SFINX: A Multi-sensor Fusion and Mining System*

Zoran Dimitrijević
zoran@cs.ucsb.edu

Gang Wu
gwu@engineering.ucsb.edu

Edward Y. Chang
echang@vimatech.com

Abstract

In a surveillance system, video signals are generated by multiple cameras with or without spatially and temporally overlapping coverage. These signals need to be transmitted, processed, fused, stored, indexed, and then summarized as semantic events to allow efficient and effective querying and mining. SfinX *aims to build several core components for multi-sensor fusion and mining. This paper first depicts* SfinX*' architecture and its core components, namely, data fusion, event detection, event characterization, event recognition, and storage. In particular, we survey representative methods and discuss plausible research directions for event recognition and storage.*

Keywords: video surveillance, event recognition, real-time storage.

1 Introduction

Government agencies, business, schools, and homes are increasingly turning toward video surveillance as a means to increase public security. With the proliferation of inexpensive cameras and the availability of high-speed, broad-band wired/wireless networks, deploying a large number of cameras for security surveillance has become economically and technically feasible. However, several important research questions remain to be addressed before we can rely upon video surveillance as an effective tool for crime prevention. The SfinX project aims to develop several core components to process, transmit, and fuse video signals from multiple cameras, to mine unusual activities from the collected trajectories, and to index and store video information for effective viewing.

Figure 1 depicts a typical hardware architecture of SfinX. Cameras are mounted at the edges of a sensor network to collect signals (shown on the upper-right of the figure). When activities are detected, signals are compressed and transfered to a server (lower-left of the figure). The server fuses multi-sensor data and constructs spatio-temporal descriptors to depict the captured activities. The server indexes and stores video signals with their meta-data on RAID storage (lower-right of the figure). Users of the system (upper-left of the figure) are alerted to unusual events and they can perform online queries to retrieve and inspect video-clips of interest.

*This work was partially supported by NSF grants IIS-0133802 (Career), IIS-0219885 (ITR), and EIA-0080134.

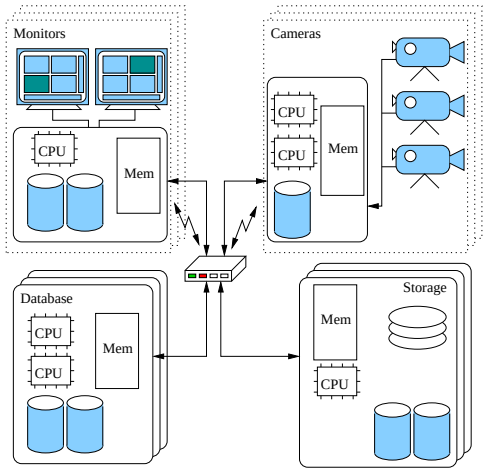

Figure 1. System architecture.

A surveillance task can be divided into four phases: *event detection*, *event representation*, *event recognition*, and *event query* [4]. The *detection* phase handles multi-source spatio-temporal data fusion for efficient and reliable extraction of motion trajectories from videos. The *representation* phase summarizes raw trajectory data to construct hierarchical, invariant, and adequate representations of the motion events. The *recognition* phase deals with event recognition and classification. Lastly, the *query* component indexes and retrieves videos that meet some query criteria.

In this paper, we focus our discussion on *event recognition* and *event query*. (Please see [10] for our discussion on the other components.) Both components are designed for achieving the same goal: returning information matching the user-query criteria in an efficient way. A query, for instance, can be worded like this: "select object = *'vehicles'* where event = *'circling'* and location = *'parking lots'* and time = *'since 9pm last night'*." Another example-query might be "select object = *'vehicle A'* where event = *'*'* and location = *'*'* and time = *'since 9pm last night'*." To answer these queries both effectively and efficiently, we design the recognition and query components of SfinX to fulfill the following requirements:

1. Recognizing spatio-temporal patterns under extreme statistical constraints. Event recognition deals with mapping motion patterns to semantics (e.g., benign and suspicious events). Recognizing rare events comes up against two mathematical challenges. First, the number of training instances that can be collected for modeling rare events is

typically very small. Let N denote the number of training instances, and D the dimensionality of data. Traditional statistical models such as the Hidden Markov Model (HMM) cannot work effectively under the $N < D$ constraint. Furthermore, positive events (i.e., the sought-for hazardous events) are always significantly outnumbered by negative events in the training data. In such an imbalanced set of training data, the class boundary tends to skew toward the minority class and hence results in a high incidence of false negatives.

2. Retrieving videos efficiently via different access paths. Video data can be accessed via a variety of attributes, e.g., by objects, temporal attributes, spatial attributes, pattern similarity, and by any combinations of the above. Simultaneously supporting high-throughput writes (recording videos on RAID) and short response reads (retrieving video segments relevant to a query) presents conflicting design requirements for memory management, disk scheduling, and data placement policies.

We present preliminary results of our research on event recognition and video indexing in the remainder of this paper.

2 Event Recognition

In video surveillance, event recognition deals with classifying and identifying particular motion patterns. In SfinX, each motion trajectory is summarized as an ordered sequence of labels using syntactic and semantic descriptors. Each label corresponds to a motion segment with a humanly understandable action. For instance, the following example depicts a sequence with a two-level descriptor: a primary segment symbol and two secondary variables—velocity and acceleration. Sequence "$\mathbf{s}$" denotes the segmented trajectory. Each symbol in the sequence denotes the semantic label. For example, "R" represents "right turn," "L" represents "left turn," and "C" "constant direction." Lastly, $\mathbf{v}_i$ denotes the vector of the i^{th} secondary variable. In this example, $\mathbf{v}_1$ is velocity and $\mathbf{v}_2$ is acceleration.

$\mathbf{s}:$	L	R	R	C	L	R	R	L
$\mathbf{v}_1:$	0.7	0.5	0.8	0.8	0.7	0.8	0.6	0.5
$\mathbf{v}_2:$	0.0	-0.2	0.3	0.0	-0.1	0.1	-0.2	-0.1

2.1 Generative vs. Discriminative Model

Now, given a set of aforementioned sequence observations $\mathcal{X} = \{\mathbf{x}_i, y_i\}_1^N$, where x_i consists of s_i and some v_{ij}, the event recognition problem is to learn a classifier, which assigns $\mathbf{x}_i$ to its true event-class $y_i \in \{1, 2, ..., K\}$.

A classifier generally belongs to one of the two learning models, generative or discriminative.

1. *Generative.* A generative model learns a class density $p(\mathbf{x}|y = k)$ for each class $k, k = 1, 2, ..., K$. For a query instance, its label is predicted first by using the Bayes rule to calculate the class posterior probability $p(y = k|\mathbf{x})$ for each k, and then by selecting the most likely class k using the *Maximum A Posterior* (MAP) criterion. Linear Discriminant Analysis (LDA), Hidden Markov Models (HMM), and Naive Bayes (NB) [7] belong to the generative model, in which each class density $p(\mathbf{x}|y = k)$ is learned separately, using only training instances belonging to class k.

2. *Discriminative.* A discriminative model directly estimates the posterior $p(y = k|\mathbf{x})$ for class k without modeling the class density $p(\mathbf{x}|y = k)$. Gaussian Process (GP) and Logistic Regression belong to the discriminative camp. Some discriminative classifiers such as Support Vector Machines (SVMs) directly learn a discriminant function to model the class boundary between classes. In general, a discriminative model considers both positive and negative training instances in the learning process.

The generative model and the discriminative model each enjoys its pros and cons. For detecting rare events, we employ the discriminative model because of the following two practical constraints:

1. $N < D$. In video surveillance, the size of training dataset N for modeling rare events is typically very small and can be under-representative. A generative method such as HMM requires $N >> D$ to learn a robust classifier, and hence may not be suitable. On the other hand, SVMs can depict class boundary (though fuzzily) with a small number of training instances.

2. $N^+ \ll N^-$. Positive training data (i.e., the suspicious events we are seeking) are always significantly outnumbered by negative data. A discriminative model such as SVM is known to be *less sensitive* to the imbalanced dataset learning because it performs class prediction without explicitly considering prior distributions. However, since the imbalanced dataset learning problem still plagues SVMs, we will discuss the problem and our remedies shortly.

2.2 Kernel Boundary Alignment

In our prior work [8], we proposed an adaptive conformal transformation ACT, based on feature-space distribution, to improve SVMs in learning imbalanced vector data. Kernel boundary alignment, KBA, generalizes the work of ACT to deal with both vector data and non-vector data (e.g., sequence data). Here, we first give a brief description of ACT and then present KBA.

2.3 Conformally Transforming K

Kernel-based methods, such as SVMs, introduce a mapping function Φ which embeds the the input space I into a high-dimensional feature space F as a curved Riemannian manifold S where the mapped data reside [1, 2]. A Riemannian metric $g_{ij}(\mathbf{x})$ is then defined for S, which is associated with the kernel function $K(\mathbf{x}, \mathbf{x}')$ by

$$g_{ij}(\mathbf{x}) = \left(\frac{\partial^2 K(\mathbf{x}, \mathbf{x}')}{\partial x_i \partial x'_j} \right)_{\mathbf{x}'=\mathbf{x}}. \quad (1)$$

The metric g_{ij} shows how a local area around $\mathbf{x}$ in I is magnified in F under the mapping of Φ. The idea of con-

formal transformation in SVMs is to enlarge the margin by increasing the magnification factor $g_{ij}(\mathbf{x})$ around the boundary (represented by support vectors) and to decrease it around the other points. This could be implemented by a conformal transformation of the related kernel $K(\mathbf{x}, \mathbf{x}')$ according to Eq. 1, so that the spatial relationship between the data would not be affected too much [1]. Such a conformal transformation can be depicted as

$$\tilde{K}(\mathbf{x}, \mathbf{x}') = D(\mathbf{x})D(\mathbf{x}')K(\mathbf{x}, \mathbf{x}'). \quad (2)$$

In the above equation, $D(\mathbf{x})$ is a properly defined positive conformal function. $D(\mathbf{x})$ should be chosen in a way such that the new Riemannian metric $\tilde{g}_{ij}(\mathbf{x})$, associated with the new kernel function $\tilde{K}(\mathbf{x}, \mathbf{x}')$, has larger values near the decision boundary. Furthermore, to deal with the skew of the class boundary caused by imbalanced classes, we magnify $\tilde{g}_{ij}(\mathbf{x})$ more in the boundary area close to the minority class. In [8], we demonstrate that an RBF distance function such as

$$D(\mathbf{x}) = \sum_{k \in SV} \exp(-\frac{|\mathbf{x} - \mathbf{x}_k|}{\tau_k^2}) \quad (3)$$

is a good choice for $D(\mathbf{x})$.

In Eq. 3, we can see that if τ_k^2's are fixed for all support vectors $\mathbf{x}_k$'s, $D(\mathbf{x})$ would be very dependent on the density of support vectors in the neighborhood of $\Phi(\mathbf{x})$. To alleviate this problem, we adaptively tune τ_k^2 according to the spatial distribution of support vectors in F [8]. This goal can be achieved by the following equations:

$$\tau_k^2 = AVG_{i \in \{\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2 < M,\ y_i \neq y_k\}} \left(\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2\right). \quad (4)$$

In this equation, the average on the right-hand side comprises all support vectors in $\Phi(\mathbf{x}_k)$'s neighborhood within the radius of M but having a different class label. Here, M is the average distance of the nearest and the farthest support vectors from $\Phi(\mathbf{x}_k)$. Setting τ_k^2 in this way takes into consideration the spatial distribution of the support vectors in F. Although the mapping Φ is unknown, we can play the kernel trick to calculate the distance in F:

$$\|\Phi(\mathbf{x}_i) - \Phi(\mathbf{x}_k)\|^2 = K(\mathbf{x}_i, \mathbf{x}_i) + K(\mathbf{x}_k, \mathbf{x}_k) - 2 \times K(\mathbf{x}_i, \mathbf{x}_k). \quad (5)$$

Substituting Eq. 5 into Eq. 4, we can then calculate the τ_k^2 for each support vector, which can adaptively reflect the spatial distribution of the support vector in F, not in I.

When the training dataset is very imbalanced, the class boundary would be skewed towards the minority class in the input space I. We hope that the new metric $\tilde{g}_{ij}(\mathbf{x})$ would further magnify the area far away from a minority support vector $\mathbf{x}_i$ so that the boundary imbalance could be alleviated. Our algorithm thus assigns a multiplier for the τ_k^2 in Eq. 4 to reflect the boundary skew in $D(\mathbf{x})$. We tune $\tilde{\tau}_k^2$ as $\eta_p \tau_k^2$ if $\mathbf{x}_k$ is a minority support vector; otherwise, we tune it as $\eta_n \tau_k^2$. Examining Eq. 3, we can see that $D(\mathbf{x})$ is a monotonically increasing function of τ_k^2. To increase the metric $\tilde{g}_{ij}(\mathbf{x})$ in an area which is not very close to the support vector $\mathbf{x}_k$, it would be better to choose a larger η_p for the τ_k^2 of a minority support vector. For a majority support vector, we can choose a smaller η_n, so as to minimize influence on the class-boundary. We empirically demonstrate that η_p and η_n are proportional to the skew of support vectors, or η_p as $O(\frac{|\mathbf{SV}^-|}{|\mathbf{SV}^+|})$, and η_n as $O(\frac{|\mathbf{SV}^+|}{|\mathbf{SV}^-|})$, where $|\mathbf{SV}^+|$ and $|\mathbf{SV}^-|$ denote the number of minority and majority support vectors, respectively. (Please see [8] for the details of ACT.)

2.4 Modifying K

For data that do not have a vector-space representation (e.g., sequence data), ACT may not be applicable. We thus employ KBA [9], which modifies kernel matrix $\mathbf{K}$ based on training-data distribution. Kernel matrix $\mathbf{K}$ contains the pairwise similarity information between all pairs of instances in a training dataset. Hence, in kernel-based methods, all we need is a kernel matrix to learn the classifier, even if the data do not reside in a vector space.

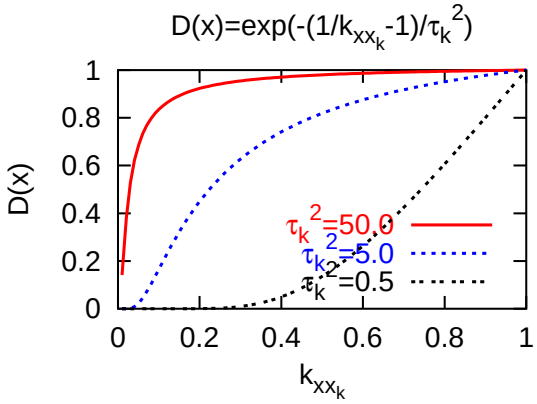

Figure 2. $\mathbf{D}(\mathbf{x})$ vs. $\mathbf{k}_{\mathbf{xx_k}}$ with different $\tau_\mathbf{k}^2$.

Now, since a training instance $\mathbf{x}$ might not be a vector, we introduce a term, *support instance*, to denote $\mathbf{x}$ when its embedded point via $\mathbf{K}$ is a support vector[1]. In this situation, we cannot choose $D(\mathbf{x})$ as in Eq. 3. (It is impossible to calculate the Euclidean distance $|\mathbf{x} - \mathbf{x}_i|$ for non-vector data.) In Section 2.3, we show that $D(\mathbf{x})$ should be chosen in such a way that the spatial resolution of the manifold S would be magnified around the support instances. In other words, if $\mathbf{x}$ is close to a support instance $\mathbf{x}_k$ in F (or in its neighborhood), we hope that $D(\mathbf{x})$ would be larger so as to achieve a greater magnification. In KBA, we use the pairwise-similarity $k_{\mathbf{xx_k}}$ to measure the distance of $\mathbf{x}$ from $\mathbf{x}_k$ in F. Therefore, we choose $D(\mathbf{x})$ as

$$D(\mathbf{x}) = \sum_{k \in SI} \exp\left(-\frac{\frac{1}{k_{xx_k}} - 1}{\tau_k^2}\right), \quad (6)$$

where SI denotes the support-instance set, and τ_k^2 controls the magnitude of $D(\mathbf{x})$.

Figure 2 illustrates a $D(\mathbf{x})$ for a given support instance $\mathbf{x}_k$, where we can see that $D(\mathbf{x})$ (y-axis) becomes larger

[1] In KBA algorithm, if $\mathbf{x}$ is a support instance, we call both $\mathbf{x}$ and its embedded support vector via $\mathbf{K}$ in F *support instance*.

when an instance $\mathbf{x}$ is more similar to $\mathbf{x}_k$ (a larger $k_{\mathrm{xx_k}}$ in the x-axis), so that there would be more magnification on the spatial resolution around the support vector embedded by $\mathbf{x}_k$ in F. Notice in the figure that $D(\mathbf{x})$ can be shaped very differently with different τ_k^2. We thus need to adaptively choose τ_k^2 as

$$\tau_k^2 = AVG_{i \in \{Dist^2(\mathbf{x}_i, \mathbf{x}_k) < M,\ y_i \neq y_k\}} Dist^2(\mathbf{x}_i, \mathbf{x}_k), \quad (7)$$

where the distance $Dist^2(\mathbf{x}_i, \mathbf{x}_k)$ between two support instances $\mathbf{x}_i$ and $\mathbf{x}_k$ is calculated via the kernel trick as

$$Dist^2(\mathbf{x}_i, \mathbf{x}_k) = k_{x_i x_i} + k_{x_k x_k} - 2 \times k_{x_i x_k}. \quad (8)$$

The neighborhood range M in Eq. 7 is chosen as the average of the minimal distance $Dist^2_{min}$ and the maximal distance $Dist^2_{max}$ from $\mathbf{x}_k$. In addition, τ_k^2 is scaled in the same way as we did in Section 2.3 for dealing with the imbalanced training-data problem. For details of KBA and results of our experiments, please refer to [9].

3 Event Query

In addition to standard performance requirements (high throughput, high availability, scalability, etc.), the storage system for surveillance applications must support real-time IOs to store and retrieve video segments. For instance, a casino in San Diego uses two large RAIDs to store video recordings from 900 high-resolution cameras and to retrieve video clips on demand to 20 displays. The RAIDs must support simultaneously 900 real-time write streams and 20 interactive read streams (playback, instant-replay, and fast-forward operations) to keep the surveillance system operational. The requirements for such a storage system can be summarized as follows:

1. *Differentiated service.* Surveillance applications require both real-time and traditional best-effort data access. This means that a storage system should differentiate IO requests and service them according to their quality of service (QoS) requirements. In effect, this requires that each IO request is associated with its QoS requirements.

2. *Guaranteed-latency response.* Read requests must be serviced before their deadlines, or "viewing hiccups" might occur. There are two types of deadlines: the time to retrieve the first frame into main memory for decoding, and the time to supply subsequent frames to the decoder. Write requests can be delayed at the cost of more DRAM buffering space.

3. *High throughput.* A storage system must support high throughput given QoS requirements and real-time constraints.

To meet these performance requirements, SfinX's storage system offers strategies in *data placement*, *admission control*, *disk scheduling*, and *backup manager*. Due to space limitations, we discuss only its *admission control* and *disk scheduling* modules.

3.1 Admission Control

The *admission control* module [5] uses the following analytical model to decide if a newly arrived request (read or write) can be admitted to a RAID disk. Given N streams to support, in each time cycle the RAID must perform N IO operations, each consisting of a latency component and a data transfer component. Let T_{disk} denote the disk cycle time. Let L_{disk_i} denote the disk latency to start IO transfer for stream i. Let r_{RT} denote the fraction of time reserved for real-time streams within each disk cycle. Let B_i denote the bit-rate for a stream i. Let R_{disk_i} denote the disk throughput for the zone on which stream i resides. Then the relationship between these parameters can be expressed as

$$r_{RT} \times T_{disk} \geq \sum_{i=1}^{N} L_{disk_i} + \sum_{i=1}^{N} \frac{T_{disk} \times B_i}{R_{disk_i}} \ .$$

The above equation can be simplified as

$$T_{disk} \quad \geq \quad \frac{N \times \bar{L}_{disk} \times R_{disk}}{r_{RT} \times R_{disk} - N \times \bar{B}} \quad (9)$$

where $r_{RT} \times R_{disk} > N \times \bar{B}$. When a read IO is requested, the *admission control* module of SfinX checks to see whether Inequality 9 is satisfied. If not, the read request is rejected.

3.2 Disk Scheduling

The *disk scheduling* module is responsible for the local disk scheduling and buffer management on each storage node. SfinX uses time-cycle scheduling for guaranteed-rate real-time streams, improved to also support non-real-time IO requests with different priorities. Figure 3 depicts cycle-based scheduling with three guaranteed-rate streams. The scheduler maintains a bubble slot [3], and services high priority IO requests in the bubble slot after the *admission control* module decided that servicing the requests will not degrade the system QoS.

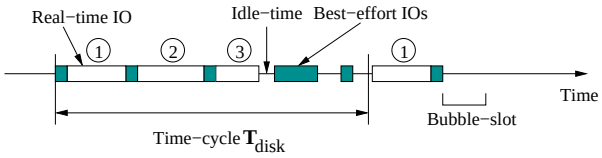

Figure 3. Cycle-based disk QoS scheduler.

To achieve short latency for high-priority requests, while maintaining high disk throughput, SfinX uses preemptible disk scheduler, which services IO requests using *Semi-preemptible IO* [6]. *Semi-preemptible IO* maps each IO request into multiple fast-executing (and hence short-duration) disk commands using three methods. (The ongoing IO request can be preempted between these disk commands.) Each of these three methods addresses the reduction of one of the following IO components: $T_{transfer}$ (denoting transfer time), T_{rot} (denoting rotational delay), and T_{seek} (denoting seek time).

1. *Chunking* $T_{transfer}$. A large IO transfer is divided into a number of small chunk transfers, and preemption is made possible between the small transfers. If the IO is not preempted between the chunk transfers, chunking does not incur any overhead. This is due to the prefetching mechanism in current disk drives.

2. *Preempting T_{rot}.* By performing just-in-time (JIT) seek for servicing an IO request, we can virtually eliminate the rotational delay at the destination track. The pre-seek slack time thus obtained is preemptible. This slack can also be used to perform prefetching for the ongoing IO request, or/and to perform seek splitting.

3. *Splitting T_{seek}. Semi-preemptible IO* splits a long seek into sub-seeks, and permits a preemption between two sub-seeks.

The following example illustrates how *Semi-preemptible IO* improves the expected waiting time for a high-priority request.

[Illustrative Example] Suppose a 2 MB read-request has to seek $20,000$ cylinders requiring T_{seek} of 14 ms, must wait for a T_{rot} of 8 ms, and requires $T_{transfer}$ of 100 ms at a transfer rate of 20 MBps. The expected waiting time, $E(T_{waiting})$, for a higher-priority request arriving during the execution of this request, is 61 ms, while the maximum waiting time is 122 ms. *Semi-preemptible IO* can reduce the waiting time by performing the following operations.

It first predicts both the seek time and rotational delay. Since the predicted seek time is long ($T_{seek} = 14$ ms), it decides to split the seek operation into two sub-seeks, each of $10,000$ cylinders, requiring $T'_{seek} = 9$ ms each. This seek splitting does not incur extra overhead because the $T_{rot} = 8$ can mask the 4 ms increased total seek time ($2 \times T'_{seek} - T_{seek} = 2 \times 9 - 14 = 4$) incurred by seek splitting, which is not always possible. The rotational delay is now $T'_{rot} = T_{rot} - (2 \times T'_{seek} - T_{seek}) = 4$ ms. With this $T'_{rot} = 4$ ms knowledge, *Semi-preemptible IO* can wait for 4 ms before performing a JIT-seek. The JIT-seek method makes T'_{rot} preemptible, since no disk operation is being performed. The disk then performs the two sub-seek disk commands, and then 100 successive read commands, each of size 20 kB, requiring 1 ms each. A high-priority IO request can be serviced immediately after any disk-command. *Semi-preemptible IO* thus makes preemptible the originally non-preemptible IO request. During the service of this IO, we have two scenarios:

1. *No higher-priority IO arrives.*
In this case, the disk does not incur additional overhead for transferring data (due to disk prefetching) nor any additional disk latency (due to rotational delay prediction). (If T_{rot} cannot mask the seek-splitting overhead, we can choose not to perform seek-splitting.)

2. *A higher-priority IO arrives.*
In this case, the maximum waiting time for the high-priority request is now a mere 9 ms, if it arrives during one of the two seek disk commands. However, if the ongoing request is at the data-transfer stage, the longest stall for the high-priority request is just 1 ms. The expected value for waiting time is only $\frac{1}{2} \frac{0 \times 4^2 + 2 \times 9^2 + 100 \times 1^2}{4 + 2 \times 9 + 100} = 1.1$ ms, a significant reduction from 61 ms.

□

Both the *admission control* module and the *disk scheduling* module have been implemented in Linux. Please refer to [6] for details.

4 Conclusion

In this paper, we have introduced SfinX and reviewed its event recognition and event query components. We enumerated challenges and requirements for these components and outlined our solutions. We plan to enhance our system in several ways. For KBA, we are investigating more robust kernel-matrix alignment methods that can achieve good generalization performance (to unseen data) without suffering from overfitting. Preliminary results show KBA to be promising, and we are researching theoretical justifications for its effectiveness. For event query, we are investigating effective sequence-data indexing methods to support queries via different attributes. Although SfinX is oriented to support surveillance applications, its components will continue to involve research in the areas of Computer Vision, Signal Processing, Machine Learning, Databases, and Systems.

References

[1] S. Amari and S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999.

[2] C. Burges. *Geometry and Invariance in Kernel Based Methods. In Adv. in Kernel Methods: Support Vector Learning*. MIT Press, 1999.

[3] E. Chang and H. Garcia-Molina. Bubbleup - Low latency fast-scan for media servers. *Proceedings of the 5th ACM Multimedia Conference*, pages 87–98, November 1997.

[4] R. T. Collins, A. J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt, and L. Wixson. A system for video surveillance and monitoring (VSAM project final report). *CMU Technical Report CMU-RI-TR-00-12*, 2000.

[5] Z. Dimitrijevic, R. Rangaswami, and E. Chang. The XTREAM multimedia system. *Proceedings of the IEEE Conference on Multimedia and Expo*, August 2002.

[6] Z. Dimitrijevic, R. Rangaswami, and E. Chang. Design and implementation of Semi-preemptible IO. *Proceeding of the 2nd Usenix FAST*, pages 145–158, March 2003.

[7] Y. D. Rubinstein and T. Hastie. Discriminative vs. informative learning. *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, March 1997.

[8] G. Wu and E. Chang. Adaptive feature-space conformal transformation for learning imbalanced data. *Proceedings of the Twentieth International Conference of Machine Learning (ICML)*, pages 816–823, August 2003.

[9] G. Wu and E. Chang. Class-boundary alignment for learning imbalanced data. *Proceedings of the Twentieth International Conference of Machine Learning (ICML) Workshop on Learning from Imbalanced Datasets*, pages 49–56, August 2003.

[10] G. Wu, Y. Wu, L. Jiao, Y.-F. Wang, and E. Y. Chang. Multi-camera spatio-temporal fusion biased sequence-data learning for video surveillance. *ACM International Conference on Multimedia*, November 2003.