A Distributed Approach for Optimizing Cascaded Classifier Topologies in Real-Time Stream Mining Systems

Brian Foo and Mihaela van der Schaar, *Senior Member, IEEE*

***Abstract*—In this paper, we discuss distributed optimization techniques for configuring classifiers in a real-time, informationally-distributed stream mining system. Due to the large volume of streaming data, stream mining systems must often cope with overload, which can lead to poor performance and intolerable processing delay for real-time applications. Furthermore, optimizing over an entire system of classifiers is a difficult task since changing the filtering process at one classifier can impact both the feature values of data arriving at classifiers further downstream and, thus, the classification performance achieved by an ensemble of classifiers, as well as the end-to-end processing delay. To address this problem, this paper makes three main contributions. 1) Based upon classification and queuing theoretic models, we propose a utility metric that captures both the performance and the delay of a binary filtering classifier system. 2) We introduce a low-complexity framework for estimating the system utility by observing, estimating, and/or exchanging parameters between the interrelated classifiers deployed across the system. 3) We provide distributed algorithms to reconfigure the system, and analyze the algorithms based upon their convergence properties, optimality, information exchange overhead, and rate of adaptation to non-stationary data sources. We provide results using different video classifier systems.**

***Index Terms*—Multiagent systems, multimedia stream classification, queuing theory, systems.**

I. Introduction

RECENTLY, a variety of applications have emerged that require operations such as classification, filtering, aggregation, and correlation over high-volume, continuous data streams [1], [5], containing a variety of multimedia data, such as digital audio, video, and images. Each application can be viewed as a processing pipeline that analyzes streaming data from a set of raw data sources to extract valuable information in real time [1]. Due to spatially and proprietarily distributed datasets, as well as high computational burdens for the analytics, distributed mining systems have been recently developed [2], [3], [18], [19]. These systems leverage computational resources from a set of distributed processing nodes to deploy and run different stream mining applications on various networked topologies [3]. In such systems, stream processing jobs are often decomposed into a network of distributed operators featuring extraction, classification, aggregation, and correlation. This decomposition has merits that transcend the scalability, reliability, and performance objectives of large-scale, real-time stream mining systems [5], [8], [19], [25].

A key challenge in distributed real-time stream mining systems is how to cope effectively with system overload, while maintaining high performance under resource constraints. A commonly used approach is load-shedding, where algorithms determine when, where, what, and how much data to discard given the observed data characteristics, desired quality-of-service (QoS) requirements [9]–[13], and delay constraints [14]. While naïve load shedding performs well for simple data management jobs such as aggregation, for which the quality of job results depend only upon the sample size, this is generally not the case for jobs involving classification of data.

While it has been shown that classifiers operating in series (boosting) can result in improved classification performance [5], recent works have additionally shown that a chain of classifiers can be used to perform intelligent load shedding [16], [17]. By placing a low complexity classifier in front of a higher complexity classifier, a significant volume of stream data can be shed prior to the successive operator. This concept is similar to recent work on intelligent load shedding [15], where a load shedder attempts to maximize certain quality of decision (QoD) measures based upon the predicted distribution of temporally-correlated feature values. However, the approach in [15] considers information pertaining to only a single classifier. Without a joint consideration of resource constraints at possibly multiple downstream classifiers in the chain, the joint classification performance can be highly suboptimal, and the end-to-end processing delay for a cascade of classifiers can become intolerable for real-time applications [39], [40]. On the other hand, using an ensemble of classifiers with explicit operating points (e.g., configurable threshold, probability of false alarm) enables the entire chain to jointly optimize based upon the quality of classification, as well as the resource requirements and availabilities across utilized processing nodes [16].

Nevertheless, existing solutions are based upon an analytical model that requires classifiers to filter subset data from the pre-

Manuscript received July 01, 2009; revised November 05, 2009; accepted April 05, 2010. Date of publication June 01, 2010; date of current version October 15, 2010. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Sharathchandra Pankanti.

B. Foo was with the Deptartment of Electrical Engineering, University of California Los Angeles (UCLA), Los Angeles, CA, 90095-1594 USA. He is now with Lockheed Martin Space Systems, Sunnyvale, CA 94089 USA (e-mail: brian.foo@gmail.com).

M. van der Schaar with the Deptartment of Electrical Engineering, University of California Los Angeles (UCLA), Los Angeles, CA 90095-1594 USA (e-mail: mihaela@ee.ucla.edu).

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/TIP.2010.2051866

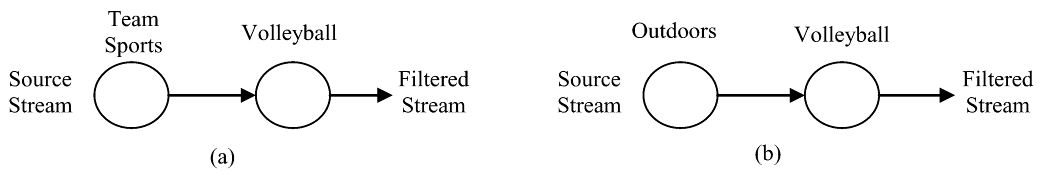


Fig. 1. Chain of image classifiers with (a) subset filtering and (b) without subset filtering.

vious classifier in the chain or tree (e.g., the exclusivity principle [16]). For example, as shown in an image classification example in Fig. 1(a), a low complexity "team sports" classifier can be used to filter out data that is not a team sport, thereby shedding a significant volume before passing the stream to the "volleyball" classifier. While this approach is viable for classifiers trained at the same site, classifiers located across different sites may be trained to detect different semantic high level features that do not obey simple subset relationships, e.g., one site trained to detect "outdoors" images, and the other "volleyball" images, as shown in Fig. 1(b). Furthermore, many sites may not be willing to share their proprietary datasets or analytics, nor capable of providing a repository to store an entire collection of data across multiple sites [19], [42], which prevents the construction of centralized models [22]–[24]. This presents a major challenge to optimizing the performance of a delay-sensitive stream mining application, as both the quality and delay across a classifier chain cannot be analytically determined due to informational constraints. In some prior works, classifiers use state space models and Bayesian networks to estimate the underlying stream data distribution as a function of classifier configurations [20], [21]. However, forming accurate models may require a large state space, and learning the parameters for these models can take a long time, making this approach less suitable for nonstationary, dynamic environments.

In this paper, we propose a novel, low-complexity solution for optimizing the performance of a delay-sensitive stream filtering application across a network of distributed classifiers. Our contributions are as follows.

- We configure each classifier by choosing an operating point to control its performance and throughput, as in [4]. Since the stream data is usually transmitted over a network infrastructure, we model the stream traffic at each classifier using Poisson models and introduce a utility metric for real-time stream processing applications that captures the tradeoffs between the throughput, classification accuracy, and end-to-end delay of filtered streams.
- We identify a key challenge in optimizing over an ensemble of classifiers. In particular, classifiers do not have sufficient information to jointly configure their operating points due to distributed analytics, since changing the operating point of one classifier will affect the distribution of input data to the next classifier in the sequence. To address this challenge, we introduce a method for decomposing the utility function into a set of locally observable metrics that can be calculated directly by each classifier. Each classifier can then exchange these metrics with other classifiers to compute the utility of the entire system for any fixed configuration of classifiers.
- We introduce a model-free, stochastic, multiagent learning algorithm for reconfiguring classifiers in a distributed fashion. This algorithm has very low complexity, and requires little coordination across different sites. We show that in stationary environments, the algorithm converges to an optimal or near optimal system configuration with high probability. Moreover, the fast convergence rate of the algorithm enables the system to adapt quickly to dynamic stream characteristics.

Note that in this paper we do not modify the underlying classification scheme, but rather focus on configuring operating points of individual classifiers in a fixed processing sequence. This allows the designed algorithms to be applicable to any available type of underlying classification algorithms (e.g., support vector machines, k-nearest neighbors, maximum likelihood etc.). The paper is organized as follows. Section II introduces the model used for classifiers and distributed stream processing systems, and derives a utility function for stream mining applications. Section III introduces estimation and modeling techniques for a chain of classifiers, and proposes a framework for distributed algorithms. Based upon this framework, a low complexity, model-free algorithm is proposed in Section IV. Section V extends the approach to a general networked topology of classifiers, and discusses issues involving multiple streams sharing the system. Section VI provides results of the performance of our algorithm in different system environments, and Section VII concludes the paper.

II. Distributed Stream Processing System Model

A. Characterizing Binary Classifiers and Classifier Chains

Consider the binary chain classifier topology shown in Fig. 2. Each binary classifier v_i processes an input stream by classifying each stream data object (SDO) as belonging to a positive class H_i, or a negative class $\bar{H}_i$. Each SDO that is classified as belonging to H_i is forwarded from a classifier node v_i to the next classifier node v_{i+1} in the chain or, otherwise, the SDO is dropped from the stream. SDOs generated by the source v_0 are only received at the terminal v_N if they are forwarded by every classifier in the chain (i.e., they are relevant for the application).

Given the ground truth X_i for an input SDO to classifier v_i, denote the classification decision on the SDO by $\hat{X}_i$. The proportion of correctly forwarded samples is captured by the *probability of detection* $P_i^D = \Pr\{\hat{X}_i \in \mathrm{H}_i \,|\, X_i \in \mathrm{H}_i\}$, and the proportion of incorrectly forwarded samples is captured by the *probability of false alarm* $P_i^F = \Pr\{\hat{X}_i \in \mathrm{H}_i | X_i \notin \mathrm{H}_i\}$.

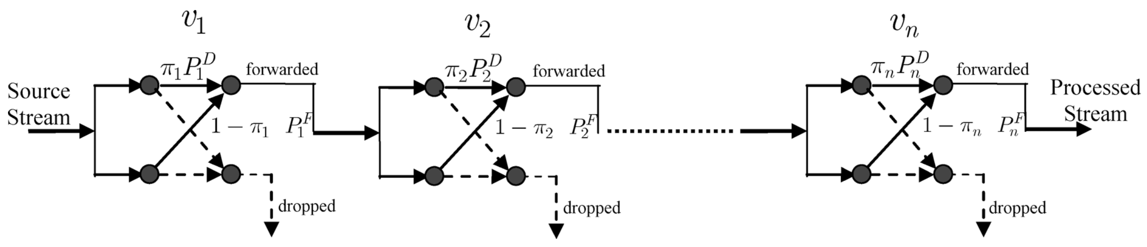


Fig. 2. Classifier chain with probabilities labeled on each edge.

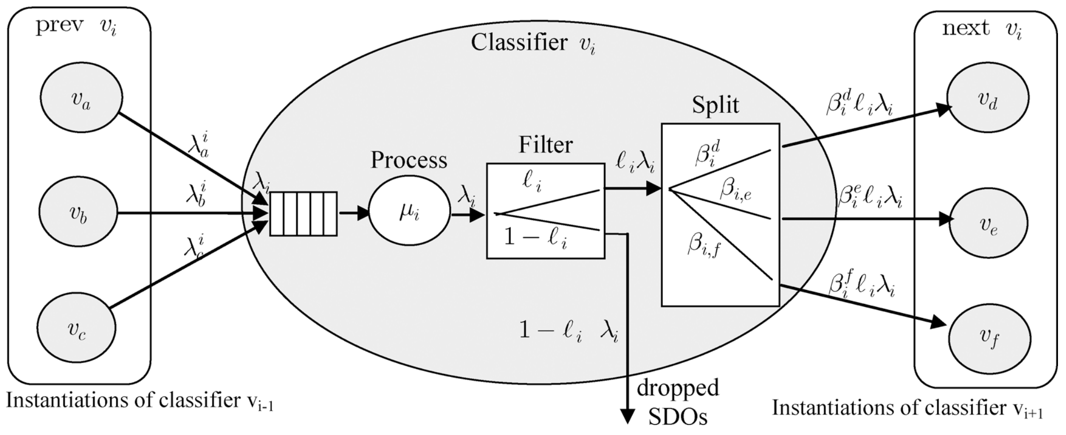


Fig. 3. Classifiers can be instantiated on multiple nodes and, therefore, each classifier is preceded and followed by multiple copies of the previous/next classifiers in the processing chain. The dataflow and queuing model is shown here.

Suppose that the input stream to classifier v_i has (conditional) *a priori probability* (APP) π_i of being positive. The probability of forwarding an SDO to the next classifier can be given by

$$\ell_i = \pi_i P_i^D + (1-\pi_i) P_i^F. \tag{1}$$

Moreover, the probability of *correctly forwarding* data to the next classifier is

$$\wp_i = \pi_i P_i^D. \tag{2}$$

A filtering classifier is often designed such that the probability of detection is maximized subject to a false alarm probability constraint [28], [29]. Hence, by varying the false alarm constraint, different probabilities of detection can be obtained, and different volumes of the stream can be forwarded. Assuming that each classifier operates at a fixed complexity level, a detection-error-tradeoff (DET) curve, or a curve relating P_i^D to P_i^F, can be obtained for each classifier v_i. Hence, given the APP π_i and the DET curve, ℓ_i and $\wp_i$ become deterministic functions of P_i^F.

B. Classifier System Model

In general, classifiers may be instantiated on multiple nodes with different resource constraints and processing power. To represent the dataflow and compute the end-to-end delay for each received SDO (an important metric for many applications [38]–[40]), each classifier v_i is modeled as a G/G/1 queuing facility, followed by stream filtering and splitting operations, where each SDO is discarded with probability $1-\ell_i$ or forwarded to one of several next-hop classifiers with probability ℓ_i (see Fig. 3). The total SDO input rate, and the average processing rate for classifier v_i, are denoted by λ_i and μ_i, respectively, and the set of previous-hop neighbors and next-hop neighbors of v_i as $\mathrm{prev}(v_i)$ and $\mathrm{next}(v_i)$, respectively. We indicate the corresponding subscripts of nodes in $\mathrm{prev}(v_i)$ and $\mathrm{next}(v_i)$ as prev(i) and next(i). Moreover, define the ancestors of v_i and descendants of v_i, $\mathrm{anc}(i)$ and $\mathrm{des}(i)$, as the nodes which have a path to v_i, or to which v_i has a path, respectively. If v_i has multiple previous-hop neighbors $v_x \in \mathrm{prev}(i)$, we denote the arrival rate from each neighbor as λ_x^i, with $\sum_{x\in\mathrm{prev}(i)} \lambda_x^i = \lambda_i$. When an SDO is forwarded, it is sent to the next-hop neighbor v_y, $v_y \in \mathrm{next}(i)$, with path selection probability β_i^y configured by the classifier v_i. (If we require that each SDO is forwarded to only one next-hop neighbor, then $\sum_{y\in\mathrm{next}(i)} \beta_i^y = 1$.) Therefore, the resulting arrival rate at the next-hop neighbor v_y from v_i is $\lambda_i^y = \beta_i^y \ell_i \lambda_i$.

C. Objective Function for a Stream Processing Application

The goal of a stream processing application is not only to maximize the amount of processed data (the *throughput*), but also the amount of data that is correctly processed and received (the *goodput*). However, increasing the volume of data processed also leads to an increased load on the system, which increases the end-to-end delay for the stream and can lead to extra misses. To capture the tradeoff between system performance and the incurred delay, we construct a stream utility function to be maximized by the system.

System Performance Metric: In prior works, the performance of stream mining applications is evaluated based upon the probability of misclassifying input data. Depending upon the importance of true and false positives [37], the performance of the chain of classifiers can be specified by a cost associated with misses and false alarms. For a simple chain of classifiers labeled from 1 to n, where there is no splitting of streams after the filtering process, the end-to-end cost per SDO can be given by

$$\begin{aligned} C &= (\pi - \wp) + \theta(\ell - \wp) \\ &= \pi - \prod_{i=1}^{n} \wp_i + \theta\left(\prod_{i=1}^{n} \ell_i - \prod_{i=1}^{n} \wp_i\right) \end{aligned} \tag{3}$$

where π indicates the true APP of input data that is returned by the entire chain, and θ specifies the weight of false positives relative to true positives. Since π depends only upon the stream characteristics, we can regard it as constant and remove it from the cost function, invert it, and produce a utility function $F = \prod_{i=1}^{n} \wp_i - \theta(\prod_{i=1}^{n} \ell_i - \prod_{i=1}^{n} \wp_i)$.

Note that $\prod_{i=1}^{n} \ell_i$ is simply the total fraction of stream data forwarded across the entire chain. $\prod_{i=1}^{n} \wp_i = \prod_{i=1}^{n} \pi_i P_i^D$, on the other hand, is the fraction of data out of the entire stream that is correctly forwarded across the entire chain, which is calculated by the probability of detection by each classifier, times the conditional APP of positive data at the input of each classifier v_i. For example, in Fig. 1(b), the conditional probability π_2 refers to the probability that the probability that input data correspond to volleyball images, conditioned on the outdoor filtering process by the first classifier.

Delay Penalty: Because many multimedia applications are delay sensitive, we consider a function $H(D)$ to capture the loss of utility as a function of the processing delay of D [39], [40]. For example, embedded and multirate systems often have a hard delay deadline Δ before the next interval [41]. The fraction of data that is useful to the system would be given by $H(D) = \Pr\{D \leq \Delta\}$, which is the probability that the delay on a processed data object meets the hard deadline. For such an application, any data that is retrieved after the delay deadline can be regarded as a miss (as it is dropped or no longer useful), while false alarm data that is retrieved after the deadline is ignored (regarded as a true negative). Note that in practice, this probability can be estimated analytically if an accurate model (e.g., M/M/1) for the arrival and service times are used, or else it can be obtained by time stamping the processed SDO packets. This effectively leads to a single objective function $F \cdot H(D)$, which satisfies the concept of fairness between accuracy and delay implemented by the Nash product [49].[1] Hence, the system attempts to configure the false alarm probabilities of each classifier v_i, P_i^F, and the path selection parameters $\beta_{i,y}$, $y \in \text{next}(i)$, to maximize the average utility as follows:

$$\begin{aligned} \max_{\mathbf{P}^F, \mathbf{B}} Q(\mathbf{P}^F, \mathbf{B}) = \sum_{\vartheta} \beta_\vartheta H(D_\vartheta) \\ \times \left(\prod_{i \in \vartheta} \wp_i - \theta\left(\prod_{i \in \vartheta} \ell_i - \prod_{i \in \vartheta} \wp_i\right)\right) \\ \text{s.t.} \quad \mathbf{0} \leq \mathbf{P}^F \leq \mathbf{1} \\ \sum_{\vartheta} \beta_\vartheta = 1, \beta_\vartheta \geq 0 \end{aligned} \tag{4}$$

[1]The generalized Nash product provides a tradeoff between misclassification cost [37], [17] and delay depending upon the exponent attached to each term F^α and $H(D)^{(1-\alpha)}$, respectively. In practice, we observed through simulations that for the considered applications, an equal weight $\alpha = 0.5$ provided the best tradeoff between classification accuracy and delay.

where $\mathbf{P}^F = (P_1^F, \ldots, P_N^F)$ is the vector of false alarm probabilities for all N classifiers in the system, $\mathbf{B}$ is a vector of all path selection probabilities β_i^y for classifiers v_i and a next-hop neighbors v_y, β_ϑ denotes the probability that an SDO is scheduled to take an end-to-end path ϑ (or to be filtered by the respective classifier chain given by ϑ), $i \in \vartheta$ indicates that classifier v_i is a node along path ϑ, and D_ϑ the end-to-end delay across that path. Note that the condition $\lambda_i \leq \mu_i$ must hold in order for the average delay to be finite; otherwise the utility is always 0. The overall utility is the sum of path utilities weighted by the probability of choosing that path. Note that the parameters *configured* by each classifier to optimize (4) include both its own false alarm probability, and its next-hop path selection probabilities. In the subsequent sections of this paper, we will represent these parameters as *actions* that are taken by each classifier over repeated, distributed interactions [42].

D. Discussion of Stream Utility Maximization

Before proposing any solutions, a fundamental question is whether all the necessary information can be gathered to solve the utility maximization problem in (4). In particular, the objective function relies on obtaining the terms $\wp_i$, ℓ_i, and the end-to-end delay D, which are all functions of the conditional APP π_i, the false alarm configuration P_i^F, and the detection curve P_i^D across different classifiers. While classifiers may be willing to provide information about P_i^F and P_i^D, note that the conditional APP π_i at every classifier v_i is, in general, a complicated function of the false alarm configurations of all previous classifiers, i.e., $\pi_i = \pi_i(\mathbf{P}_x^F)_{x \in \text{anc}(i)}$. Importantly, $\pi \neq \prod_{i=1}^{n} \pi_i$, since setting different thresholds for the false alarm probabilities at previous classifiers will also affect the ground truth input data distribution to classifier v_i. (Note that this differs from prior work [16], [17] in that the exclusivity assumption makes π_i purely a function of the stream characteristics, and not the configurations of previous classifiers.) As discussed before, if the analytics are not shared between classifiers, and if constructing a joint classification model by inference requires intolerable time and complexity, then the objective function $Q(\mathbf{P}^F, \mathbf{B})$ cannot be computed directly.

To address this challenge, in the rest of this paper, we provide a low complexity, distributed approach for obtaining the utility for fixed configurations based upon this informational constraint, and construct algorithms for maximizing the utility. Our distributed algorithm is motivated by limiting the amount of *information* that needs to be exchanged between classifiers, lowering *computational complexity*, and improving the *convergence/dynamic adaptation rate*. Ultimately, we show that by periodically exchanging small amounts of information throughout the system, an algorithm with very low complexity and fast convergence rate can be constructed. We then provide simulations to verify the proposed solution in Section VI.

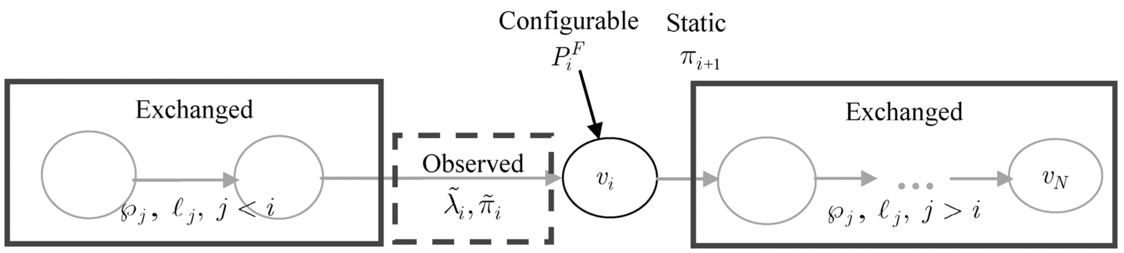


Fig. 4. Various parameters in relation to v_i.

TABLE I
SUMMARY OF PARAMETER TYPES AND A FEW EXAMPLES

Type of parameter for v_i :	Description:	Examples:
Static Parameters:	Fixed parameters, exchanged during initialization	π
Observed Parameters:	Can be measured/stamped by the last classifier v_n	D
Exchanged Parameters:	Traded with other classifiers	$\wp_i,\ \ell_i$
Configurable Parameters:	Configured by classifier v_i	P_i^F

III. Distributed Framework for a Single-Path Topology

A. Distributed Utility Estimation Approach

In the single-path binary classifier chain topology (Fig. 2), the stream utility maximization problem can be simplified as follows:

$$\begin{aligned}\max_{\mathbf{P}^F \forall v_i \in V} Q(\mathbf{P}^F) &= \max_{\mathbf{P}^F} C(\vartheta)\\ \text{s.t.}\quad & \mathbf{0} \le \mathbf{P}^F \le \mathbf{1}\\ & \vartheta = \{1,\ldots,n\}. \end{aligned} \tag{5}$$

While the utility cannot be explicitly formulated as a function of $\mathbf{P}^F$ due to informational constraints, the utility at any given time instant can still be estimated as follows: each of these parameters in (5) can be easily obtained by at least one classifier, as displayed in Fig. 4. First, the goodput and throughput ratios $\wp_i$ and ℓ_i are functions of both the configuration P_i^F and the (conditional) APP π_i. The configuration is set by the classifier and therefore known. The only unknown element, the APP, can still be estimated from the input stream, as will be shown in the following subsection. Finally, $H(D)$ can be determined by the final classifier in the chain by time stamping data objects. Hence, for *every parameter* in (5), there exists a classifier that can easily estimate the value of that parameter based upon locally observable data and its own configuration. By exchanging these locally obtained parameters and configurations across all classifiers, each classifier can then estimate the overall cost for that given time instant. Table I lists the various parameter types, their descriptions, and specific examples in our problem.

B. Estimating Parameters in Stationary and Nonstationary Environments

We now discuss several low complexity methods to update observed parameters. While parameters such as the arrival rate λ_i can be directly measured from the incoming SDOs, parameters such as the APP π_i are not directly observable and must be *estimated*. For example, since the classifier function is fixed, a lookup table can be precomputed for discrete data values, and the APP extrapolated based upon the actual value of the input data. Another method could be to use a model. For example, Gaussian mixture models (GMMs) have been shown to be accurate for classifying color and texture features in images [26]. Suppose that for classifier v_i, the training data set obeys a Gaussian model, with a random variable $\mathcal{N}(d,\sigma_1)$ representing the distribution for positive SDOs, and $\mathcal{N}(-d,\sigma_2)$ representing the distribution for negative SDOs. The *maximum a posteriori* (MAP) estimation can be used to determine the APP for each SDO observation o_i

$$\tilde{\pi}_i[o_i] = \frac{\operatorname{erfc}\left(\frac{d-o_i}{\sigma_1\sqrt{2}}\right)}{\operatorname{erfc}\left(\frac{d-o_i}{\sigma_1\sqrt{2}}\right) + \operatorname{erfc}\left(\frac{-o_i-d}{\sigma_2\sqrt{2}}\right)}. \tag{6}$$

Note that in order to perform MAP estimation, an operation similar in complexity to classification must be performed. This operation can be relatively cheap or expensive depending upon the complexity feature extraction, which needs only be performed once per image prior to classification and/or APP estimation. In the case where classification is nonnegligible compared to feature extraction, it would be useful to limit the frequency at which the APP is computed. For example, the workload can be mitigated by randomly computing the APP for only 10% of the images that are classified.

Because the APP prediction for a single SDO does not accurately capture the *average* APP over a time period (or *control interval*), the estimated APP must be averaged over all observations in the control interval, or possibly over multiple intervals when the arrival rate is small. For simplicity, we normalize the control intervals to be $[0,1), [1,2), [2,3), \ldots$, such that the beginning of each interval corresponds to an "iteration." During each iteration, we allow each classifier to adapt its configuration based upon updated information over the prior control interval.

A simple approach to update the estimated APP is to average all observed values within a time interval, i.e., if N APPs are computed within the control interval $[t-1,t)$, then the estimated APP is

$$\tilde{\pi}_i[\text{obs}[t-1,t)] = \frac{1}{N}\sum_{n=1}^{N}\tilde{\pi}_i[o_i(\text{n})] \tag{7}$$

where $o_i(n)$ indicates the value of the nth arriving SDO. In order to update the estimated APP for iteration t, $\tilde{\pi}_i(t)$, over a history of observations, we choose a sequence of values α_t, $0 \leq \alpha_t \leq 1$, and update $\tilde{\pi}_i(t)$ recursively

$$\tilde{\pi}_i(t) := \alpha_t\tilde{\pi}_i[\text{obs}([t-1,t))] + (1-\alpha_t)\tilde{\pi}_i(t-1). \tag{8}$$

For example, in a stationary environment where the APPs are fixed, a time-averaging value for $\tilde{\pi}_i(t)$ converges to the expectation $E[\tilde{\pi}_i]$ as $t \to \infty$ and can be given by $\alpha_t = 1/t$. In a nonstationary environment, a discounted average can be utilized as in [34]. The discounted average weighs the observations of the interval associated with the current iteration t using a unit weight, while observations from past iterations t' are scaled by $e^{-\varphi(t-t')}$, where φ is inversely proportional to the amount of average "memory" allocated to prior observations, i.e., φ should be increased in more dynamic environments in order to update the APP more frequently. The discounted estimation of the APP is given by

$$\begin{aligned}\tilde{\pi}_i(t) &\approx (1-e^{-\varphi})\Bigg(\tilde{\pi}_i[\text{obs}(t)] \\ &\quad + \sum_{t'=0}^{t-1} e^{-\varphi(t-t')}\cdot\tilde{\pi}_i[\text{obs}(t')]\Bigg) \\ &\approx (1-e^{-\varphi})\tilde{\pi}_i[\text{obs}(t)] + e^{-\varphi}\tilde{\pi}_i(t-1). \end{aligned}\tag{9}$$

Note that regardless of the choice of values α_t, efficient implementations of (7) and (8), which are simple add-multiply operations, require only a constant space complexity of a couple scalars. Hence, updating observed parameters, and calculating the local metrics during each iteration requires negligible complexity.

C. Summary of the Proposed Distributed Framework

Next, we summarize our proposed algorithmic framework for estimating the overall cost and reconfiguring the classifiers.

Distributed Framework for Maximizing Utility

1) Initialize the configuration $P_i^F(0)$ and exchange static parameters.

Each classifier v_i sets $P_i^F(0)$ to an initial configuration.

2) For each iteration (or integer time) t, measure/estimate the arrival rate $\tilde{\lambda}_i(t)$ from the last elapsed control interval $[t-1,t)$.

The arrival rate from the previous-hop, $\tilde{\lambda}_i(t)$, is obtained by observing either the number of arrivals at classifier v_i during the interval $[t-1,t)$, or forming an estimate based upon time-averaging, discounting, etc. of previous measurements, as discussed in Section III-B.

3) Estimate the APP via MAP $\tilde{\pi}_i(t)$ from past time intervals.

Based upon the location of the *a posteriori* points of arriving SDOs during time interval $[t-1,t)$, find the APP of each SDO and update $\tilde{\pi}_i(t)$ based upon time-averaging, discounting, etc. (see Section III-B.)

4) For each classifier $v_i, v_j, j \neq i$, exchange local metrics $\wp_i$ and ℓ_i with all other classifiers just prior to time t to obtain an estimate of the $\tilde{Q}^t$.

5) Based upon empirical analysis, modeling, experimentation, etc., choose the best configuration $P_i^F(t)$ to maximize the following:

$$P_i^F(t) = \arg\max_{P_i^F} E[\tilde{Q}^{t+1}(\mathbf{P}^F)] \tag{10}$$

where (10) is the *predicted* stream utility at time $t+1$. Since the $P_i^F(t)$ affects the performance in the following interval $[t,t+1)$, solving (10) gives the (predicted) optimal configuration.

Note that in this framework, we did not specify here how parameters are predicted for the next time interval (i.e., step 5), as different prediction schemes can be used. For example, in [15], a Markov model was used to predict feature values (e.g., APPs) of SDOs based upon short-term temporal correlations. However, modeling stream behavior requires building, estimating, and updating unknown parameters, which increases both the complexity and the convergence time. Moreover, because we allow other classifiers to change configurations during each iteration, the resulting APPs and arrival rates often behave in nonstationary manners. Hence, a predictive Markov model is not well suited for the problem that we are studying.

IV. Distributed Learning Algorithm for the Single-Path Topology

A. Safe Experimentation for Discrete Configuration Sets

In this section, we introduce a low-complexity, *model-free* learning approach called *safe experimentation* [44] for choosing the best configuration for classifiers (i.e., Step 5 of the distributed framework). Safe experimentation was first proposed for large, distributed, multiagent systems, where each agent is unable to observe the actions of all other agents (due to informational or complexity constraints) and, hence, cannot build a model of other agents. The agent therefore adheres to a "trusted" action at most times, but occasionally "explores" a different action in search of a potentially better action. Essentially, safe experimentation does not require coordinating actions between agents, or in our case, between autonomous classifier sites.

Our stream processing system falls under such a category and is equivalent to a common interest game [45], where distributed classifiers (i.e., agents) want to configure themselves (i.e., perform actions) to maximize the same utility function. The safe experimentation algorithm for reconfiguring classifiers is given as follows.

1) Initialization: At iteration $t=0$, each classifier randomly selects a configuration $P_i^F(0)$ from a *discrete* action set A_i, which is set as the baseline configuration $P_i^{F,b}(1)$. After exchanging information about the derived local utilities from the initial configurations, each classifier's baseline utility at iteration 1 is initialized as $u^b(1) = Q(\mathbf{P}^F(0))$.

2) Configuration Selection: At each subsequent iteration, each classifier selects his baseline configuration with probability $(1-\varepsilon_t)$ or experiments with a new random configuration with probability ε_t. Hence, $P_i^F(t) = P_i^{F,b}(t)$ with probability $(1-\varepsilon_t)$, and $P_i^F(t)$ is chosen uniformly over A_i with probability ε_t. ε_t is denoted the *exploration rate* at iteration t.

3) Baseline Configuration and Baseline Utility Update: Each classifier compares the utility received, $Q(\mathbf{P}^F(t))$, with his baseline utility $u^b(t)$, and updates his baseline configuration and utility as follows:

$$P_i^{F,b}(t+1) = \begin{cases} P_i^F(t), & Q(\mathbf{P}^F(t)) > u_i^b(t) \\ P_i^{F,b}(t), & Q(\mathbf{P}^F(t)) \le u_i^b(t) \end{cases} \quad (11)$$

$$u_i^b(t+1) = \max\left(u_i^b(t), Q(\mathbf{P}^F(t))\right). \quad (12)$$

4) Return to Step 2 and Repeat: The reason why this learning algorithm is called "safe experimentation" is because the baseline utility is nondecreasing with respect to time (or the number of iterations) and, hence, the performance of the algorithm only improves over time. We now provide a sufficient condition for finding the optimal solution.

Theorem 1: The following two conditions for the exploration rate ε_t are sufficient to guarantee that the safe experimentation algorithm for common interest games converges to the global optimal solution with probability 1

$$\lim_{t\to\infty} \varepsilon_t = 0 \quad (13)$$

$$\lim_{t\to\infty} \prod_{\tau=1}^{t} \left[1 - \left(\frac{\varepsilon_\tau}{|A_1|}\right)\left(\frac{\varepsilon_\tau}{|A_2|}\right)\cdots\left(\frac{\varepsilon_\tau}{|A_n|}\right)\right] = 0. \quad (14)$$

Proof: The proof is similar to the proof for Theorem 3.1 in [44]. We provide a short sketch of the proof to highlight properties of the exploration rate. First, the exploration rate must converge to 0 as $t \to \infty$, as indicated by (13), such that the algorithm will play its baseline configuration with probability 1. Moreover, note that each multiplicative term in (14) represents the probability that the joint configuration played at time τ is *not* the optimal joint configuration, *or* all classifiers experimented at time τ. Hence, the left-hand side of (14) forms an upper bound on the probability that the optimal joint configuration is *not* played before time t. Thus, (14) provides a sufficient condition for the optimal joint configuration to be eventually played with probability 1. ■

Note that we have shown (14) to be a *sufficient* condition on the exploration rate for convergence to an optimal solution, but only for a *discrete* action set. Moreover, the proof provides no bounds for the convergence time of safe experimentation. In general, the method converges very slowly, and the expected time for finding the optimal solution can be bounded below by $|\mathcal{A}| = |A_1||A_2|\ldots|A_n|$, which is the expected time for finding the optimal solution via i.i.d. uniform sampling (i.e., $\varepsilon_t = 1, \forall t$) of the action set. Because the action set for each classifier can be large when configurations are finely quantized, safe experimentation becomes impractical for dynamic environments where good configurations are promptly needed. To address this limitation, in the next section, we will provide an alternative to safe experimentation for continuous configurations.

B. Combining Safe Experimentation With Randomized Local Search

To reduce the convergence time as compared to the discrete safe experimentation algorithm, we propose a simple stochastic algorithm that fits within the framework of two-phase methods highlighted in [46]. In this approach, we combine a uniform random search for a baseline configuration over a continuous feasible set, and we perform a *randomized* local search algorithm around the baseline configuration. Unlike pure safe experimentation, which uses random sampling over the entire configuration space, the local search procedure take advantage of smoothness and continuity properties that exist in the global utility function, thereby allowing classifiers to converge to locally optimal points near their baseline configurations. The algorithm is given by modifying step 2 of safe experimentation as follows:

Modified 2) Configuration Selection: At each subsequent iteration, each classifier selects its baseline configuration with probability $(1-\varepsilon_t)$ or experiments with a new random configuration with probability ε_t. If the baseline configuration is selected, it is perturbed by a small random variable (e.g., Gaussian, uniform, etc.) $Z_i(t)$. Hence, $P_i^F(t)$ is chosen uniformly over the feasible configuration space with probability ε_t, and$P_i^F(t) = P_i^{F,b}(t) + Z_i(t)$ with probability $(1-\varepsilon_t)$, where $Z_i(t)$ is a zero-mean random variable, with $\lim_{t\to\infty} Z_i(t) = 0$.

If the size of the local search random perturbations do not decay too quickly (e.g., $E[|Z_i(t)|^2] = K/t^2$, where $K > 0$ is a constant), it can be shown that (in a stationary setting) the local search algorithm converges to a local maximum with probability 1 [47]. Moreover, if the exploration rate is sufficiently high, the algorithm will converge to the globally optimal point with probability 1.

C. Summary of Algorithms, and Nonstationary Dynamics

In Table II, a summary comparing the different algorithms is provided. For the combined safe experimentation and local search algorithm, it is impossible to provide useful sufficient conditions to guarantee both *fast and sure* convergence in a dynamic environment to a global maxima for arbitrary utility functions. However, it is useful to have a very high exploration rate at the beginning of the algorithm, such that a good baseline configuration can be found as a starting point for local search. During later iterations, a very low exploration rate is preferable, such that the local search algorithm can perform "refined" exploration around a good baseline point until stream characteristics change again.

V. Distributed Algorithms for a Multiple-Path Topology

Stream processing engines are generally distributed, not only due to the localized (or private) nature of data, but also because distributed systems are more robust toward changes in system load and nodal failures [19], [25]. We will now consider distributed systems where several different nodes may perform the same type of classification function. We refer to these classifiers as *siblings* and use $\text{sib}(i)$ to denote the set of all subscripts of classifiers that are siblings of v_i. Note that due to the distributed nature of the system, not all siblings may be reachable from the

TABLE II
SUMMARY COMPARING THE DIFFERENT ALGORITHMS AND THE VARIOUS CRITERIA

Algorithm	*Information Overhead*	*Complexity*	*Convergence/Dynamics*
Discretized Safe Experimentation	Global exchange of local utilities	Very low (random configuration, constant memory usage)	Slow, suboptimal
Continuous Safe Experimentation with Randomized Local Search	Global exchange of local utilities	Very low (random configuration, constant memory usage)	Fast; optimality can vary depending on experimentation frequency and decay rate of local search

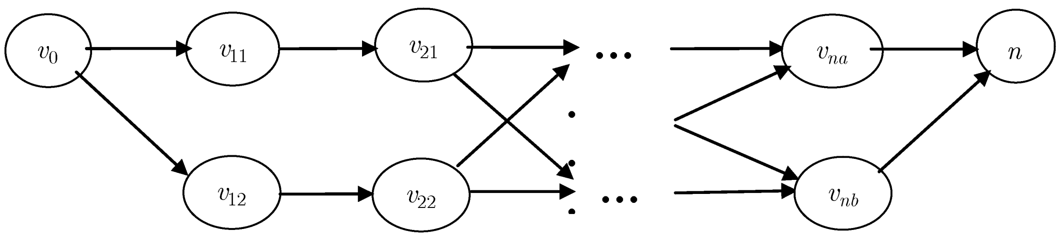


Fig. 5. Example of a classifier trellis for the single-stream, multiple-path model.

same previous-hop neighbors. The paths that the stream may take through the distributed system can therefore be represented by a trellis, as in Fig. 5.

A. Distributed Framework for Global Utility Maximization in Multiple-Path Topologies

In this subsection, we extend the distributed framework for maximizing utility (Section III-C) to multiple-path topologies. Here, we decompose the multiple-path stream utility function in (4) into partial utilities of ancestors and descendants of each hop in the processing topology. For a classifier v_i, the hop-wise decomposition can be given by

$$\begin{aligned} Q(\mathbf{P}^F,\mathbf{B}) = H(D)[(1+\theta) \\ \times \sum_{j\in\mathbf{sib}(i)\cup\{i\}} \tilde{G}_0^j\left(\mathbf{P}_{0,j}^F,\mathbf{B}_0^j\right) \\ \times \sum_{y\in\text{next}(j)} \beta_j^y \tilde{G}_y^N\left(\mathbf{P}_{y,N}^F,\mathbf{B}_y^N\right) \\ -\theta \sum_{j\in\mathbf{sib}(i)\cup\{i\}} \tilde{T}_0^j\left(\mathbf{P}_{0,j}^F,\mathbf{B}_0^j\right) \\ \times \sum_{y\in\text{next}(j)} \beta_j^y \tilde{T}_y^N\left(\mathbf{P}_{y,N}^F,\mathbf{B}_y^N\right) \end{aligned} \quad (15)$$

where

$$\tilde{G}_0^j\left(\mathbf{P}_{0,j}^F,\mathbf{B}_0^j\right) = (\tilde{\wp}_j) \sum_{\vartheta_{\text{anc}(j)}} \beta_{\vartheta_{anc(j)}} \left(\prod_{x\in\vartheta_{\text{anc}(j)}} \tilde{\wp}_x\right)$$
$$\tilde{T}_0^j\left(\mathbf{P}_{0,j}^F,\mathbf{B}_0^j\right) = (\tilde{\ell}_j) \sum_{\vartheta_{\text{anc}(j)}} \beta_{\vartheta_{\text{anc}(j)}} \left(\prod_{x\in\vartheta_{\text{anc}(j)}} \tilde{\ell}_x\right) \quad (16)$$

indicates the measured goodput and throughput factors across all paths $\vartheta_{\text{anc}(j)}$ from the source v_0 through node v_j, and

$$\tilde{G}_j^N\left(\mathbf{P}_{j,N}^F,\mathbf{B}_j^N\right) = (\tilde{\wp}_j) \sum_{\vartheta_{\text{dec}(j)}} \beta_{\vartheta_{\text{dec}(j)}} \prod_{z\in\vartheta_{\text{dec}(j)}} \tilde{\wp}_z$$
$$\tilde{T}_j^N\left(\mathbf{P}_{0,j}^F,\mathbf{B}_0^j\right) = (\tilde{\ell}_j) \sum_{\vartheta_{\text{dec}(j)}} \beta_{\vartheta_{\text{dec}(j)}} \left(\prod_{z\in\vartheta_{\text{dec}(j)}} \tilde{\ell}_z\right) \quad (17)$$

indicates the measured goodput and throughput factors of all paths from v_j to the destination v_N. Note that the only part of (15) that is directly known by classifier v_i is $(\tilde{\wp}_i - \theta_i(\tilde{\ell}_i - \tilde{\wp}_i))$, which is part of Q_0^i. (Classifier v_i cannot predict the delay of its next-hop neighbors, since the load is affected by its siblings.) Hence, to construct (16) and (17), *the local metrics and path selection probabilities for each classifier must be exchanged with every other classifier*. The total information exchange overhead is $O(N^2)$, where N is the total number of classifiers in the system.

In order to reduce the informational overhead, it is possible to exploit the trellis structure of the networked system by multiplying accumulated path selection probabilities and local utility parameters, and propagating the result down each path. Both a forwards and backwards propagation of $O(dn)$ overhead, where d is the average in-degree/out-degree of each classifier node, are needed to distribute the information to each classifier. The recursive formula for forward propagation of $\tilde{Q}_0^i(\mathbf{P}_{0,i}^F,\mathbf{B}_0^i)$ is

$$\tilde{G}_0^j\left(\mathbf{P}_{0,j}^F,\mathbf{B}_0^j\right) = (\tilde{\wp}_j) \sum_{x\in\text{prev}(i)} \beta_{x,i}\tilde{G}_0^x\left(\mathbf{P}_{0,x}^F,\mathbf{B}_0^x\right)$$
$$\tilde{T}_0^j\left(\mathbf{P}_{0,j}^F,\mathbf{B}_0^j\right) = (\tilde{\ell}_j) \sum_{\vartheta_{\text{anc}(j)}} \beta_{\vartheta_{\text{anc}(j)}}\tilde{T}_0^x\left(\mathbf{P}_{0,x}^F,\mathbf{B}_0^x\right) \quad (18)$$

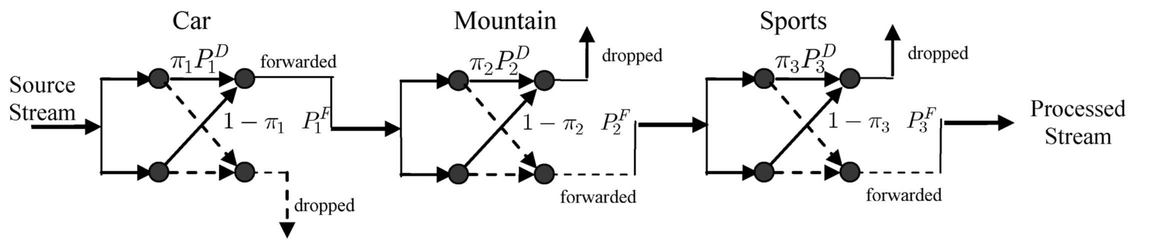


Fig. 6. Chain of classifiers for car images that do not include mountains, nor are related to sports.

and the recursive formula for backward propagation of $\tilde{Q}_i^N(\mathbf{P}_{i,N}^F, \mathbf{B}_i^N)$ is:

$$\tilde{G}_i^N\left(\mathbf{P}_{i,N}^F, \mathbf{B}_i^N\right) = (\tilde{\wp}_i) \sum_{y \in \mathrm{next}(i)} \beta_{i,y} \tilde{G}_y^N\left(\mathbf{P}_{y,N}^F, \mathbf{B}_y^N\right)$$
$$\tilde{T}_i^N\left(\mathbf{P}_{i,N}^F, \mathbf{B}_i^N\right) = (\tilde{\ell}_i) \sum_{y \in \mathrm{next}(i)} \beta_{i,y} \tilde{T}_y^N\left(\mathbf{P}_{y,N}^F, \mathbf{B}_y^N\right). \tag{19}$$

Multiplying corresponding terms of (18) and (19) together (and dividing by the redundant term) gives the average goodput and throughput ratio of SDOs passing through classifier v_i, but not its siblings. (Note that from (15), the sum of $\tilde{Q}^j(\mathbf{P}^F, \mathbf{B}_i^N)$ over all $j \in \{i\} \cup \mathrm{sib(i)}$ is the global utility $\tilde{Q}(\mathbf{P}^F, \mathbf{B})$.) Hence, while the reduced information exchange does not enable classifier v_i to capture the state of the entire system, it captures information about the path directly affected by (or directly affecting) classifier v_i using low informational overhead.

Based upon this information exchange, the safe experimentation algorithm with local search may be performed by each classifier v_i, which requires optimization over both the false alarm configuration P_i^F and the path selection probabilities $(\beta_{i,y})_{y \in \mathrm{next}(i)}$. The performance of this algorithm will be discussed in more detail in the simulations section.

B. Handling Multiple Streams and Multiple Paths

In this section, we briefly discuss how to extend the proposed solution to a system with multiple streams. First, an appropriate objective function must be constructed. For example, should the system maximize the sum of utilities, or should it employ a fairness scheme? While the system objective function is ultimately the system designer's choice and is beyond the scope of this paper, to highlight possible approaches, we provide two exemplary utility functions. First, consider maximizing a social welfare function, defined by a (weighted) sum of utilities [48]. For simplicity, assume that each stream requires the system to perform the same type of classification. The maximization problem may be given by

$$\max_{\mathbf{P}^F, \mathbf{B}} \sum_{m=1}^{M} Q^{(m)}(\mathbf{P}^F, \mathbf{B})$$
$$\text{s.t.} \quad \mathbf{0} \le \mathbf{P}^F \le \mathbf{1}$$
$$\sum_{\vartheta^{(m)}} \beta_{\vartheta^{(m)}} = 1, \; \beta_{\vartheta^{(m)}} \ge 0 \tag{20}$$

where $\lambda_0^{(m)}$ is the rate of SDO generation from the source of stream m, ϑ_m is a path taken by an SDO of stream m, $\beta_{\vartheta^{(m)}}$ is the probability that an SDO arriving at the terminal of stream m took path ϑ_m, and $\wp_i^{(m)}$, $\theta_i^{(m)}$, $\ell_i^{(m)}$, $\varphi^{(m)}$ are the parameters and values associated with stream m (and classifier v_i). Because maximizing the sum of utilities may be unfair, since streams that gain little utility per resource will also be allocated little resources, we may also define a fairness objective function that maximizes a (weighted) product of utilities [49], i.e.,

$$\max_{\mathbf{P}^F, \mathbf{B}} \prod_{m=1}^{M} Q^{(m)}(\mathbf{P}^F, \mathbf{B})$$
$$\text{s.t.} \quad \mathbf{0} \le \mathbf{P}^F \le \mathbf{1}$$
$$\sum_{\vartheta} \beta_{\vartheta} = 1, \beta_{\vartheta} \ge 0. \tag{21}$$

In the same way as for the single stream case, by trading information across classifiers about the local utilities derived for each stream, one can apply the same distributed learning algorithm to configure the system and optimize the global objective.

VI. Simulation Results

A. Application: Classification of TV Video Data

Our proposed algorithm is tested using classifiers and videos provided by IBM's TRECVID 2007 project (see [27] for more details.). By extracting features such as color histogram, color correlogram, and co-occurrence texture, the classifiers are trained to detect high-level features, such as whether the video shot takes place outdoors, or in an office building, or whether there is an animal or a car in the video. The classifiers are SVM-based and can therefore dynamically set detection thresholds for the output scores for each image without changing the underlying implementation. Based upon the set of video images used, we chose to construct a chain out of classifiers to detect images that contained cars, but did not include mountains and were not sports related, as this includes a sizable fraction of images from the total set (113 images from a total of 18000), and also requires heavy filtering of images at each classifier. The arrangement of classifiers is shown in Fig. 6.

We considered two system scenarios for comparing the results of our algorithms: a) when resources are sufficient and the application is not delay sensitive, and b) when resources are scarce (the available resources constitute approximately 1/10 of the resource required by the classifiers), and application delay deadline is fixed at 10 s. (This delay deadline can be varied depending upon the system/application context [40], [41].) Note that the delay can be obtained by time stamping the processed data objects, but to save time in our simulations, we computed the delay analytically using an M/M/1 model, which is highly relevant to network traffic modeling [32], [35]. Hence, data objects arrive with exponential interarrival times, and classifiers

TABLE III
DETECTION AND FALSE ALARM TRADEOFF FOR THE ENTIRE CHAIN AFTER GLOBAL CONVERGENCE OF ALGORITHMS

	Safe Exp	Safe Exp with local search	Optimal classifier configuration with "random load shedding"
High resources (pd, pf)	0.7742, 0.3126	0.7742, 0.3096	0.7742, 0.3096
Low resources (pd, pf)	0.1129, 0.0050	0.1129, 0.0048	0.0060, 0.0025
Avg delay (no deadline, 10 sec deadline)	Inf, 5.4415	Inf, 5.0701	6.06

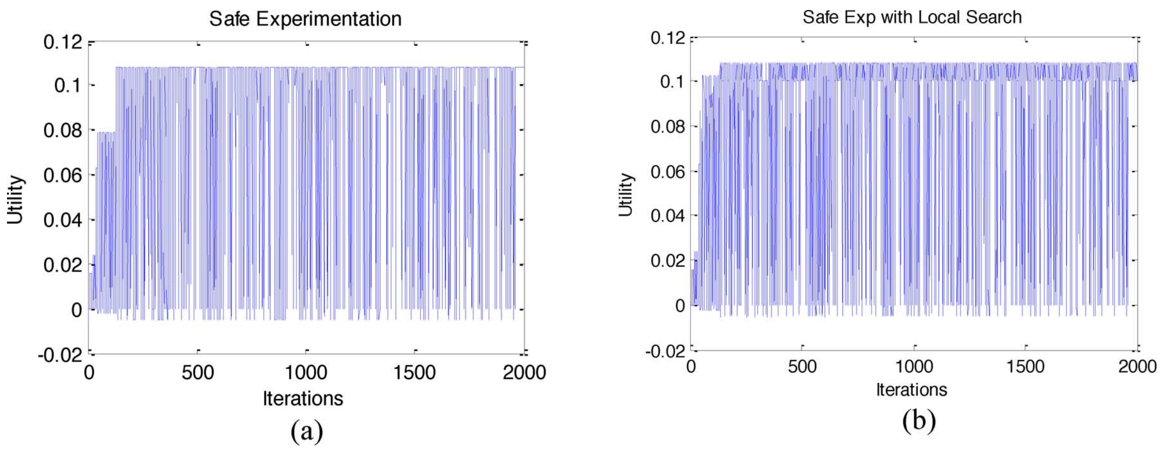


Fig. 7. Evolution of utility without delay penalty for (a) safe experimentation without local search, and (b) safe experimentation with local search.

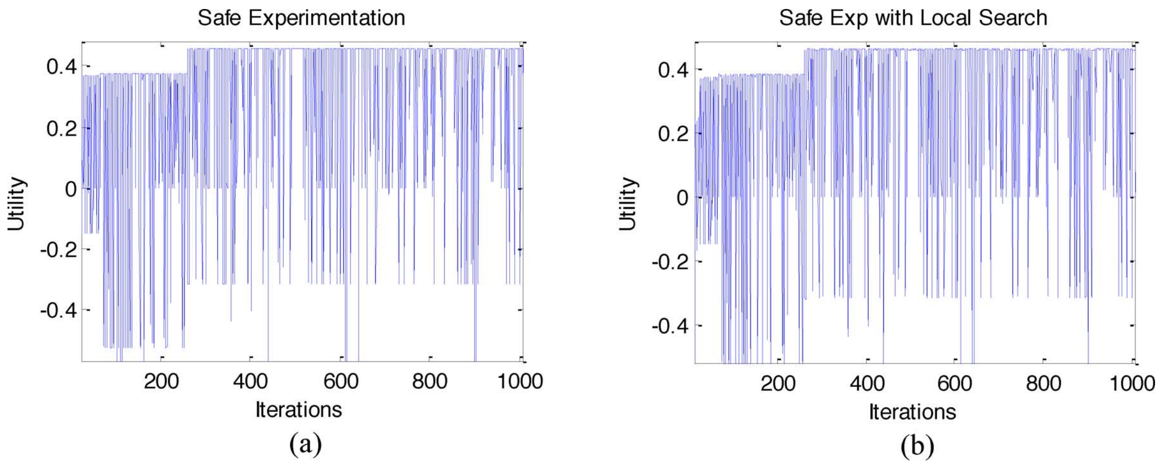


Fig. 8. Evolution of utility without resource constraints and delay deadline of 10 s for (a) safe experimentation without local search, and (b) safe experimentation with local search.

process data with exponentially distributed times with mean service time $1/\mu_i$.

B. Safe Experimentation Results for a Chain of Classifiers

We configured the operating points (thresholds) of the chain classifiers based upon different classifier service rates and different application delay sensitivities. The video shots (i.e., key frames) are extracted and input into the system at an average rate of 100 per s. The initial "car" classifier is given enough processing resources to process all images (If insufficient processing resources exist, the false alarm and detection rate would be scaled down proportionally to the fraction of data objects shed at the input, in order to reduce the load to a tolerable level for the application.) To test system cases a-b, the "mountain" and "sports" classifiers are placed on nodes with ample processing power, and very limited processing power. The resulting false alarm and detection rates are shown in the top row of Table III, compared against an optimal fusion of SVM classifier scores without explicitly considering resource constraints.[2] In this case, the load shedder simply drops from a heavily loaded queue the data objects with the highest delay timestamps, such that the average load on each classifier is 0.7. Simulations were

[2]Note that this is in fact the optimal approach when analytics are distributed and have proprietary restrictions. If it were possible to combine the datasets across these sites, one could construct a single classifier that fully optimizes over the three high-level features. The purpose of our algorithm is not to demonstrate that this approach outperforms a single classifier with shared analytics, but to solve the distributed problem under resource constraints and delay requirements.

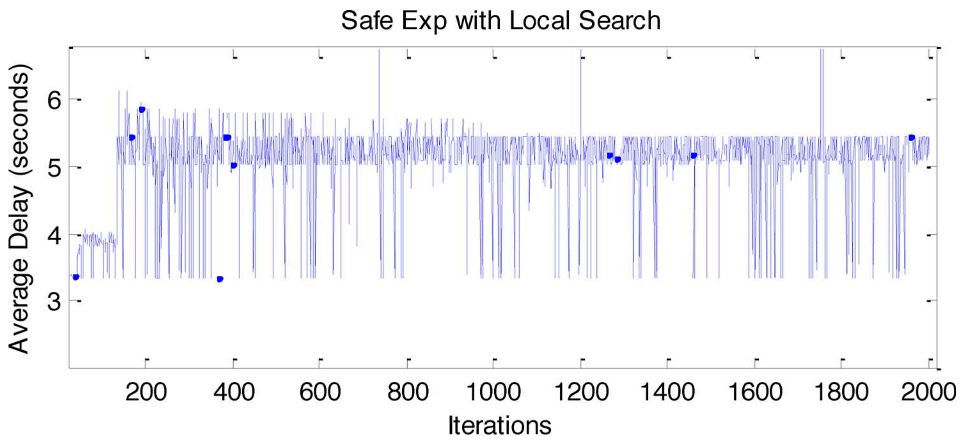


Fig. 9. Evolution of expected delay for safe experimentation with local search under a delay deadline of 10 s.

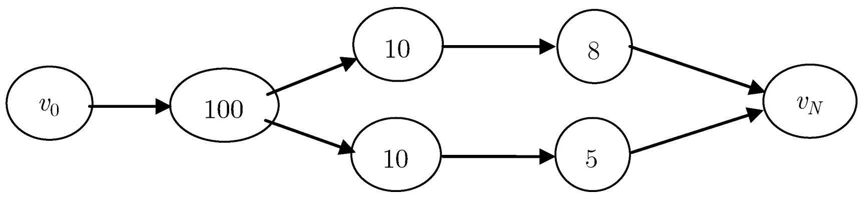


Fig. 10. Simulated multipath topology with normalized service rates listed on each classifier node.

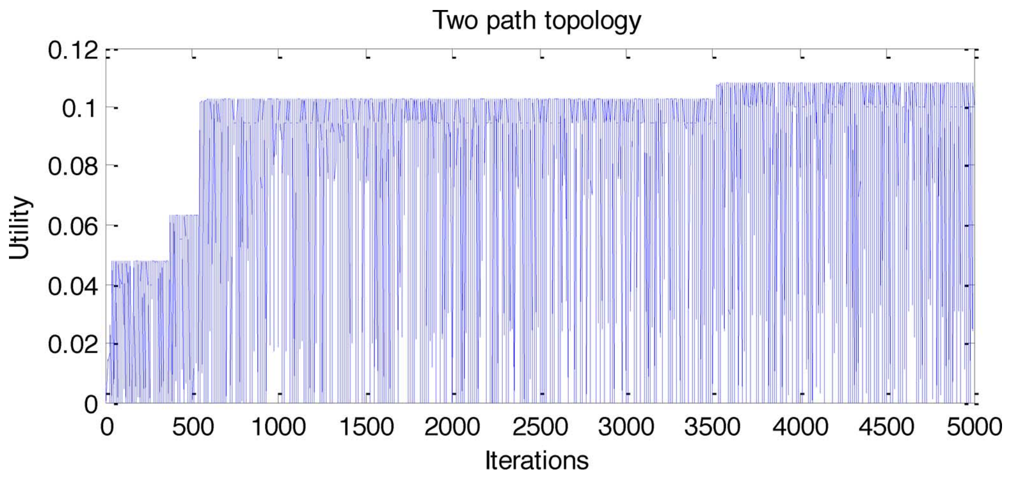


Fig. 11. Illustrative view of the utility of safe experimentation with local search used for a topology with two routes.

performed multiple times for each algorithm using the same random generator seeds for a more accurate comparison.

In Fig. 7(a), we show the evolution of the utility function under safe experimentation using an experimentation decay rate of $t^{-(1/3)}$, where t is the iteration, as well as the resulting probability of detection versus false alarm rate after 2000 iterations, when the application is not delay sensitive. The corresponding safe experimentation with local search algorithm is compared against this in Fig. 7(b). Note from Table III that the local search algorithm can achieve a slightly lower false alarm rate by discovering a better configuration within the neighborhood.

For the resource constrained, delay-sensitive setting, we determined the average service rate for the "mountain" and "car" classifiers to be approximately $m_2 = 10$, and $m_3 = 8$ images per second, respectively. The same plots are shown in Fig. 8 when a delay deadline of 10 s is set for processing, and the final detection and miss probabilities shown in the bottom row of Table III. Note that under a stringent delay deadline and low resource availability, each classifier will configure its operating point to discard a significant volume of data. Hence, the detection rate, which is around 10%, accounts for the 90% of images that would have been shed by the system given the resource constraints, regardless of the algorithm used. Compared to the result obtained by an optimal fusion of classifier scores and random load shedding, the proposed resource-aware algorithms achieve a detection rate approximately 15 times higher, and an average delay about 2 s lower, under similar false alarm rates and average delays. For a qualitative comparison, note that if a hard delay deadline of 5 s is fixed for the application, the decreased loading in the resource aware configuration significantly reduces the

TABLE IV
PERFORMANCE COMPARISON OF SPEECH DATA PROCESSING ACROSS THE MULTIPATH TOPOLOGY

	shedding	safe exp. without path selection	safe exp. with path selection
Path Selection	(0.5,0.5)	(0.5,0.5)	(0. 0.6915,0.3085)
DET values (pd, pf)	0.0064, 0.0024	0. 1129, 0.0048	0.1048, 0.0023
Average delay	0.0729	0.1194	0.1122
Utility	0.0035	0.0292	0.1081

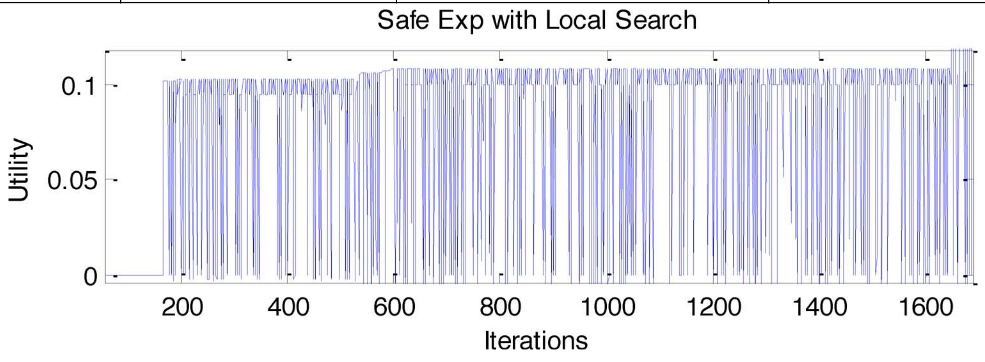


Fig. 12. Dynamic adaptation results with constant experimentation rate of 1/10.

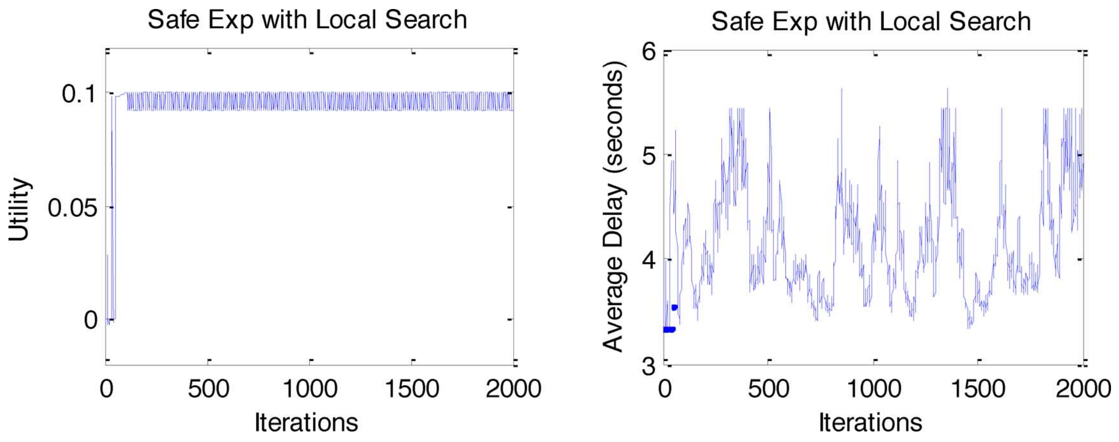


Fig. 13. Dynamic adaptation results of frequent early experimentation, and local search thereafter.

fraction of data results that are returned past the deadline. Finally, note that with local search, a near optimal configuration is found within 50 iterations, compared to safe experimentation alone which requires over 600 iterations, since the small random perturbations enables the classifiers to discover better configurations within their local neighborhoods. The evolution of delay is depicted in Fig. 9. Note that the delay converges approximately with some variation due to local search perturbations.

C. *Performance Comparisons for a Multipath Topology*

Consider now the topology shown in Fig. 10, where a stream can take two possible paths to the destination. Note that the top path consists of a second classifier that has about 1.6 times the service rate of the second classifier in the bottom path. The utility plot of one particular realization of safe experimentation with local search is shown in Fig. 11, which shows that approximately 500 iterations are required to find the near optimal point. Shown in Table IV are the results of different configuration schemes. Without information exchange, the initial classifier node will always transmit SDOs along each path with equal probability. However, as indicated by Table IV, the safe experimentation algorithm with information exchange outperforms the other algorithms, since the classifier learns that one path is better than the other. To view the impact of path selection on the overall utility, we also performed safe experimentation as in the single chain case, while fixing the path selection probabilities to be (0.5, 0.5). This choice is based upon the local utility-based algorithm, where the first classifier considers only the delay caused by the next-hop neighbor, and therefore fails to identify the resource bottleneck that occurs downstream. Nevertheless, the local utility-based algorithm performs almost optimally with respect to the proposed utility function, with only slightly worse average delay and probability of detection. Note that the proposed resource-aware algorithms both outperform the random load shedding approach by approximately 20 times in terms of detection rate under similar false alarm rate and delay.

D. *Dynamic Reconfiguration for Changing Stream Characteristics*

In order to analyze how the algorithm performs as stream characteristics change, we analyze the single path scenario again, but this time letting the APP vary over time. To simulate a dynamic stream, we let each incoming image be selected uniformly from the *a priori* positive and negative datasets with the probabilities of picking positive data varying according to a Martingale, using the following rule: $\pi_t = \max(\min(\pi_{t-1} + X, 1), 0)$, where $X = \pm s$ with probability 1/2. Here, we let $s = 0.05$, such that there is an approximate 5% change in APP per interval. Note that this is significant, since over 10 intervals, the APP can vary as much as 16% based upon properties of the binomial distribution [33]. Since each interval is attributed to filtering approximately 20000 images (the sample size) at a rate of 100 images a second, each interval corresponds to 200 s, or just under 4 min. Extrapolating from this, significant changes in stream characteristics (above 10%) would occur on average every 15 min. Due to the high cost of the APP estimation (which is comparable to the classification cost), we only re-estimate the APP once every 100 intervals. The experimentation rate is set to a constant 10%, in order to ensure that configurations outside the local neighborhood are explored approximately every 10 intervals.

Shown in Fig. 12 are the dynamic adaptation results from a particular realization. Note that the actual utility does not remain constant due to changing APP. However, approximately every 10 intervals, experimentation is used to look for a new optimal point. Because safe experimentation requires multiple iterations to discover a near optimal point, the first 160 iterations are shown to yield poor utility. However, once a good configuration is discovered, due to the slow varying rate of the APP, local search enables the algorithm to adapt to the gradually changing optimal configuration. In fact, it would seem that frequent exploration is only needed during the first 200 iterations, and then local search can be used afterwards. To determine whether this intuition is correct, we attempted to use pure random experimentation for the first 20 iterations, and local search thereafter. Shown in Fig. 13 is the result, which demonstrates that in the context of gradually changing stream characteristics, local search is a useful method to maintain high (although not necessarily optimal) utility. Due to dynamic stream characteristics but infrequent APP estimation, the average delay would vary between 3 and 5 s. Overall, this method produced a detection probability of 10.48%, and false alarm rate of 0.48%, comparable to safe experimentation in the static scenario.

VII. Conclusion

In this paper, we proposed a delay-sensitive, stream processing utility metric, and provided a distributed algorithmic framework for configuring a networked classifier topology. We showed that by exchanging local performance metrics between classifiers, each classifier can run a low complexity, model-free distributed algorithm that converges quickly to an optimal or near optimal system configuration, even when the utility function cannot be analytically characterized. We also provided insights into the tradeoffs between information availability, complexity, convergence rate, and dynamics in a classifier system, and confirmed these insights using classifiers trained for high-level concept detection. Most importantly, the proposed optimization framework can also be extended to other informationally-distributed (or informationally-restricted) multimedia networks or parallel computing systems where individually configurable entities have, as a common goal, optimizing the system performance.

An orthogonal but related direction of research involves optimizing the ordering of classifier filtering elements [51], and utilizing more general topologies, such as parallelization of some filtering elements, to improve performance [50]. Furthermore, while we have considered independent configuration of identical classifier elements over multiple distributed nodes, the instantiation of multiple classifier elements on each node can also yield interesting challenges with respect to shared resource constraints (i.e., time-sharing on a CPU). Future work could consider how safe experimentation would work for resource allocation to classifier elements on a shared processing node. The mixed continuous and discrete optimization problem of ordering, placement, resource sharing and operating point configuration would be a very challenging and interesting problem to explore.

References

[1] M. Shah, J. Hellerstein, and M. Franklin, “Flux: An adaptive partitioning operator for continuous query systems,” in *Proc. 19th Int. Conf. Data Eng.*, Mar. 2003, pp. 25–36.

[2] C. Olston, J. Jiang, and J. Widom, “Adaptive filters for continuous queries over distributed data streams,” in *Proc. Int. Conf. Manag. Data*, Jun. 2003, pp. 563–574.

[3] L. Amini, H. Andrade, F. Eskesen, R. King, Y. Park, P. Selo, and C. Venkatramani, “The stream processing core,” Tech. Rep. RSC 23798, Nov. 2005.

[4] D. Turaga, O. Verscheure, U. Chaudhari, and L. Amini, “Resource management for chained binary classifiers,” in *Proc. Workshop Tackling Comput. Syst. Problems Mach. Learn. Techniques*, 2006.

[5] Y. Schapire, “A brief introduction to boosting,” in *Proc. 16th Int. Joint Conf. Artif. Intell.*, 1999, pp. 1401–1406.

[6] E. Allwein, R. Schapire, and Y. Singer, “Reducing multiclass to binary: A unifying approach for margin classifiers,” *J. Mach. Learn. Res.*, vol. 1, pp. 113–141, Dec. 2001.

[7] R. Xiao, L. Zhu, and H. Zhang, “Boosting chain learning for object detection,” in *Proc. 9th IEEE Int. Conf. Comput. Vis.*, Oct. 2003, vol. 1, pp. 709–715.

[8] A. Garg and V. Pavlovic, “Bayesian networks as ensemble of classifiers,” in *Proc. 16th Int. Conf. Pattern Recognit.*, 2002, pp. 779–784.

[9] B. Babcock, S. Babu, M. Datar, and R. Motwani, “Chain: Operator scheduling for memory minimization in data stream systems,” in *Proc. Int. Conf. Manag. Data*, 2003, pp. 253–264.

[10] N. Tatbul, U. Cetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, “Load shedding in a data stream manager,” in *Proc. 29th Int. Conf. Very Large Databases*, Sep. 2003, pp. 309–320.

[11] B. Babcock, M. Datar, and R. Motwani, “Cost-efficient mining techniques for data streams,” in *Proc. 2nd Workshop Australasian Inf. Security, Data Mining Web Intell., and Software International.*, 2003, vol. 32, pp. 109–114.

[12] N. Tatbul and S. Zdonik, “Dealing with overload in distributed stream processing systems,” in *Proc. IEEE Int. Workshop Netw. Meets Databases*, Atlanta, GA, Apr. 2006, p. 24.

[13] N. Tatbul, “QoS-driven load shedding on data streams,” in *Proc. EDBT Ph.D. Workshop*, Prague, Czech Republic, Mar. 2002, pp. 566–576.

[14] V. Eide, F. Eliassen, O. Granmo, and O. Lysne, “Supporting timeliness and accuracy in distributed real-time content-based video analysis,” in *Proc. 11th ACM Int. Conf. Multimedia*, 2003, pp. 21–32.

[15] Y. Chi, P. Yu, H. Wang, and R. Muntz, “Loadstar: A load shedding scheme for classifying data streams,” in *Proc. 5th IEEE Int. Conf. Data Mining*, Oct. 2005, pp. 346–357.

[16] D. Turaga, O. Verscheure, U. Chaudhari, and L. Amini, “Resource management for networked classifiers in distributed stream mining systems,” in *Proc. 6th IEEE Int. Conf. Data Mining*, Dec. 2006, pp. 1102–1107.

[17] 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. Topics Signal Process.*, vol. 1, no. 4, pp. 548–563, Dec. 2007.

[18] Y. Xing, S. Zdonik, and J.-H. Hwang, "Dynamic load distribution in the borealis stream processor," in *Proc. 21st Int. Conf. Data Eng.*, Tokyo, Japan, Apr. 2005, pp. 791–802.

[19] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, and S. Zdonik, "Scalable distributed stream processing," in *Proc. Conf. Innovative Data Syst. Res.*, Asilomar, CA, Jan. 2003, pp. 257–268.

[20] D. Heckerman, "Bayesian networks for data mining," *Data Mining Knowl. Discov.*, vol. 1, no. 1, pp. 79–119, Mar. 1997.

[21] K. Murphy, "Dynamic Bayesian networks: representation, inference and learning," Ph.D. dissertation, Comput. Sci. Div., UC Berkeley, Berkeley, Jul. 2002.

[22] P. Varshney, *Distributed Detection and Data Fusion*. New York: Springer-Verlag, 1997.

[23] J. Vaidya and C. Clifton, "Privacy-preserving k-means clustering over vertically partitioned data," in *Proc. 9th ACM Int. Conf. Knowl. Discov. Data Mining*, 2003, pp. 206–215.

[24] S. Merugu and J. Ghosh, "Privacy-preserving distributed clustering using generative models," in *Proc. 3rd Int. Conf. Management Data*, 2003, pp. 211–218.

[25] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker, "Fault tolerance in the borealis distributed stream processing system," in *Proc. Int. Conf. Management Data*, 2005, pp. 13–24.

[26] H. H. Permuter, J. Francos, and I. H. Jarmyn, "Gaussian mixture models of texture and colour for image database retrieval," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, Apr. 2003, vol. 3, pp. 569–572, (2003).

[27] M. Campbell, A. Haubold, M. Liu, A. P. Natsev, J. R. Smith, J. Tešić, L. Xie, R. Yan, and J. Yang, IBM Research TRECVID-2007 Video Retrieval System [Online]. Available: http://www-nlpir.nist.gov/projects/tvpubs/tv7.papers/ibm.pdf

[28] A. Hero and J. Kim, "Simultaneous signal detection and classification under a false alarm constraint," in *Proc. Int. Conf. Acoust., Speech Signal Process.*, Apr. 1990, vol. 5, pp. 2759–2762.

[29] D. Turaga and T. Chen, "I/P frame selection using classification based mode decisions," in *Proc. Int. Conf. Image Process.*, 2001, vol. 3, pp. 550–553.

[30] A. Pentland, B. Moghaddam, and T. Starner, "View-based and modular eigenspaces for face recognition," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 1994, pp. 84–91.

[31] D. Gross and C. Harris, *Fundamentals of Queueing Theory*. Hoboken, NJ: Wiley, 1997.

[32] P. Burke, "The output of a queuing system," *Oper. Res.*, vol. 4, no. 6, pp. 699–704, 1956.

[33] R. G. Gallager, *Discrete Stochastic Processes*. Norwell, MA: Kluwer, 1996.

[34] A. Roth and I. Erev, "Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term," *Games Econom. Behav.*, vol. 8, pp. 164–212, 1995.

[35] D. Bertsekas and R. Gallager, *Data Networks*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 1991.

[36] S. Viglas and J. Naughton, "Rate-based query optimization for streaming information sources," in *Proc. Int. Conf. Management Data*, 2002, pp. 37–48.

[37] 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.

[38] D. Abadi, D. Carney, U. Centintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: A new model and architecture for data stream management," *Very Large Databases J.*, vol. 12, no. 2, pp. 120–139, Aug. 2003.

[39] V. Kumar, B. Cooper, and K. Schwan, "Distributed stream management using utility-driven self-adaptive middleware," in *Proc. 2nd Int. Conf. Auton. Comput.*, Jun. 2005, pp. 3–14.

[40] E. Horvitz and G. Rutledge, "Time-dependent utility and action under uncertainty," in *Proc. 7th Conf. Uncertainty Artif. Intell.*, Jul. 1991, pp. 151–158.

[41] N. Weiderman, "Hartstone: Synthetic benchmark requirements for hard real-time applications," *Ada Lett.*, vol. 10, no. 3, Winter 1990.

[42] P. Young, *Strategic Learning and Its Limits*. Oxford, U.K.: Oxford Univ. Press, 2004.

[43] F. Douglis, M. Branson, K. Hildrum, B. Rong, and F. Ye, "Multi-site cooperative data stream analysis," *Special Interest Group on Operating Systems*, vol. 40, no. 3, pp. 31–37, Jul. 2006.

[44] J. Marden, H. Young, G. Arslan, and J. Shamma, "Payoff based dynamics for multi-player weakly acyclic games," *J. Control Optim.*, vol. 48, Special Issue on Control and Optimization in Cooperative Networks, no. 1, 2007.

[45] D. Fudenberg and J. Tirole, *Game Theory*. Cambridge, MA: MIT Press, 1991.

[46] P. Pardalos, H. Romeijn, and H. Tuy, "Recent developments and trends in global optimization," *J. Comput. Appl. Math.*, vol. 124, no. 1–2, pp. 209–228, Dec. 2000.

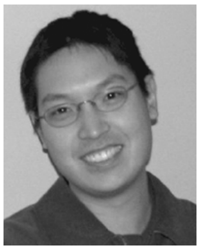
[47] R. Horst and P. M. Pardalos, *Handbook of Global Optimization*. Dordrecht, The Netherlands: Kluwer, 1995.

[48] K. Arrow, *Social Choice and Individual Values*. New Haven, CT: Yale Univ. Press, 1970.

[49] J. Nash, "The bargaining problem," *Economet.*, vol. 18, no. 2, pp. 155–162, Apr. 1950.

[50] A. Condon, A. Deshpande, L. Hellerstein, and N. Wu, "Flow algorithms for two pipelined filter ordering problems," in *Proc. Symp. Principles Database Syst.*, 2006, pp. 193–202.

[51] Z. Liu, S. Parthasarathy, A. Ranganathan, and H. Yang, "A generic flow algorithm for shared filter ordering problems," in *Proc. Symp. Principles Database Syst.*, 2008, pp. 79–88.



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

He is currently a Research Scientist at Lockheed Martin Space Systems Company, Advanced Technology Center, Sunnyvale, CA. His interests lie in 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 DAC and SPIE conferences, respectively.

Mihaela van der Schaar (SM'04) is currently an Associate Professor in the Electrical Engineering Department at the University of California, Los Angeles.

Dr. van der Schaar received the NSF Career Award in 2004, the Best Paper Award from the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY in 2005, the Okawa Foundation Award in 2006, the IBM Faculty Award in 2005, 2007, and 2008, and the Most Cited Paper Award from the *Image Communications Journal* in 2006. She holds 30 granted U.S. patents and three ISO awards.