Supporting a Real-Time Distributed Intrusion Detection Application on GATES

Qian Zhu, Liang Chen, and Gagan Agrawal

Department of Computer Science and Engineering
The Ohio State University, Columbus OH 43210
{zhuq, chenlia, agrawal}@cse.ohio-state.edu

Abstract. Increasingly, a number of applications across computer sciences and other science and engineering disciplines rely on, or can potentially benefit from, analysis and monitoring of *data streams*. We view the problem of flexible and adaptive processing of distributed data streams as a grid computing problem. In our recent work, we have been developing a middleware, GATES (Grid-based AdapTive Execution on Streams), for enabling grid-based processing of distributed data streams.

This paper reports an application study using the GATES middleware system. We focus on the problem of intrusion detection. We have created a distributed and self-adaptive real-time implementation of the algorithm proposed by Eskin using our middleware. The main observations from our experiments are as follows. First, our distributed implementation can achieve detection rates which are very close to the detection rate by a centralized algorithm. Second, our implementation is able to effectively adjust the adaptation parameters.

1 Introduction

Increasingly, a number of applications across computer sciences and other science and engineering disciplines rely on, or can potentially benefit from, analysis and monitoring of *data streams*. In the stream model of processing, data arrives continuously and needs to be processed in *real-time*, i.e., the processing rate must match the arrival rate. There are several trends contributing to the emergence of this model. First, scientific simulations and increasing numbers of high precision data collection instruments (e.g. sensors attached to satellites and medical imaging modalities) are generating data continuously, and at a high rate. The second is the rapid improvements in the technologies for Wide Area Networking (WAN). As a result, often the data can be transmitted faster than it can be stored or accessed from disks within a cluster.

The important characteristics that apply across a number of stream-based applications are: 1) the data arrives continuously, 24 hours a day and 7 days a week, 2) the volume of data is enormous, typically tens or hundreds of gigabytes a day, and the desired analysis could also require large computations, 3) often, this data arrives at a distributed set of locations, and all data cannot be communicated to a single site, 4) it is often not feasible to store all data for processing at a later time, thereby, requiring analysis in *real-time*.

W.E. Nagel et al. (Eds.): Euro-Par 2006, LNCS 4128, pp. 360–370, 2006.

We view the problem of flexible and adaptive processing of distributed data streams as a grid computing problem. We believe that a distributed and networked collection of computing resources can be used for analysis or processing of these data streams. Computing resources close to the source of a data stream can be used for initial processing of the data stream, thereby reducing the volume of data that needs to be communicated. Other computing resources can be used for more expensive and/or centralized processing of data from all sources.

In our recent work, we have been developing a middleware for enabling grid-based processing of distributed data streams [6,5]. Our system is referred to as GATES (Grid-based AdapTive Execution on Streams). One of the important characteristic of this middleware is that it can enable an application to achieve the best accuracy, while maintaining the *real-time* constraint. For this, the middleware allows the application developers to expose one or more *adaptation* parameters. An adaptation parameter is a tunable parameter whose value can be modified to increase the processing rate, and in most cases, reduce the accuracy of the processing. Examples of such adaptation parameters are, rate of sampling, i.e., what fraction of data-items are actually processed, and size of summary structure at an intermediate stage, which means how much information is retained after a processing stage. The middleware automatically adjusts the values of these parameters to meet the real-time constraint on processing, through a *self-adaptation* algorithm. Self-adaptation algorithms currently implemented in the middleware are described in our earlier papers [6,5].

This paper reports an application study using the GATES middleware system. We focus on the problem of intrusion detection, which a widely studied problem in computer security and data mining [1]. We have created a distributed and self-adaptive real-time implementation of the algorithm proposed by Eskin [3]. This implementation generates local models using data received at each node, and then combines these local models to create a global model. We use the functionality of GATES in two different ways. First, as network records typically arrive at multiple locations, a flexible distributed implementation can avoid high communication costs associated with a centralized implementation. Second, as data arrival rates can vary significantly, it is important for an intrusion detection implementation to choose the right trade-off between accuracy and processing rate, to continue to meet real-time constraints.

We have carried-out a number of experiments to evaluate our distributed implementation. The main observations from our experiments are as follows. First, our distributed implementation can achieve detection rates which are very close to the detection rate by a centralized algorithm. Second, our implementation is able to adjust the adaptation parameters. When the rate of data arrival is low, it chooses a small value of the adaptation parameter, EM convergence threshold, resulting in the best detection rate. On the other hand, when the data arrival rate is very high, it chooses a larger value of this parameter, resulting in somewhat lower accuracy, but still maintaining the same rate of processing.

Input: k, # of EM clusters, $D = \{d_1, d_2, \ldots, d_n\}$, set of n $10-$*dimentional* points,
λ, probability for the set of intrusions, c, anomaly detection threshold.
Output: intrusion detection result.
var
M_t = probability distribution for normal elements at time t
A_t = probability distribution for anomalous elements at time t
C_t = number of intrusions detected at time t
begin
M_t = GMM generated by Expectation Maximization (EM) algorithm on D
A_t = a uniform distribution
Logistic Regression (LR) on D using 3 categorical attributes
for t = 1 to n
$LL_t(D) = |M_t|log(1-\lambda) + \sum_{x_i \in M_t} log(P_{Mt}(x_i)) + |A_t|log(\lambda) + \sum_{x_i \in A_t} log(P_{At}(x_i))$
$M_t = M_{t-1} - x_t$
$A_t = A_{t-1} \bigcup x_t$
if $(LL_t - LL_{t-1}) > c$
then
$C_t = C_t + 1$
else
$M_t = M_{t-1}$
$A_t = A_{t-1}$
if (the result says 0 but LR says 1)
then d_t is intrusion
else if (the result says 1 but LR says 0)
then d_t is normal
else d_t remains the same from the result
endfor
end

Fig. 1. Pseudo-code for the Anomaly Detection Algorithm

2 Anomaly Detection Algorithm

Intrusion detection problem has been extensively studied in recent years. There are many different approaches for modeling normal and anomalous data, based on which the detections are carried out. A survey and comparison of anomaly detection techniques can be found in[1].

Our goal in this paper is to demonstrate that distributed and adaptive versions of anomaly detection can be implemented using our middleware, GATES. For this purpose, we have chosen an existing algorithm by Eskin [3]. One reason for choosing this algorithm is that many other anomaly detection approaches require training models over clean data. This can lead to problems since online data is not clean and once the anomalies hidden in the data have been detected as normal, further detections will also fail. Eskin's algorithm, in comparison, has the advantage that it can identify anomalies without clean data.

We now briefly describe this algorithm. This method identifies anomalies buried within the dataset. An assumption is made that the number of normal elements in the data set is significantly larger than the number of anomalous elements. The pseudo-code for the algorithm is shown in Figure 1. We use Gaussian Mixture Model (GMM) to represent the distribution of the normal elements in the dataset. This is because it has the property of being able to represent any distribution as long as the number of Gaussians in the mixture is large enough[7]. Further details of the use of EM algorithm to generate GMM can be found in [4]. Once we have the model, anomaly detection begins with first assuming every element is normal. Motivated by the model of anomalies, we use GMM to test each element to determine whether it is an intrusion or not. This is based on the difference of the loglikelihood by treating it as a normal element and as an intrusion. As compared to Eskin's original algorithm, we also use logistic regression[2] to further improve the performance of the algorithm.

3 GATES Middleware and Distributed Anomaly Detection Implementation

In this section, we initially describe the GATES middleware system, and then describe our distributed anomaly detection implementation.

3.1 Overview of the GATES System

GATES (Grid-based AdapTive Execution on Streams) is a middleware that supports the flexible and adaptive analysis of distributed data streams. A key goal is to able to allow the most accurate analysis while still meet the real-time constraint. For this purpose, GATES applies *self-adaptation* algorithm. In summary, GATES has the following features:

- It is designed to use the existing grid standards and tools to the extent possible. Specifically, GATES is built on the Open Grid Services Architecture (OGSA) model and uses the initial version of Globus Toolkits (GT) 3.0's API functions. Therefore, all components of GATES, including applications, exist in the form of Grid services.
- It supports distributed processing of one or more data streams, by facilitating applications that comprise a set of *stages*. For analyzing more than one data stream, at least two stages are required. Each stage accepts data from one or more input streams and outputs zero or more streams. The first stage is applied near sources of individual streams, and the second stage is used for computing the final results. However, based upon the number and types of streams and the available resources, applications can also take more than two steps. GATES's APIs are designed to facilitate specification of such stages.
- It flexibly achieves the best accuracy that is possible while maintaining the real-time constraint on the analysis. To do this, the system monitors the arrival rate at each source, the available computing resources and memory, as well as the available network bandwidth. Then it automatically adjusts

the accuracy of the analysis by tuning the parameter within a certain range specified by the user.

The *self-adaptation* algorithm used in GATES has been evaluated using a number of stream-based data mining applications, including counting samples and finding frequent itemsets in distributed data streams, using data stream processing for computational steering, and clustering evolving data streams[6,5]. Results from the evaluation show that GATES is able to self-adapt effectively, and achieve the highest accuracy possible while maintaining the real-time processing constraint, regardless of the resource availability, network bandwidth, or processing power.

3.2 Real-Time Distributed Intrusion Detection on GATES

The use of GATES middleware can provide two advantages in implementing intrusion detection. First, it can allow a distributed implementation. Many scenarios for intrusion detection involve data arriving at multiple locations. One possible solution for handling such cases is to forward all data to a single node, however, this can result in high communication and computation overheads. Second, it can allow for an *adaptive* implementation, which can trade-off rate of processing and accuracy. Therefore, it can allow the implementation to meet real-time constraints, and allow best accuracy for the given data arrival rate and available computational resources.

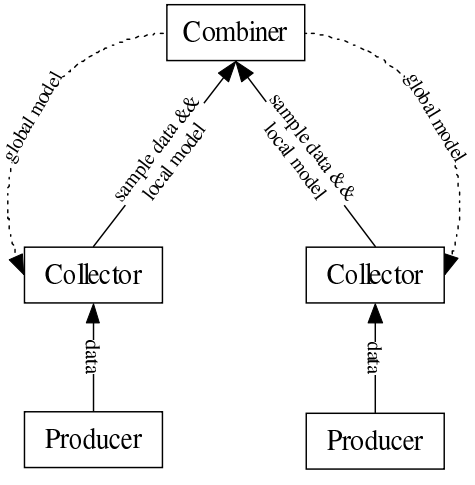

Fig. 2. Communication Topology for the Distributed Intrusion Detection Application

Figure 2 shows the hierarchical structure of the real-time distributed intrusion detection application built on GATES. Our implementation is composed of three stages. The entire procedure of the intrusion detection is divided into a *data*

preprocessing stage, a stage which performs *local model generation* and *detects intrusions*, and a *global model generation* stage. We now describe each of these stages in more details.

Producer is simply the data source, where some initial preprocessing can be performed. We chose 10 out of 41 attributes (7 continuous and 3 categorical) for each network data record. The attributes with the most significant variance are included as they most likely to contain more information for distinguishing intrusions from normal data. The filtered data is sent to the second stage.
Collector first collects the data from the *Producer* and applies the EM algorithm to generate local GMMs. These are then used in the anomaly detection algorithm we described earlier, for detecting local intrusions. Another function of this stage is to forward *samples*, i.e. normal data points which have been detected as local intrusions, to the final stage. The goal is to allow the global model to capture their distribution. Finally, once the global model is generated, it is sent back to this stage. Here, the anomaly detection algorithm is applied again to get the global intrusions.

Note that the global model is improved iteratively, i.e., we approach closer to the true probability distribution after each iteration, and have greater accuracy in intrusion detection.

Combiner generates the global model based on the local model parameters and samples. For GMM, the local model parameters used are the mean vector, covariance matrix for each Gaussian model, and their weights contributing to the mixture. We use Kullback-Leibler(KL)-divergence [8] as the measurement to decide how similar two distributions are. KL-divergence is the most natural comparison measure since it is linearly related to the average loglikelihood of the data generated by one model with respect to the other. It is also a well-behaved differentiable function of the model parameters, unlike the other measures. Hence, we combine two local models if the KL-divergence between them is below a user-defined threshold, in which case, a modification has to be made to the global model. Otherwise, new models would be added in. The Combiner sends back the global model to the Collector to end the processing associated with each iteration.

As we stated earlier, GATES uses programmer declared *adaptation parameters* to achieve self-adaptation. In our implementation, we have used two different adaptation parameters. The first parameter is the rate of sampling between the Collector and the Combiner. If the Collector sends a very large number of samples to the Combiner, it increases the communication as well as the computations at the Combiner stage. On the other hand, a very small number of samples will result in a less accurate global model. Similarly, the convergence threshold for the EM algorithm also impacts the accuracy and the processing rate. The smaller the value of this parameter, the more accurate local model we can get, which also leads to a better global model. However, a small value of this parameter also results in more computations for the EM algorithm to converge, making the Collector overloaded.

Note that the self-adaptation algorithm in GATES can only adjust one parameter at a time. Thus, in our experiments, we keep one of them fixed, while allowing the middleware to adjust the other.

4 Experimental Evaluation

This section presents results from the real-time distributed intrusion detection application built on GATES. We had several goals in our experiments. First, we wanted to demonstrate that distributed implementations can achieve high accuracy. Second, we wanted to evaluate the middleware's ability to adjust the two adaptation parameters, under different conditions. Finally, we also evaluate the improvements in accuracy achieved through logistic regression, which is an area where we had improved Eskin's algorithm.

The dataset we used is KDD-CUP'99 Network Intrusion Detection data, which contains a wide variety of intrusions simulated in a military network environment. It consists of approximately 4,900,000 network connection records with more than 80% as intrusions. Each connection record has 41 attributes, including categorical and continuous ones. According the requirement from the anomaly detection algorithm[3], the majority distribution should have at least 90% of the entire dataset. Therefore, we randomly duplicate the normal data and choose part of the intrusion data, which result in a data set with 335,892 records in total and only 9.04% are intrusions. Each type of intrusion is evenly distributed and comes in a burst.

We conducted our experiments in a Linux cluster. Each node has a Pentium III 933MHz CPU with 512MB of main memory and 300GB local disk space. The interconnection network is a switched 100Mb/s Ethernet. In all our experiments, the number of mixtures used in the centralized anomaly detection algorithm is 3 and the distributed version also results in the same number of mixtures.

4.1 Experiment 1: Adjustable EM Threshold vs. Fixed Sampling Rate

In this experiment, we fix the sampling rates at certain values and let GATES adjust the EM threshold. The data production (arrival) rates vary from 100kb/s, 80kb/s, 50kb/s, 30kb/s to 10kb/s. Figure 3 shows how the EM threshold parameter converges to the ideal values with different data production rates. As expected, when the production rate is small, EM threshold converges to a smaller value since it can have enough time to perform the EM algorithm without being overloaded from its upstream. Also notice that the smallest EM threshold GATES converges to is 0.000012, which is very close to the EM threshold used in the centralized algorithm, 0.00001.

The detailed results, including processing time, detection rate, and false positive rate, are shown in Table 1. A centralized version, with no time constraints, takes 923 seconds, and is able to detect 97.63% of the intrusions. It also has a 8.08% rate of false positives. The best accuracy from the distributed version

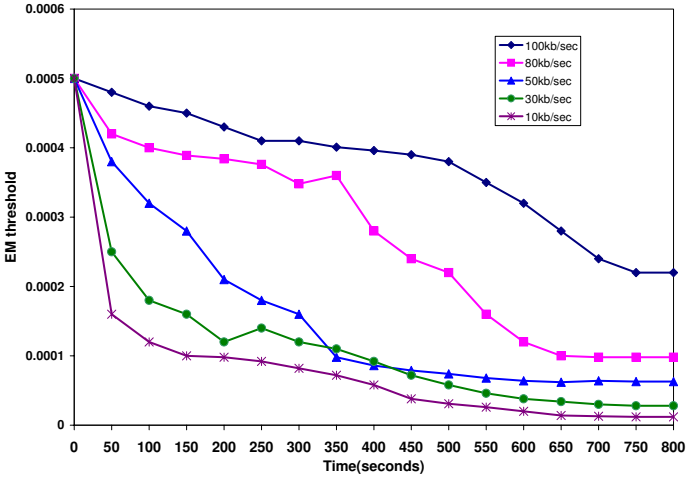

Fig. 3. Adaptation of EM Threshold with Different Data Production Rates

Table 1. Intrusion Detection Results

		Exe. time(sec)	Detection rate	False positive
Centralized		923.0	**97.63%**	**8.08%**
Distributed *Producing rate = 100kb/s*	Sample rate=40%	667.8	82.79%	5.95%
	Sample rate=20%	637.2	91.38%	7.39%
	Sample rate=16%	618.7	86.48%	6.35%
	Sample rate=13%	609.8	83.71%	6.04%
	Sample rate=10%	602.1	82.57%	5.83%
Distributed *Producing rate = 80kb/s*	Sample rate=40%	710.1	84.72%	6.22%
	Sample rate=20%	698.9	92.09%	7.57%
	Sample rate=16%	674.2	88.69%	6.83%
	Sample rate=13%	653.3	86.21%	6.32%
	Sample rate=10%	642.9	84.58%	6.17%
Distributed *Producing rate = 50kb/s*	Sample rate=40%	766.2	88.07%	6.88%
	Sample rate=20%	738.6	92.44%	7.63%
	Sample rate=16%	708.1	90.71%	7.19%
	Sample rate=13%	692.6	89.00%	6.95%
	Sample rate=10%	680.4	87.85%	6.%
Distributed *Producing rate = 30kb/s*	Sample rate=40%	795.6	90.20%	7.22%
	Sample rate=20%	766.7	94.63%	7.78%
	Sample rate=16%	732.1	93.13%	7.67%
	Sample rate=13%	719.8	91.38%	7.55%
	Sample rate=10%	706.9	90.10%	7.10%
Distributed *Producing rate = 10kb/s*	Sample rate=40%	862.1	93.74%	7.72%
	Sample rate=20%	798.9	**95.36%**	**7.90%**
	Sample rate=16%	762.4	95.16%	7.87%
	Sample rate=13%	741.8	94.29%	7.82%
	Sample rate=10%	728.3	93.50%	7.66%

is 95.36%, which is quite close to the accuracy of the centralized version. The best accuracy obtained under other (higher) data rates is at least 91.38%, which shows that the middleware is able to tradeoff processing rates and accuracy effectively.

Two other observations can be made from this table. First, the false positive rate is always a fixed fraction of the detection rate, i.e., the higher the detection rate, the higher is the false positive rate. The false positive rate is not impacted by whether the implementation is centralized or distributed, or the value of the adaptation parameters. Second, the trends between the choice of the sampling rate and detection rate are quite interesting. Across different data production rates, best accuracy is achieved when sampling rate is 20%. Both lower and higher values of sampling rates result in lower detection rates. The reason is as follows. When the sampling rate is higher, the Combiner takes a longer time to compute global models. As a result, the collector operates with an older model for a longer duration of time. On the other hand, when the sampling rate is lower, a small number of samples at the Combiner results in lower quality global models.

Table 2. Improvements Through Logistic Regression

	without L.R.		with L.R.	
	Detection Rate	False Positive	Detection Rate	False Positive
Centralized	92.36%	8.35%	97.63%	8.08%
Producing rate=100kb/s	86.45%	7.64%	91.38%	7.09%
Producing rate=80kb/s	87.12%	7.82%	92.09%	7.57%
Producing rate=50kb/s	87.45%	7.88%	92.44%	7.63%
Producing rate=30kb/s	89.52%	8.04%	94.63%	7.78%
Producing rate=10kb/s	90.21%	8.12%	95.36%	7.90%

We also now evaluate the benefits of using logistic regression. We have implemented logistic regression using three categorical attributes. The results are shown in Table 2. The detection rate increases from 92.36% to 97.63% with the false positive dropping from 8.35% to 8.08% for the centralized version. Comparing the best results from the real-time distributed intrusion detection implementation, we can get 95.36% as the detection rate and 7.90% as the false positive rate, compared with 90.21% and 8.12% without using the logistic regression, respectively. The overall observation is that once a data record fails anomaly test, i.e. either normal data is detected as intrusion or an intrusion is detected as being normal, the categorical attributes, namely, the protocol type, the service information, and the flag, can correct the detection results.

Two other observations from our implementation are shown through Figures 4 and 5. Figure 4 shows that as the processing proceeds, we are having smaller KL-divergence comparing to the true model, namely, the global model generated from our algorithm is closer to the true model, as expected. As we can see from the Figure 5, we are getting better detection rates for each processing round.

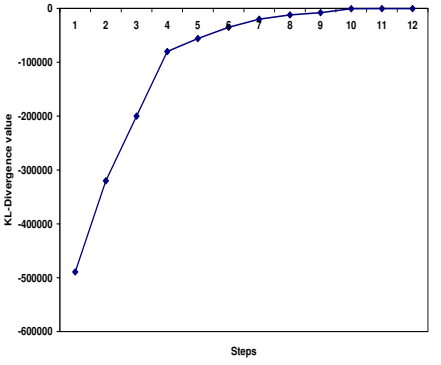

Fig. 4. KL-divergence of the Global Model Compared to the True Model

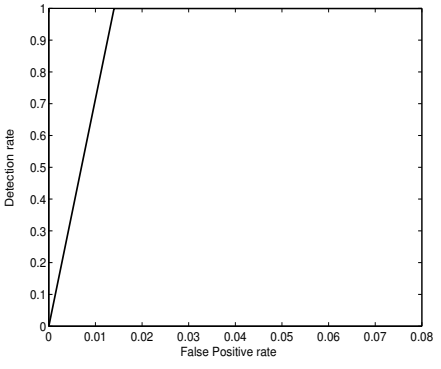

Fig. 5. ROC for the Intrusion Detection Application

We have used a ROC curve in this Figure, which is a graphical representation of the false positive rate versus the detection rate. The reason for our observation is that we have more data to generate the global model.

4.2 Experiment 2: Adjustable Sampling Rate vs. Fixed EM Threshold

The other experiment we carried out involved a fixed EM threshold, and sampling rate as the adaptation parameter. Again, under different data production rates, we observed how the middleware is able to converge to a stable value of the adaptation parameter. The results are shown in Figure 6. As expected,

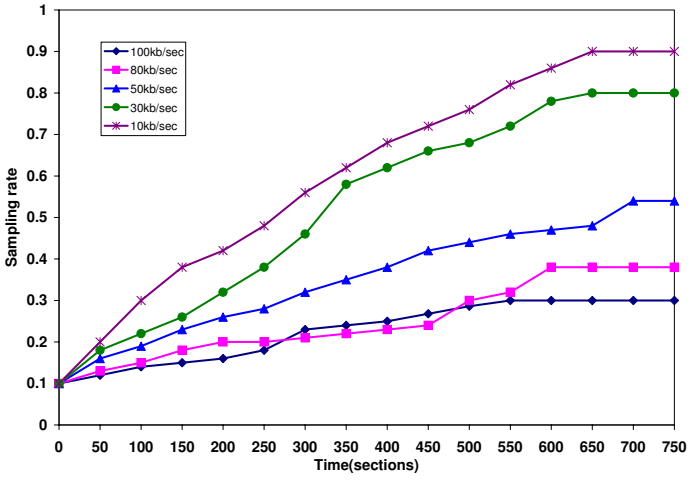

Fig. 6. Adaptation of Sampling Rate with Different Production Rates

higher data production levels result in a smaller sampling rate, and lower data production levels result in a higher sampling rate.

5 Conclusion

This paper has reported an application study using the GATES middleware, which has been developed for supporting grid-based streaming applications. We have focused on the problem of intrusion detection. We have created a distributed and self-adaptive real-time implementation of the algorithm proposed by Eskin using our middleware. The main observations from our experiments are as follows. First, our distributed implementation can achieve detection rates which are very close to the detection rate by a centralized algorithm. Second, our implementation is able to effectively adjust the adaptation parameters.

References

1. C.Warrender, S. Forrest, and B.Pearlmutter. Detecting intrusions using system calls: alternative data models. In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 133–145, 1999.
2. D.Hosmer and S.Lemeshow. *Applied Logistic Regression*. Wiley, New York, 1989.
3. E.Eskin. Anomaly detection over noisy data using learned probability distributions. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 255–262, June 2000.
4. J.A.Belmis. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussion mixture and hidden markov models. In *Technical report*, UC Berkeley, USA, April 1998.
5. L.Chen and G.Agrawal. Supporting self-adaptation in streaming data mining applications. In *Proceedings of IEEE International Parallel & Distributed Processing Symposium(IPDPS)*, April 2006.
6. L.Chen, K.Reddya, and G.Agrawal. Gates: A grid-based middleware for distributed processing of data streams. In *Proceedings of IEEE Conference on High Performance Distributed Computing(HPDC)*, pages 192–201, June 2004.
7. N.Gilardi, T.Melluish, and M.Maignan. Conditional gaussian mixture models for environmental risk mapping. In *Proceedings of the 2002 12th IEEE Workshop on Neural Networks for Signal Processing (NNSP)*, pages 777–786, Sept. 2002.
8. S.Kullback. *Information Theory and Statistics*. Wiley, New York, 1959.