ & + \nu^T \mathbf{p}_F - \lambda^T(\mathbf{Ar}(\mathcal{P}_F) - \mathbf{y}) \end{aligned}$$

where $\lambda \in \Re^M$, $\xi, \nu \in \Re^I$ correspond to the inequalities $\mathbf{Ar}(\mathcal{P}_F) \le \mathbf{y}$, $\mathbf{p}_F \le 1$, and $\mathbf{p}_F \ge 0$, respectively. At each iteration (j), SQP approximates the Lagrangian function by a quadratic function, and linearizes the constraints

$$\begin{aligned} & \max_{\mathbf{z}} \tfrac{1}{2}\mathbf{z}^T \Gamma^{(j)} \mathbf{z} + \nabla\left(\mathcal{U}\left((\mathbf{p}_F)^{(j)}\right)\right)^T \mathbf{z} \\ \text{s.t.} \quad & \nabla\left(\mathbf{Ar}\left((\mathcal{P}_F)^{(j)}\right) - \mathbf{y}\right)^T \mathbf{z} + \mathbf{Ar}\left((\mathcal{P}_F)^{(j)}\right) \le \mathbf{y} \\ & (\mathbf{p}_F)^{(j)} + \mathbf{z} \le \mathbf{1}, \text{ and } (\mathbf{p}_F)^{(j)} + \mathbf{z} \ge \mathbf{0} \end{aligned} \tag{18}$$

where $\mathbf{z}$ is the search direction given the current configuration approximation $(\mathbf{p}_F)^{(j)}$. The matrix $\Gamma^{(j)}$ is a positive definite approximation of the Hessian matrix of the Lagrangian function and can be updated using any quasi-Newton method, e.g., the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method [35]. The $(j+1)^{(\text{th})}$ approximation is then computed as

$$(\mathbf{p}_F)^{(j+1)} = (\mathbf{p}_F)^{(j)} + \gamma^{(j)}\mathbf{z} \tag{19}$$

where the step length parameter $\gamma^{(j)}$ is determined by an appropriate line search procedure such that a sufficient decrease in the merit function is obtained. The algorithmic implementation of our solution based on SQP (see Algorithm 2) is presented in Algorithm 1.

Algorithm 1 Maximize Utility $\left(G, \mathbf{A}, c_M^k, c_F^k, \mathbf{r}, \mathbf{y}\right)$

```
Set termination threshold $\epsilon$, maximum number of iterations
$J$, and starting configuration $(\mathbf{p}_F)^{(0)}$
for $j = 0$ to $J$ do
    Compute $t_n^k$, $g_n^k$ for all $1 \le n \le N_k$, $1 \le k \le K$
    Compute $r_{v_n^k}\left((\mathcal{P}_F)^{(j)}\right) := \alpha_{v_n^k} * t_{n-1}^k$.
    Compute $\mathbf{y} := \mathbf{Ar}\left((\mathcal{P}_F)^{(j)}\right)$
    Compute $\mathcal{U}\left((\mathbf{p}_F)^{(j)}\right)$
    $\left((\mathbf{p}_F)^{(j+1)}, \Gamma^{(j+1)}\right) = \text{SQP}\left((\mathbf{p}_F)^{(j)}, \Gamma^{(j)}, \mathcal{U}, \mathbf{r}\right)$
    if $\| (\mathbf{p}_F)^{(j+1)} - (\mathbf{p}_F)^{(j)} \| < \epsilon$ then
        break;
    end if
end for
return $(\mathbf{p}_F)^{(j+1)}$
```

Algorithm 2 SQP$(\mathbf{p}_F, \Gamma, \mathcal{U}, \mathbf{r})$

```
Retrieve placement matrix $\mathbf{A}$.
Retrieve resource constraints vector $\mathbf{y}$.
Update Hessian approximation $\Gamma_{\text{new}}$.
Compute search direction $\mathbf{z}$ using (17).
Compute step size $\gamma$.
Update $(\mathbf{p}_F)_{\text{new}} := (\mathbf{p}_F) + \gamma\mathbf{z}$
return $\left((\mathbf{p}_F)_{\text{new}}, \Gamma_{\text{new}}\right)$
```

Because of the gradient descent nature of the SQP algorithm, the convergence may only be locally optimal, as it is not possible to guarantee convergence to the global optimum. However, the SQP algorithm can be initialized with multiple starting configurations in order to find a better local optimum (or even the global optimum). Since the number and size of local optima depends on the shape of the various DET curves of each classifier, a rigorous bound on the probability of finding the global optimum cannot be proven. However, certain start regions are more likely to converge to better local optima. For example, the operating point where $\left(p_F^{e_n^k}\right)_{v_n^k} = 0$ for all classifiers is a saddle point, and when starting near this point, the SQP algorithm tends to choose the steepest slope in terms of utility. Moreover, the slope of the DET curve is often very steep near $\left(p_F^{e_n^k}\right)_{v_n^k} = 0$, such that high detection probabilities can be obtained under low false alarm probabilities near the origin. Hence, the global optimum often lies along this low $\left(p_F^{e_n^k}\right)_{v_n^k}$ region. While starting near the origin does not guarantee that the point of convergence is the global optimum, intuitively, near optimal solutions can be found by starting the SQP near the origin. The starting point and solution convergence will be discussed in more detail in the experimental section.

C. Why Multiple Operating Points Always Outperform Single Operating Points

An important issue to consider is whether using multiple operating points is indeed *intelligent* in the sense that it performs at least as well as using a single operating point per classifier, and then relying on "arbitrary" load shedding. In this subsection, we show that this is always true.

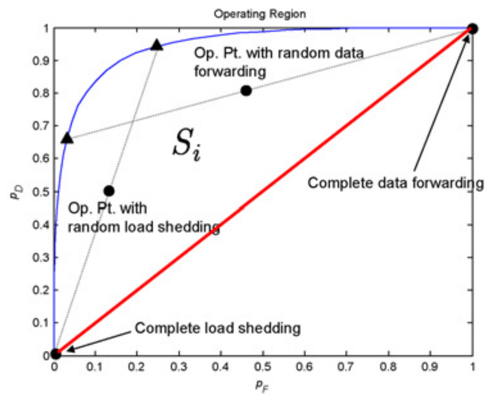


Fig. 4. Operating region for classifier (S_i).

The impact of "arbitrary" (or random) load shedding on classifier performance may be modeled equivalently as moving the operating point of the classifier below the DET curve, as shown in Fig. 4. The blue curve represents the DET curve, while the black dotted line indicates regions accessible through random load shedding given a fixed configuration. When some processing is involved using the DET curve, and some fraction of the output load is randomly shed (or randomly forwarded), the operating point will lie somewhere within the convex region given by the DET curve (blue), and the line $p_F = p_D$ (red), as indicated by the black dots. When the entire load is shed, the operating point is $(0, 0)$. When the entire load is forwarded without processing data objects (e.g., in the case of "arbitrary" replication), the operating point is $(1, 1)$. Note that the use of one operating point couples the probabilities for the two classes as $p_D^1 = 1 - p_F^0$ and $p_F^1 = 1 - p_D^0$, and therefore the entire optimization problem may be formulated purely in terms of p_D^0 and p_F^0. In this case we cannot parameterize p_D in terms of p_F as the selected point no longer lies on the DET curve. Instead, p_D can be seen as an independent variable that has constraints coupled with the corresponding p_F, where the point (p_F, p_D) must lie above $p_F = p_D$ and below the DET curve. We denote this feasible set for each classifier i by S_i^0 and S_i^1 for $((p_F^0)_i, (p_D^0)_i)$ and $((p_F^1)_i, (p_D^1)_i)$, respectively.

Hence, we propose the following effective optimization problem for random output load shedding:

$$\max_{\mathbf{p}_F} \sum_{k=1}^{K} u^k\left(\mathcal{P}_F^k, \mathcal{P}_D^k\right) \tag{20}$$

$$\text{s.t.} \quad \mathbf{Ar}\left(\mathcal{P}_F^k, \mathcal{P}_D^k\right) \le \mathbf{y}$$

$$\left(\left(p_F^0\right)_i, \left(p_D^0\right)_i\right) \in S_i^0 \text{ and } \left(\left(p_F^1\right)_i, \left(p_D^1\right)_i\right) \in S_i^1 \quad \forall i.$$

It can be shown that regardless of the configurations of other classifiers, the utility unilaterally increases in the direction of the DET curve for each classifier configuration (See Appendix A). Consequently, the optimal solution must lie on the DET curve of each classifier, i.e., $\mathcal{U}_{Mult}^* \ge \mathcal{U}_{Sing}^*$, where $\mathcal{U}_{Mult}^*$ is the optimal utility obtained using a multiple-threshold configuration without random load shedding, and $\mathcal{U}_{Sing}^*$ the optimal utility using a single-threshold configuration with random load shedding.

Another important consequence of this analysis is that the utility function does not have any local maxima in the interior of the feasible region. Hence, optimization solutions that consider both random load shedding/replication options and intelligent load shedding/replication options will always choose purely intelligent schemes (i.e., schemes that operate on the DET curve only). In other words, by allowing the use of multiple operating points, optimization solutions will always choose feasible solutions without random load shedding.

V. Distributed Optimization for Classifier Cascades

While solving a centralized SQP problem is tractable for small classifier trees, since only a few classifier configurations need to be optimized, the complexity both in terms of informational and computational overhead can be very high for a large system with many classifiers, especially when the system needs to be reconfigured frequently in a dynamic environment. Moreover, centralized algorithms have a single point of failure, such that if the central controller fails, the system can no longer adapt to time-varying data streams or nodal resource constraints. As a practical alternative to the centralized approach, we propose several distributed approaches to enable classifiers in the tree to iteratively adapt to locally optimal configurations based on local information exchanges and low complexity operations.

A. Distributed Optimization Without Resource Constraints

To keep the formulation simple, we will first ignore resource constraints. Recall from (17) that the objective for the classifier tree is to maximize the utility over all branches, $\mathcal{U}(\mathcal{P}_F) = \sum_{k=1}^{K} u^k(\mathcal{P}_F^k)$. In the distributed algorithm, each classifier i requires knowledge of the configurations (across the relevant branches) of its predecessors as well as successors in the tree. We denote this information as

$$\mathcal{P}_{F,i} = \bigcup_{k:\exists n, i=v_n^k} \mathcal{P}_F^k. \tag{21}$$

This configuration information is then used to compute classifier i's local utility function, given by

$$\mathcal{U}^i\left(\mathcal{P}_{F,i}\right) = \sum_{k:\exists n, i=v_n^k} u^k\left(\mathcal{P}_F^k\right) \tag{22}$$

We denote $\mathcal{U}^i\left(\mathcal{P}_{F,i}\right)$ as the *local utility function* for classifier i, which is a partial sum of the global utility function $\mathcal{U}(\mathcal{P}_F)$. Hence, $\mathcal{U}(\mathcal{P}_F)$ can also be regarded as a potential function [36], where unilaterally increasing $\mathcal{U}^i\left(\mathcal{P}_{F,i}\right)$ by reconfiguring classifier i also increases $\mathcal{U}(\mathcal{P}_F)$ by the same amount. In other words, by reconfiguring to optimize the local utility, the classifier also improves the overall utility. Since the potential function is upper bounded, e.g., a loose upper bound is $\sum_{k=1}^{K}\left(c_F^k + c_M^k\right)$, it can be proven that such a sequence of unilaterally reconfiguring each classifier in the tree is guaranteed to converge to a Nash equilibrium [36],

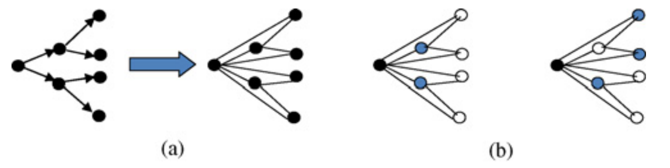


Fig. 5. Coloring for simultaneous classifier reconfiguration. (a) Information dependence. (b) Minimal coloring scheme.

Algorithm 3 Distributed Potential-based Algorithm for Utility Maximization (DP)

```
**Set** initial configuration $\mathbf{p}_F^{(0)}$, $j = 1$, and maximum number of iterations $J$
**repeat**
  **for** each independent set of classifiers $i$ **do**
    $(p_F^0)_i, (p_F^1)_i := \arg\max_{p_F^0, p_F^1} \mathcal{U}^i \left( \{ p_F^0, p_F^1 \} \cup \left( \mathcal{P}_{F,i} - \{ (p_F^0)_i, (p_F^1)_i \} \right) \right)$
    **for** each classifier $i' \in \mathcal{P}_{F,i} - i$ **do**
      **Transmit** $((p_F^0)_i, (p_F^1)_i)$ to update information at classifier $i'$.
    **end for**
  **end for**
  $j \Leftarrow j + 1$
  **Set** $(p_F^0)_i^{(j)} := (p_F^0)_i, (p_F^1)_i^{(j)} := (p_F^1)_i$ for all classifiers $i$.
**until** $j \geq J$ or $\left| \mathbf{p}_F^{(j)} - \mathbf{p}_F^{(j-1)} \right|^2 \leq \epsilon$
**return** $\left( (p_F^0)_i, (p_F^1)_i \right)^{(j)}$ for all classifiers.
```

which corresponds to a local optimal solution when resource constraints are loose.[4]

Our local utility-based solution enables information to be partially distributed, since information needs only be exchanged among local set of classifiers. Hence, classifiers that are not part of each other's local set can be *simultaneously reconfigured* without affecting each other's local utilities (i.e., information exchange is not needed). In other words, by drawing a dependency graph with edges connecting classifiers that are part of the same subtrees, the problem of configuring all classifiers in a minimal number of iterations becomes a graph coloring problem of independent classifiers. For example, the two-level binary tree shown in Fig. 5 has two minimal coloring schemes with three colors each, enabling all seven classifiers to be reconfigured using only three intervals for information exchange, thereby enabling the algorithm to converge faster. Based on this observation, we derive the distributed potential-based algorithm for utility maximization under no resource constraints, as shown in Algorithm 3.

B. Distributed Optimization with Resource Constraints

When each classifier must be configured subject to resource constraints, the potential-based *Algorithm P* may no longer converge to a local optimum. This is because a classifier may reconfigure unilaterally to maximize its local utility

[4]The Nash equilibrium implies that the partial first derivatives with respect to the equilibrium point are zero. While this property exists at either a local maximum or a saddle point, the probability of converging unilaterally to an unstable saddle point is zero unless the starting point is intentionally picked to ensure convergence to the saddle point.

Algorithm 4 Distributed Lagrangian-based Algorithm for Utility Maximization (DL)

```
**Set** initial configuration $\mathbf{p}_F^{(0)}$, $j = 1$, and maximum number of iterations $J$
**repeat**
  Update $\lambda^{(j)}$ using (23).
  Broadcast components of $\lambda^{(j)}$ to relevant classifiers.
  **for** each independent set of classifiers $i$ **do**
    $(p_F^0)_i, (p_F^1)_i := \arg\max_{p_F^0, p_F^1} \mathcal{U}^i \left( \{ p_F^0, p_F^1 \} \cup \left( \mathcal{P}_{F,i} - \{ (p_F^0)_i, (p_F^1)_i \} \right) \right) - \left( \lambda^{(j)} \right)^T (\mathbf{Ar}(\mathcal{P}_F) - \mathbf{y})$
    **for** each classifier $i' \in \mathcal{P}_{F,i} - i$ **do**
      Exchange $(p_F^0)_i, (p_F^1)_i$ to update information at classifier $i'$.
    **end for**
  **end for**
  Exchange throughput information with affected resource nodes.
  $j \Leftarrow j + 1$
  **Set** $(p_F^0)_i^{(j)} := (p_F^0)_i, (p_F^1)_i^{(j)} := (p_F^1)_i$ for all classifiers $i$.
**until** ($j \geq J$ or $\left| \mathbf{p}_F^{(j)} - \mathbf{p}_F^{(j-1)} \right|^2 \leq \epsilon$) and $\mathbf{Ar}(\mathcal{P}_F) \not\preceq \mathcal{R}$
**return** $\left( (p_F^0)_i, (p_F^1)_i \right)^{(j)}$ for all classifiers $i$.
```

Algorithm 5 Distributed Simultaneous Reconfiguration Algorithm (DS)

```
**Set** initial configuration $\mathbf{p}_F^{(0)}$, $j = 1$, and maximum number of iterations $J$
**repeat**
  Update $\lambda^{(j)}$ using (23).
  Broadcast components of $\lambda^{(j)}$ to relevant classifiers.
  **for** all classifiers $i$ **do**
    $(p_F^0)_i, (p_F^1)_i := \arg\max_{p_F^0, p_F^1} \mathcal{U}^i \left( \{ p_F^0, p_F^1 \} \cup \left( \mathcal{P}_{F,i} - \{ (p_F^0)_i, (p_F^1)_i \} \right) \right) - \left( \lambda^{(j)} \right)^T (\mathbf{Ar}(\mathcal{P}_F) - \mathbf{y})$
  **end for**
  Exchange new configurations globally.
  Exchange throughput information with affected resource nodes.
  $j \Leftarrow j + 1$
  **Set** $(p_F^0)_i^{(j)} := (p_F^0)_i, (p_F^1)_i^{(j)} := (p_F^1)_i$ for all classifiers $i$.
**until** ($j \geq J$ or $\left| \mathbf{p}_F^{(j)} - \mathbf{p}_F^{(j-1)} \right|^2 \leq \epsilon$) and $\mathbf{Ar}(\mathcal{P}_F) \not\preceq \mathcal{R}$
**return** $\left( (p_F^0)_i, (p_F^1)_i \right)^{(j)}$ for all classifiers.
```

while making many resource constraints tight for downstream classifiers, thus preventing other classifiers from reconfiguring to improve performance of their local utility metrics. Under such conditions, it is likely that a poor resource allocation solution is obtained which is *not* a local maximum.

To overcome the problems introduced by resource constraints, we propose a distributed algorithm that iteratively exchanges information between resource nodes and classifier chains, until the solution converges. Instead of applying resource constraints, the algorithm makes use of a Lagrangian multiplier vector λ for resource consumption. Based on the value of $\lambda^{(j)}$ at each iteration j, each classifier reconfigures by solving the problem

$$(p_F^0)_i, (p_F^1)_i := \arg\max_{p_F^0, p_F^1} \mathcal{U}^i \left(\left\{ p_F^0, p_F^1 \right\} \cup \left(\mathcal{P}_{F,i} - \left\{ (p_F^0)_i, (p_F^1)_i \right\} \right) \right) - \lambda^T (\mathbf{Ar}(\mathcal{P}_F) - \mathbf{y}).$$

After each classifier reconfigures, each component of the Lagrangian multiplier λ_m corresponding to a specific resource node m, is updated by

$$\lambda_m^{(j+1)} = \left[\lambda_m^{(j)} + \beta^{(j)} \left(\mathbf{A}_m \mathbf{r}(\mathcal{P}_F) - y_m \right) \right]^+ \quad (23)$$

where $[x]^+ = \max(0, x)$, $\mathbf{A}_m$ denotes the mth row of placement matrix $\mathbf{A}$, and $\beta^{(j)}$ is a positive scalar sequence that has the following properties:

$$\beta^{(j)} \to 0 \text{ as } j \to \infty \text{ and } \sum_{j=1}^{\infty} \beta^{(j)} = \infty \quad (24)$$

to guarantee convergence [37]. One sequence which works well in practice is $\beta^{(j)} \propto 1/j$. The Lagrangian has the following interpretation: the slope of each utility function is increased or decreased depending on whether the demanded resources at each node exceeds the available resources. This drives each classifier to decrease its throughput accordingly if certain resource nodes are overused. Note that while the formulation in (23) is written as a function of the entire set of configurations $\mathcal{P}_F$, the only information that is required are the resource availabilities at all downstream classifiers in $\mathcal{P}_F$, since the configuration of classifier i will not affect. We now prove that the algorithm converges to a locally optimal solution.

Proposition 1: There exists a distributed algorithm that uses sequences of updates from (23) and (24) and converges to a locally optimal solution.

Proof: The proof is a direct result of the fact that repeatedly updating configurations via (23) achieves a Nash equilibrium, or a local optimum, of the unconstrained problem [36]

$$\max_{\mathcal{P}_F} \mathcal{U}(\mathcal{P}_F) - \lambda^T (\mathbf{Ar}(\mathcal{P}_F) - \mathbf{y}). \quad (25)$$

Repeatedly solving (25) and updating the Lagrangian variable using (25) is equivalent to the subgradient method, which has been proven to converge to a local optimum [37]. ■

The pseudocode for the Lagrangian-based distributed *Algorithm DL* that alternates between primal (23) and dual (23) updates is given in Algorithm 4. In general, one could also run several iterations of (23) to ensure convergence to (25) before updating the dual variable. Additionally, we may consider another distributed approach shown in Algorithm 5, where between each dual update, all classifiers are *simultaneously* reconfigured based only on their local utility functions during previous iterations. In other words, there is no information exchange used to update utility functions between dual variable updates. Instead, all classifiers reconfigure simultaneously and at the same time exchange their requirements with processing nodes. During the following iteration, the processing nodes update the Lagrangian cost and submit this information back to the classifier elements. This algorithm is similar to the nonlinear Jacobi algorithm, which has strict requirements for convergence that may not be satisfied by the classifier tree utility function [38]. In particular, simultaneously reconfiguring classifiers can have unpredictable results on the utility, since reconfiguring one classifier changes the shape of the utility function for other classifiers, thus shifting their (unilaterally) optimal configurations. However, we will later show experimentally that this approach can lead to a faster adaptation rate under certain classifier placements.

VI. Experimental Results

A. Description of Classifiers, Concepts, and Complexity

To evaluate the performance of our proposed approach, we performed experiments by applying our algorithms to a set of sports video classifiers supplied by IBM's Multimedia Analysis and Retrieval Searching Service (MARVEL) [39]. Each state-of-the-art classifier is implemented as a SVM, and is trained specific to the characteristic it detects. The classifiers use up to 82 features, such as color, texture, and so on, which are extracted from the key-frame images.[5] The DET curves for individual classifiers were experimentally measured by testing the classifiers on a set of key-frames disjoint from the training set. The number of support vectors (complexity of the classifier) varied across different types of classifiers, but was generally of the order of 4000–21 000. Based on simulations, we observed that the complexity of processing one key-frame (image) is approximately proportional to the number of support vectors. In Table II, we list the approximate amount of processing complexity (normalized) per image for various different classifiers in our repository. In this table, C is a normalization constant for the complexity of various classifiers, given by the product of the average time of processing each image, and the speed of the processor. The total set consisted of approximately 20 000 training frames, as well as 20 000 test frames, streamed at a data rate of 1 frame per second through the system.[6] Note that streaming 1 key frame per second is likely to correspond to video with much higher frame rates, since scene changes typically occur over several seconds.

B. Motivating Example: Comparison of a Tree Topology Against a Parallel Topology

To demonstrate classification quality under resource constraints, we consider an example application that detects concepts Little League (subset of baseball), Basketball, and Cricket from image frames. We consider for this application

[5]We note that each classifier used in our experiment is implemented based on low-level features such as color correlogram, edge histogram, and so on. While other higher-level, domain specific features can also be used to improve the accuracy of individual classifiers [33], this is not the focus of this paper. Our optimization approach however can be applied to any type of classifiers.

[6]We note also that key-frame extraction and a one-time step for 82-dimensional feature extraction must be performed before classification. Since extraction requires near constant complexity, it can be performed on a given node, and the remaining resources on that node can be used for the classification process.

TABLE II
PROCESSING COMPLEXITY PER IMAGE

Classifier	Team Sports	Baseball	Little League
Complexity	$0.3884 \times C$	$0.1761 \times C$	$0.1307 \times C$
Classifier	Basketball	Cricket	Winter Sports
Complexity	$0.0772 \times C$	$0.2006 \times C$	$0.3199 \times C$
Classifier	Ice Sports	Skating	Skiing
Complexity	$0.2223 \times C$	$0.2403 \times C$	$0.2608 \times C$
Classifier	Racquet Sports	Tennis	–
Complexity	$0.1276 \times C$	$0.1720 \times C$	–

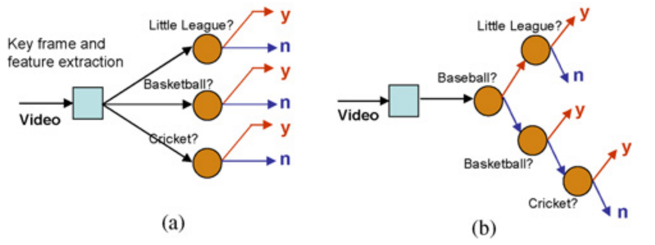


Fig. 6. Motivating classifier topologies. (a) Sports concept detection in parallel. (b) Sports concept detection with tree.

two possible topologies of classifiers placed on a single processing node, as shown in Fig. 6. We assume that after feature extraction, the remaining resources left for classification is $0.3C$ per second. The first scheme involves a parallel placement of classifiers for little league (subset of baseball), basketball, and cricket images, where the stream is filtered separately across each of the three classifiers. Note that because there are insufficient resources for processing the entire stream, a significant fraction of the load must be shed. For simplicity, we assume that data is randomly shed at the input of each classifier (i.e., buffer overflow) if the stream volume is too high. However, we mention that there are smarter ways to perform load shedding under different assumptions (e.g., temporally correlated data [24]).

The second scheme involves tree-based placement, where we add in the baseball classifier to build a tree of interest and allow classifiers in the tree to use multiple operating points, such that data objects can be intelligently filtered subject to resource constraints. In Table III, the resulting effective probabilities of detection and false alarms for each class are listed. Likewise, the cost of misclassification, as defined in Section III, with $c_M = 1$ and $c_F = 1$ (i.e., equal cost of miss and false alarm), is provided. The corresponding confusion matrix for the tree-based topology is given in Table IV, where each row corresponds to the true number of data objects belonging to that class, and each column corresponds to the number of data objects labeled as belonging to that class. Note that the detection probability is low since a large fraction of the load must be shed due to system resource constraints. Under such constraints, the tree-based topology with multiple operating points is able to achieve a comparable detection rate under a much lower false alarm rate, as it is able to intelligently discard low confidence data along paths of classifiers. Note that while the detection rate of cricket is low in the tree-based topology, this is attributed to the fact that the cricket classifier

TABLE III
FALSE ALARM AND DETECTION PROBABILITIES

Class	Parallel p_F	Parallel p_D	Tree p_F	Tree p_D
Little League	1.86%	48.14%	0.52%	48.68%
Basketball	4.04%	45.96%	0.21%	58.68%
Cricket	5.15%	44.85%	0.08%	10.97%
Cost of Error	0.2236		0.0729	

TABLE IV
CONFUSION MATRIX FOR TREE-BASED TOPOLOGY

	Little League	Basketball	Cricket	None
Little League	111	0	0	3
Basketball	0	294	0	0
Cricket	0	0	53	5
None	102	38	14	269

has a lower DET curve than the other classifiers, and hence it operates better at a steeper point of the slope where almost no false alarms occur. From our evaluations we observe that the improvements achieved by the tree configuration increases with tightening resource constraints (reduced resources).

C. Description of Centralized Solutions Used for Comparison

Consider now a larger classification tree consisting of 11 end-to-end classes, as shown in Fig. 1 in the introduction. The classes of images consist of little league, non-little league baseball, basketball, cricket, other team sports (e.g., soccer), figure skating, other ice sports (e.g., hockey), skiing, other snow sports (e.g., snowboarding), tennis, other racquet sports (e.g., badminton), and other miscellaneous sports (e.g., biking). In our experiments, we compared four different centralized solutions for configuring classifiers. The first three solutions use a single operating point per classifier, while the last solution allows for multiple operating points per classifier, where data objects can be classified under multiple classes, or dropped. The algorithms are described below.

1) *Algorithm A* uses the equal error rate (EER) configuration for each classifier, i.e., the point where the DET curve crosses the line $p_F = 1 - p_D$. This ensures that the probability of false alarm, and the probability of misses across both output edges are equal. This may seem an intuitive approach when the costs are equal for all classes.
2) *Algorithm B* jointly determines the operating point of each classifier to minimize the overall cost (maximize performance) without considering resource constraints. Conventional probabilistic load shedding techniques are then applied to overloaded classifiers. Hence, if a fraction of stream data is shed at the input of a classifier, the goodput and throughput are proportionally decreased, and the effective operating point is brought below the DET curve.
3) *Algorithm C* determines a single operating point for each classifier in the same way as algorithm B, but jointly determines the point on the DET curve, as well as the percentage of output load to shed (randomly) across each

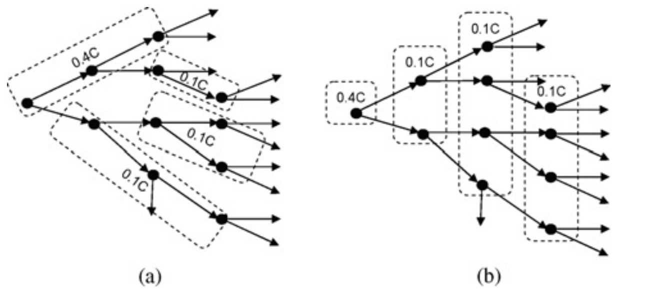


Fig. 7. Placement of classifiers. (a) Minimized node cross-talk. (b) Failure resilient.

output branch, such that the resulting resource consumption is feasible, and the overall cost is minimized. This solution is a simple extension of the algorithm in [30].

4) *Algorithm D* selects multiple operating points per classifier to independently filter and replicate data across each output edge.

D. Effect of Resource Constraints on Classifier Configurations

While the optimal placement of classifiers on nodes is beyond the scope of this paper, various placements were considered for practical reasons. Specifically, we considered three different types of system conditions and placements. The first type assumes system resources are abundant and hence rate constraints (and placement) do not need to be considered while configuring classifiers. The second type involves placing classifiers on heavily resource constrained processing nodes in a manner that reduces cross-talk between nodes, i.e., traffic across the network. The third type involves placing classifiers on nodes in a hierarchical fashion to potentially enable fault tolerance, where the most important (upstream) classifiers are placed on the most reliable nodes. The two different placements for conditions in type 2 and 3 are shown in Fig. 7. Note that while we have one node with four times the processing power as the other three nodes, this is also equivalent to instantiating the classifiers on four different nodes with 10C processing power, configuring each instantiation with equal operating points, and routing the results to the same destination. Note that similarly, parallelizing across multiple nodes can also be used to improve the bottleneck associated with computationally intensive operations such as feature extraction.

To evaluate the performance of our algorithms, we ran *Algorithms B-D* using SQP repeatedly from 50 different randomized starting points, and provided the minimum cost incurred by the application over all trials in Table V. Note that the performances of *Algorithms B and C* show significant reduction in cost over the EER configuration for all scenarios. In the non-resource constrained case, the cost of the EER was approximately 2.5 times the cost of *Algorithms B and C*. For the resource constrained cases, the cost of EER was 40-50% greater than that of *Algorithms B and C*.

Note that under resource constraints, *Algorithm C* shows minor improvement over *Algorithm B*, since it explicitly configures classifiers based on knowledge of the utility reduction as a result of load shedding. However, when resource constraints

TABLE V
MISCLASSIFICATION COST ($C_M^k = C_F^k = 1, \forall k$)

Algorithm	No Resource Constraints	Placement in Fig. 7(a)	Placement in Fig. 7(b)
A	0.8303	1.2971	1.3604
B	0.7742	0.9226	0.9442
C	0.7907	0.9158	0.8964
D	0.6959	0.8640	0.8419

are very loose, there is no real advantage in using *Algorithm C* over *Algorithm B*, as they have the same global optimal point under single threshold configuration per classifier. Finally, in all cases, enabling multiple operating points (*Algorithm D*) saves 6–12% in cost over the *Algorithms B and C*, since each classifier can operate on the DET curve for its respective output edge, leading to higher quality data filtering.

To provide a comparison of the accuracy and volume of data processed using each algorithm, the corresponding average false alarm and detection probabilities for the first placement are given as follows: $(12.69\%, 11.08\%)$ for *Algorithm A*, $(11.84\%, 15.02\%)$ for *Algorithm B*, $(4.33\%, 9.04\%)$ for *Algorithm C*, and $(3.17\%, 23.92\%)$ for *Algorithm D*. Note that *Algorithm D* achieves the highest detection rate at the lowest false alarm rate.

E. Effect of Unequal Costs on Classifier Configurations

In this section we consider the effects of different cost functions on the performance of each algorithm. In the first scenario, we set the cost functions to be $c_M^k = 1$, $c_F^k = 4$ for all classes k. In the second scenario, we set the cost functions to be $c_M^k = 4$, $c_F^k = 1$ for all classes. The experiments were performed under loose resource constraints such that the effect of the cost function on the resulting costs could be highlighted. Table VI depicts the resulting costs achieved by each algorithm.

For high costs of false alarms, we discovered significant cost savings when load shedding at the output was considered (*Algorithm C*). The reason is that, unlike *Algorithm B*, which always keeps the entire output load from each classifier, *Algorithm C* can completely shed the output load whenever the quality of decision (e.g., goodput to throughput ratio) falls below a certain threshold. In our simulations, the load was completely shed by *Algorithm C* at the edges going into classifiers "Winter Sports" and "Cricket." Nevertheless, the best solution using multiple operating points performed better because rather than shedding the entire load down certain branches, it could intelligently shed only the data objects with lower confidence levels.

For high costs of misses, a 21% cost savings resulted from using multiple operating points. This is due to the intelligent replication of data, which reduces the probability of miss for each class. For example, we discovered that approximately 18% of the data from "Team Sports" was replicated and transmitted along both the "yes" and "no" output edges, while 10% of the data from "Baseball" was replicated, and 9% of data from "Winter Sports" was replicated.

TABLE VI
MISCLASSIFICATION COST UNDER DIFFERENT c_M^k, c_F^k

Alg.	$c_M^k = 1$, $c_F^k = 4$	$c_M^k = 4$, $c_F^k = 1$
A	2.0757	2.0757
B	1.9356	1.9355
C	0.9655	1.9365
D	0.8703	1.5438

TABLE VII
EVALUATION OF DISTRIBUTED ALGORITHMS [FIG. 7(A) AND B]

Algorithm	*L*		*S*		*P*	
Placement	7(a)	7(b)	7(a)	7(b)	7(a)	7(b)
Msg/interval	9.1	8.6	32	21.5	9.2	8.1
Conv. time	120	100	48	N/A	9	12
Tot msgs exch	1092	860	1536	N/A	83	97
$c_F^k = c_M^k$ cost	0.852	0.847	0.854	0.857	0.868	0.847
$4c_F^k = c_M^k$ cost	2.842	3.948	2.858	3.948	3.158	3.948

F. Evaluation of the Distributed Algorithms

We analyzed the misclassification cost at convergence, convergence time and the amount of information exchanged for distributed algorithms *DL*, *DS*, and *DP* presented in Section IV. We measured convergence time of the distributed algorithms in number of iterations, where an iteration corresponds to a period between dual and primal updates, which may be slightly different from the actual time required for computation and information exchange in each of the three algorithms. Table VII shows these results for two different cost functions for the crosstalk-minimizing placement [Fig. 7(a)] and the failure-resilient placement [Fig. 7(b)], along with the number of messages exchanged per interval, the approximate convergence time (time to obtain a configuration $\mathbf{p}_F$ of within Euclidean distance 10^{-4} of its limit point), and the total number of messages exchanged before convergence. In all of our experiments, each distributed algorithm starts iterating from the point $p_F = 0$ for all classifiers and branches, based on our observations that this yields the minimal cost solutions.

First, note that for both the crosstalk-minimizing and failure-resilient placements, *Algorithm DL* always provides the best cost value upon convergence (and is in fact within 0.01 of the centralized solution). However, the convergence time is long, since it needs to update the Lagrangian parameters as well as exchange configuration information between different classifiers. On the other hand, *Algorithm DS* has a significantly lower convergence time than *Algorithm DL* for the crosstalk-minimizing placement, but it does not always converge for the failure-resilient placement. Finally, *Algorithm DP* had the quickest convergence time (an order of magnitude less than both *Algorithm DL* and *Algorithm DS*), since the Lagrangian vector, which constantly modifies the shape of the utility function for the other algorithms, does not need to be considered. However, *Algorithm DP* is not guaranteed to converge to an efficient solution under resource constraints.

Algorithm DS had a higher information exchange overhead than *Algorithm DL* due to frequent communication between classifiers and nodes. However, the tradeoff is that *Algorithm DS* can adapt more quickly than *Algorithm DL*, although it may not always converge precisely. In terms of performance, we showed experimentally that for the crosstalk-minimizing placement, where classifiers along the same branch share resource constraints, *Algorithm DP* can lead to about 10% higher costs than *Algorithm DL* and *Algorithm DS*. However, for the failure-resilient placement, *Algorithm DP* converges to the same local optimum as *Algorithm DL*. This is largely due to the way the resource constraints affect the global utility. For example, the hierarchical placement in Fig. 7(b) enables the root classifier to reconfigure and redistribute resources across the tree using its multiple thresholds. Based on the configuration of the root classifier, downstream classifiers can then reconfigure to locally optimize the metrics along their subtrees. This type of reconfiguration, however, is not possible with the placement scheme in 5(a) due to the coupled nature of classifiers that share resources as well as (local) utility functions. Importantly, our results suggest that *Algorithm DP* can be used as an efficient method to quickly adapt to dynamic environments. However, its optimality is highly dependent on the placement of classifiers on shared resource nodes. Hence, the choice of an appropriate distributed algorithm is based largely on both classifier placement, as well as system requirements (e.g., the system may require a highly cost-efficient solution in steady state, or a quick suboptimal solution for adaptation in highly dynamic environments).

VII. CONCLUSION

In this paper, we introduced a paradigm for configuring binary classification trees using multiple thresholds for each classifier to minimize the cost of misclassification under resource constraints. We demonstrated that, depending on the cost of false alarms and misses for each class of interest, not only does a joint consideration of an entire topology of classifiers significantly reduce the cost of misclassification, but by enabling multiple thresholds per classifier, intelligent load shedding and replication can provide further substantial savings. We also proposed various distributed algorithms and provided insight into how different placement schemes and resource constraints can affect their performances, convergence times, and information overhead. The algorithms prescribed in this paper, as well as the insights obtained from experimentation, can guide both application programmers and system designers on how to jointly configure classifiers and allocate limited resources, such that the performance of a distributed tree topology of binary classifiers is optimized.

In particular, with the growth of image sharing applications, and sites like Flickr, there is a significant demand for applications that support automated image annotation and retrieval on large-scale image sets [40]. We are building a service for such automated image annotation, by implementing a large set (spanning over 200 semantic concepts) of MARVEL [39] classifiers in conjunction with our dynamic configuration algorithms, on System S [7]. A preliminary description of our system is available in [41]. Moreover, we are in the process of a comprehensive evaluation of the performance, latency, and scaling of the system and our deployed algorithms using a large-scale distributed cluster of machines.

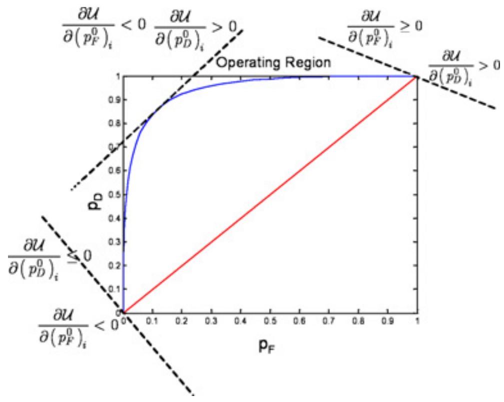


Fig. 8. Three conditions where the optimal operating point lies on the DET curve.

There are several directions for future research. One interesting extension involves dynamic and domain-specific feature extraction at each classifier—i.e., the classifier decides dynamically which feature(s) it wants to use given utility-complexity tradeoffs. This requires joint optimization of the classifier operating point with the desired feature extraction complexity, and is a complex combinatorial problem that changes as the fraction of stream data in each class changes over time. This will also involve examining the more general tree construction and placement problem. Other directions for future work include design of optimization algorithms for more generic topologies, such as using multiple classifier trees in parallel, combining parallel and serial processing topologies, configuring directed acyclic graph structures, and instantiating classifiers on more than one processing node. Finally, we are building infrastructure and applications to benchmark the performance of the algorithms on large scale datasets.

APPENDIX

Lemma 1: Given the option of arbitrary load shedding or replication, a local maximum always lies on the DET Curve.

Proof: Since utility is a linear function of $\left(p_F^0\right)_i$ and $\left(p_D^0\right)_i$, the DET curve will be optimal as long as the following two conditions never simultaneously hold:

$$\frac{\partial \mathcal{U}}{\partial \left(p_D^0\right)_i} < 0 \quad \text{and} \quad \frac{\partial \mathcal{U}}{\partial \left(p_F^0\right)_i} > 0.$$

If the two conditions simultaneously hold, the optimal achievable operating point would be to mirror the DET curve across the $p_D = p_F$ line, thereby purposely sending data along the opposite edge. Note however that this will never be the case if the condition does not hold, since the operating point will be configured to $(0, 0)$ when $\frac{\partial \mathcal{U}}{\partial (p_D^0)_i} \leq 0$, or $(1, 1)$ when $\frac{\partial \mathcal{U}}{\partial (p_F^0)_i} \geq 0$. A graphical representation of the three conditions where the optimal operating point lies on the DET curve is shown in Fig. 8.

For the "yes" output edge of classifier i, unilaterally increasing the probability of false alarm keeps the goodput fixed, but increases the throughput (and this effect propagates to the end-to-end throughput and goodput for all classes on whose paths classifier i sits). Since the utility for each class is a weighted difference between the throughput and the goodput, unilaterally increasing the probability of false alarm can only hurt the performance. Hence, $\frac{\partial \mathcal{U}}{\partial (p_F^0)_i} \leq 0$. Likewise, for the "no" output edge, $\frac{\partial \mathcal{U}}{\partial (p_D^1)_i} \geq 0$, and hence the optimal point always lies on the DET curve. ■

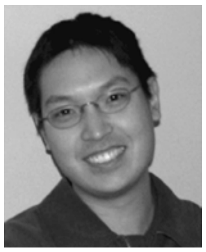
ACKNOWLEDGMENT

The authors would like to thank the MARVEL [39] team, including J. R. Smith, R. Yan, and J. Yang, for providing access to the SVM based classifiers, and the train and test image data set.

REFERENCES

[1] M. A. Shah, J. M. Hellerstein, S. Chandrasekaran, and M. J. Franklin, "Flux: An adaptive partitioning operator for continuous query systems," in *Proc. 19th ICDE*, Mar. 2003, pp. 25–36.
[2] L. Chen and G. Agrawal, "Supporting self-adaptation in streaming data mining applications," in *Proc. 20th IEEE Int. Parallel Distributed Process. Symp.*, Apr. 2006, p. 10.
[3] R. Lienhart, L. Liang, and A. Kuranov, "A detector tree for boosted classifiers for real-time object detection and tracking," in *Proc. ICME*, 2003, pp. 277–280.
[4] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly, "Detecting spam web pages through content analysis," in *Proc. 15th Int. Conf. World Wide Web*, May 2006, pp. 83–92.
[5] T. E. Senator, "Multi-stage classification," in *Proc. 5th ICDM*, 2005, pp. 386–393.
[6] J. J. C. Olston and J. Widom, "Adaptive filters for continuous queries over distributed data streams," in *Proc. SIGMOD*, Jun. 2003, pp. 563–574.
[7] L. Amini, H. Andrade, F. Eskesen, R. King, Y. Park, P. Selo, and C. Venkatramani, "The stream processing core," IBM T. J. Watson Res. Center, Hawthorne, NY, Tech. Rep. RSC 23798, Nov. 2005.
[8] Y. Xing, S. B. Zdonik, and J.-H. Hwang, "Dynamic load distribution in the borealis stream processor," in *Proc. 21st ICDE*, 2005, pp. 791–802.
[9] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, Y. Xing, and S. B. Zdonik, "Scalable distributed stream processing," in *Proc. 2nd Biennial CIDR*, Jan. 2003.
[10] R. E. Schapire, "A brief introduction to boosting," in *Proc. 16th Int. Joint Conf. Artif. Intell.*, vol. 2. 1999, pp. 1401–1406.
[11] A. Garg, V. Pavlovic, and T. S. Huang, "Bayesian networks as ensemble of classifiers," in *Proc. 16th ICPR*, 2002, pp. 779–784.
[12] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker, "Fault tolerance in the borealis distributed stream processing system," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, Jun. 2005, pp. 13–24.
[13] Y. Mao, X. Zhou, D. Pi, Y. Sun, and S. T. C. Wong, "Multiclass cancer classification by using fuzzy support vector machine and binary decision tree with gene selection," *J. Biomed. Biotechnol.*, vol. 2005, no. 2, pp. 160–71, 2005.
[14] S. Pang, "SVM classification tree algorithm with application to face membership authentication," in *Proc. Int. Joint Conf. Neural Netw.*, vol. 1. Jul. 2004.
[15] J. R. Smith, Y. Wu, and B. Tseng, "Ontology-based multi-classification learning for video concept detection," in *Proc. ICME*, 2004, pp. 1003–1006.
[16] B. Babcock, S. Babu, M. Datar, and R. Motwani, "Chain: Operator scheduling for memory minimization in data stream systems," in *Proc. SIGMOD*, 2003, pp. 253–264.
[17] N. Tatbul, U. Çetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker, "Load shedding in a data stream manager," in *Proc. 29th Int. Conf. VLDB*, 2003, pp. 309–320.
[18] B. Babcock, M. Datar, and R. Motwani, "Cost-efficient mining techniques for data streams," in *Proc. 2nd Australasian Inform. Security, Data Mining Web Intell., Software Int.*, vol. 32. 2003, pp. 109–114.
[19] N. Tatbul, U. Çetintemel, and S. B. Zdonik, "Staying FIT: Efficient load shedding techniques for distributed stream processing," in *Proc. 33rd Int. Conf. VLDB*, 2007, pp. 159–170.
[20] N. Tatbul and S. B. Zdonik, "Dealing with overload in distributed stream processing systems," in *Proc. IEEE Int. Workshop NetDB*, 2006, p. 24.

[21] N. Tatbul, "Qos-driven load shedding on data streams," in *Proc. XML-Based Data Manage. Multimedia Eng.*, 2002, pp. 566–576.
[22] S. Viglas and J. Naughton, "Rate-based query optimization for streaming information sources," in *Proc. ACM SIGMOD*, 2002, pp. 37–48.
[23] V. Eide, F. Eliassen, O. Granmo, and O. Lysne, "Supporting timeliness and accuracy in distributed real-time content-based video analysis," in *Proc. ACM Int. Conf. Multimedia*, 2003, pp. 21–32.
[24] Y. Chi, P. S. Yu, H. Wang, and R. R. Muntz, "Loadstar: A load shedding scheme for classifying data streams," in *Proc. SDM*, 2005, pp. 346–357.
[25] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query processing, approximation, and resource management in a data stream management system," in *Proc. CIDR*, 2003.
[26] B. Babcock, S. Babu, M. Datar, R. Motwani, and D. Thomas, "Operator scheduling in data stream systems," *VLDB J.*, vol. 13, no. 4, pp. 333–353, Dec. 2004.
[27] D. Carney, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring streams: A new class of data management applications," in *Proc. Int. Conf. VLDB*, 2002, pp. 215–226.
[28] M. Gaber, S. Krishnaswamy, and A. Zaslavsky, "Cost-efficient mining techniques for data streams," in *Proc. ACM 2nd Workshop Australasian Inform. Security Data Mining Web Intell. Softw. Internationalization*, 2004, pp. 109–114.
[29] W. Teng, M. Chen, and P. Yu, "Resource-aware mining with variable granularities in data streams," in *Proc. IEEE ICDM*, 2004, pp. 527–531.
[30] F. Fu, D. Turaga, O. Verscheure, M. van der Schaar, and L. Amini, "Configuring competing classifier chains in distributed stream mining systems," *IEEE J. Sel. Top. Signal Process.*, vol. 1, no. 4, pp. 548–563, Dec. 2007.
[31] M. Ciraco, M. Rogalewski, and G. Weiss, "Improving classifier utility by altering the misclassification cost ratio," in *Proc. 1st Int. Workshop Utility-Based Data Mining*, 2005, pp. 46–52.
[32] D. S. Turaga, O. Verscheure, U. V. Chaudhari, and L. Amini, "Resource management for chained binary classifiers," in *Proc. Workshop Tackling Comput. SysML*, 2006, pp. 1102–1107.
[33] L. Duan, M. Xu, Q. Tian, C. Xu, and J. Jin, "A unified framework for semantic shot classification in sports video," *IEEE Trans. Multimedia*, vol. 7, no. 6, pp. 1066–1083, Dec. 2005.
[34] P. Boggs and J. Tolle, "Sequential quadratic programming," *Acta Numerica*, vol. 4, pp. 1–51, Jan. 1995.
[35] M. Powell, "A fast algorithm for nonlinearly constrained optimization calculations," in *Numerical Analysis*, vol. 630, G. Watson, Ed. Berlin, Germany: Springer-Verlag, 1978, pp. 144–157.
[36] D. Monderer and L. Shapley, "Potential games," *Games Econ. Behav.*, vol. 14, no. 1, pp. 124–143, 1996.
[37] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA: Athena Scientific, 1999.
[38] D. P. Bertsekas and J. N. Tsisiklis, *Parallel and Distributed Computation: Numerical Methods*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
[39] J. R. Smith. *IBM Multimedia Analysis and Retrieval System (MARVEL)* [Online]. Available: http://mp7.watson.ibm.com/marvel
[40] L. Huston, R. Sukthankar, R. Wickremesinghe, M. Satyanarayanan, G. Ganger, E. Riedel, and A. Ailamaki, "Diamond: A storage architecture for early discard in interactive search," in *Proc. USENIX Conf. File Storage Technologies*, 2004, pp. 73–86.
[41] D. S. Turaga, B. Foo, O. Verscheure, and R. Yan, "Configuring topologies of distributed semantic concept classifiers for continuous multimedia stream processing," in *Proc. ACM Conf. Multimedia*, 2008, pp. 289–298.



Brian Foo received the B.S. degree in electrical engineering and computer sciences from the University of California, Berkeley, in 2003, and the Masters and Ph.D. degrees from the University of California, Los Angeles, in 2004 and 2008, respectively.

He is currently a Research Scientist with the Advanced Technology Center, Lockheed Martin Space Systems Company, Denver, CO. His current research interests include the modeling, analysis, and optimization of complex systems, including autonomous and distributed agents, cyber-physical systems, and multimedia applications and systems. He has five IEEE journal publications, and has a best paper nomination and an invited paper in design automation and SPIE conferences, respectively.

Deepak S. Turaga (M'01) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology Bombay, Mumbai, India, in 1997, and the M.S. and Ph.D. degrees in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 1999 and 2001, respectively.

He is currently a Research Staff Member with the Department of Exploratory Stream Processing, IBM T. J. Watson Research Center, Hawthorne, NY. He was with Philips Research, Amsterdam, The Netherlands, from 2001 to 2003, and with Sony Electronics, San Diego, CA, from 2003 to 2004. His current research interests include statistical signal processing, multimedia processing, coding and streaming, machine learning, and computervision applications. In these areas, he has published over 40 journal and conference papers, one book and one book chapter. He has filed over 20 invention disclosures and has participated actively in MPEG standardization activities.

Dr. Turaga received the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY 2006 Transactions Best Paper Award (with M. van der Schaar and B. Pesquet-Popescu), and is a co-author for the 2006 IEEE ICASSP Best Student Paper (with H. Tseng, O. Verscheure, and U. Chaudhari). He is an Associate Editor of the IEEE TRANSACTIONS ON MULTIMEDIA, an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, and an Associate Editor of the *Hindawi Journal for the Advances in Multimedia*.

Olivier Verscheure received the B.S. degree from the Faculte Polytechnique de Mons, Mons, Belgium, and the M.S. degree from the Swiss Federal Institute of Technology, Lausanne (EPFL), Switzerland, both in electrical engineering, in 1994 and 1995, respectively, and the Ph.D. degree in computer science from the Institute for Computer Communications and Applications, EPFL, in 1999.

Since July 1999, he has been a Research Staff Member with the IBM T. J. Watson Research Center, Hawthorne, NY. His current research interests include scalable media servers, multimedia compression and streaming, data mining, and signal processing.

Dr. Verscheure is a co-author of the 2004 IEEE International Performance Computing and Communications Conference Best Paper Award (with L. Amini and G. Paleologo) and a co-author of the 2006 IEEE ICASSP Best Student Paper (with H. Tseng, D. Turaga, and U. Chaudhari).

Mihaela van der Schaar (F'10) is currently a Professor with the Department of Electrical Engineering, University of California, Los Angeles. She holds 33 granted U.S. patents. Her current research interests include multimedia networking, communication, processing and systems, multiuser communication networks, online learning, network economics, and game theory.

Dr. van der Schaar is a Distinguished Lecturer of the Communications Society, the Editor-in-Chief of the IEEE TRANSACTIONS ON MULTIMEDIA, and a member of the Editorial Board of the IEEE JOURNAL ON SELECTED TOPICS IN SIGNAL PROCESSING. She received the NSF Career Award in 2004, the Okawa Foundation Award in 2006, the IBM Faculty Award in 2005, 2007, and 2008, and the Most Cited Paper Award from *EURASIP Image Communications Journal* in 2006. She received three ISO Awards for her contributions to the MPEG video compression and streaming international standardization activities.

Lisa Amini received the Ph.D. degree in computer science from Columbia University, New York, NY.

She currently manages the Exploratory Stream Processing Research Group, IBM T. J. Watson Research Center, Hawthorne, NY. She was with IBM, Armonk, NY, where she worked on the areas of distributed systems, networking, and multimedia for over 15 years. Her current research interests include stream mining, scalable messaging, network overlays for intelligent content distribution, data stream management, and computing cooperatives.

Dr. Amini has been and continues to be a member of the program committee for several top-tier conferences in these areas.