Chapter 13

A SURVEY OF DISTRIBUTED MINING OF DATA STREAMS

Srinivasan Parthasarathy
Department of Computer Science and Engineering
The Ohio State University
srini@cse.ohio-state.edu

Amol Ghoting
Department of Computer Science and Engineering
The Ohio State University
ghoting@cse.ohio-state.edu

Matthew Eric Otey
Department of Computer Science and Engineering
The Ohio State University
otey@cse.ohio-state.edu

Abstract With advances in data collection and generation technologies, organizations and researchers are faced with the ever growing problem of how to manage and analyze large dynamic datasets. Environments that produce streaming sources of data are becoming common place. Examples include stock market, sensor, web click stream, and network data. In many instances, these environments are also equipped with multiple distributed computing nodes that are often located near the data sources. Analyzing and monitoring data in such environments requires data mining technology that is cognizant of the mining task, the distributed nature of the data, and the data influx rate. In this chapter, we survey the current state of the field and identify potential directions of future research.

1. Introduction

Advances in technology have enabled us to collect vast amounts of data from various sources, whether they be from experimental observations, simulations,

sensors, credit card transactions, or from networked systems. To benefit from these enhanced data collecting capabilities, it is clear that semi-automated interactive techniques such as data mining should be employed to process and analyze the data. It is also desirable to have interactive response times to client queries, as the process is often iterative in nature (with a human in the loop). The challenges to meet these criteria are often daunting as detailed next.

Although inexpensive storage space makes it possible to maintain vast volumes of data, accessing and managing the data becomes a performance issue. Often one finds that a single node is incapable of housing such large datasets. Efficient and adaptive techniques for data access, data storage and communication (if the data sources are distributed) are thus necessary. Moreover, data mining becomes more complicated in the context of dynamic databases, where there is a constant influx of data. Changes in the data can invalidate existing patterns or introduce new ones. Re-executing the algorithms from scratch leads to large computational and I/O overheads. These two factors have led to the development of distributed algorithms for analyzing streaming data which is the focus of this survey article.

Many systems use a centralized model for mining multiple data streams [2]. Under this model the distributed data streams are directed to one central location before they are mined. A schematic diagram of a centralized data stream mining system is presented in Figure 13.1. Such a model of computation is limited in several respects. First, centralized mining of data streams can result in long response time. While distributed computing resources may be available, they are not fully utilized. Second, central collection of data can result in heavy traffic over critical communication links. If these communication links have limited network bandwidth, network I/O may become a performance bottleneck. Furthermore, in power constrained domains such as sensor networks, this can result in excessive power consumption due to excessive data communication.

To alleviate the aforementioned problems, several researchers have proposed a model that is aware of the distributed sources of data, computational resources, and communication links. A schematic diagram of such a distributed stream mining system is presented in Figure 13.1 and can be contrasted with the centralized model. In the model of distributed stream mining, instead of offloading the data to one central location, the distributed computing nodes perform parts of the computation close to the data, while communicating the local models to a central site as and when needed. Such an architecture provides several benefits. First, by using distributed computing nodes, it allows the derivation of a *greater degree of parallelism*, thus reducing response time. Second, as only local models need to be communicated, communication can potentially be reduced, *improving scalability*, and reducing power consumption in power constrained domains.

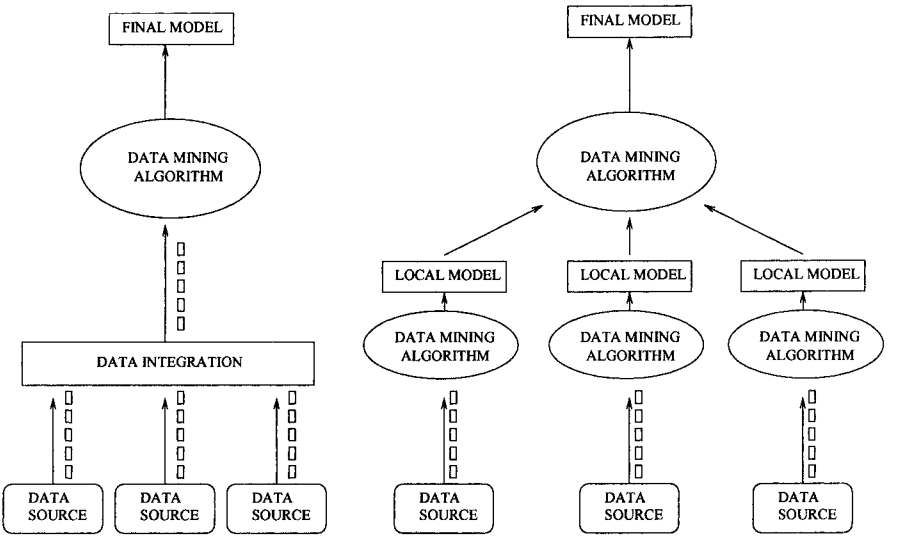

Figure 13.1. Centralized Stream Processing Architecture (left) Distributed Stream Processing Architecture (right)

This chapter presents a brief overview of distributed stream mining algorithms, systems support, and applications, together with emerging research directions. We attempt to characterize and classify these approaches as to whether they belong in the centralized model or the distributed model. The rest of this chapter is organized as follows. First, we present distributed stream mining algorithms for various mining tasks such as outlier detection, clustering, frequent itemset mining, classification, and summarization. Second, we present an overview of distributed stream mining in resource constrained domains. Third, we summarize research efforts on building systems support for facilitating distributed stream mining. Finally, we conclude with emerging research directions in distributed stream mining.

2. Outlier and Anomaly Detection

The goal in outlier or anomaly detection is to find data points that are most different from the remaining points in the data set [4]. Most outlier detection algorithms are schemes in which the distance between every pair of points is calculated, and the points most distant from all other points are marked as outliers [29]. This is an $O(n^2)$ algorithm that assumes a static data set. Such approaches are difficult to extend to distributed streaming data sets. Points in these data sets arrive at multiple distributed end-points, which may or may not be compute nodes, and must be processed incrementally. Such constraints lead us away from purely distance-based approaches, and towards more heuristic

techniques. Note that the central issue in many anomaly detection systems, is to identify anomalies in *real-time* or as close to real time as possible thus making it a natural candidate for many streaming applications. Moreover, often times the data is produced at disparate sites making distributed stream mining a natural fit for this domain. In this section we review the work in outlier or anomaly detection most germane to distributed stream mining.

Various application-specific approaches to outlier/anomaly detection have been proposed in the literature. An approach [39] has been presented for distributed deviation detection in sensor networks. This approach is tailored to the sensor network domain and targets misbehaving sensors. The approach maintains density estimates of values seen by a sensor, and flags a sensor to be a misbehaving sensor if its value deviates significantly from the previously observed values. This computation is handled close to the sensors in a distributed fashion, with only results being reported to the central server as and when needed.

One of the most popular applications of distributed outlier detection is that of network intrusion detection. Recent trends have demanded a distributed approach to intrusion detection on the Internet. The first of these trends is a move towards distributed intrusions and attacks, that is to say, intrusions and attacks originating from a diverse set of hosts on the internet. Another trend is the increasing heterogeneous nature of the Internet, where different hosts, perhaps residing in the same subnetwork have differing security requirements. For example, there have been proposals for distributed firewalls [20] for fulfilling diverse security requirements. Also, the appearance of mobile and wireless computing has created dynamic network topologies that are difficult, if not impossible, to protect from a centralized location. Efficient detection and prevention of these attacks requires distributed nodes to collaborate. By itself, a node can only collect information about the state of the network immediately surrounding it, which may be insufficient to detect distributed attacks. If the nodes collaborate by sharing network audit data, host watch lists, and models of known network attacks, each can construct a better global model of the network.

Otey *et al* [36], present a distributed outlier detection algorithm targeted at distributed online streams, specifically to process network data collected at distributed sites. Their approach finds outliers based on the number of attribute dependencies violated by a data point in continuous, categorical, and mixed attribute spaces. They maintain an in-memory structure that succinctly summarizes the required dependency information. In order to find exact outliers in a distributed streaming setting, the in-memory summaries would need to be exchanged frequently. These summaries can be large, and consequently, in a distributed setting, each distributed computing node only exchanges local outliers with the other computing nodes. A point is deemed to be a global outlier if every distributed node believes it to be an outlier based on its local model

of normalcy. While such an approach will only find approximate outliers, the authors show that this heuristic works well in practice. While the authors report that to find exact outliers they need to exchange a large summary which leads to excessive communication, it could be possible to exchange only decisive parts of the summary, instead of the entire summary, in order to more accurately detect the true outliers. Furthermore, their in-memory summaries are large, as they summarize a large amount of dependency information. Reducing this memory requirement could potentially allow the use of this algorithm in resource-constrained domains.

EMERALD is an approach for collaborative intrusion detection for large networks within an enterprise [42]. This approach allows for distributed protection of the network through a hierarchy of surveillance systems that analyze network data at the service, domain, and enterprise-wide levels. However, EMERALD does not provide mechanisms for allowing different organizations to collaborate. Locasto *et al* [33] examine techniques that allow different organizations to do such collaboration for enhanced network intrusion detection. If organizations can collaborate, then each can build a better model of global network activity, and more precise models of attacks (since they have more data from which to estimate the model parameters). This allows for better characterization and prediction of attacks. Collaboration is achieved through the exchange of Bloom filters, each of which encodes a list of IP addresses of suspicious hosts that a particular organization's Intrusion Detection System (IDS) has detected, as well as the ports which these suspicious hosts have accessed. The use of Bloom filters helps both to keep each collaborating organization's information confidential and to reduce the amount of data that must be exchanged.

A major limitation of this approach is that information exchanged may not be sufficient to identify distributed attacks. For example, it is possible that an attack may originate from a number of hosts, none of which are suspicious enough to be included on any organization's watch list. However, the combined audit data collected by each organization's IDS may be sufficient to detect that attack. To implement such a system, two problems must be addressed. The first is that each organization may collect disjoint sets of features. Collaborating organizations must agree beforehand on a set of common features to use. Some ideas for common standards for intrusion detection have been realized with the Common Intrusion Detection Framework (CIDF) [31]. The second problem is that of the privacy of each organization's data. It may not be practical to use Bloom filters to encode a large set of features. However, techniques do exist for privacy-preserving data mining [28, 23, 32] that will allow organizations to collaborate without compromising the privacy of their data.

There have been other approaches for detecting distributed denial-of-service attacks. Lee *et al* have proposed a technique for detecting novel and distributed intrusions based on the aforementioned CIDF [31]. The approach not only

allows nodes to share information with which they can detect distributed attacks, but also allows them to distribute models of novel attacks. Yu *et al* propose a middleware-based approach to prevent distributed denial of service attacks [45]. Their approach makes use of Virtual Private Operation Environments (VPOE) to allow devices running the middleware to collaborate. These devices can act as firewalls or network monitors, and their roles can change as is necessary. Each device contains several modules, including an attack detection module, a signaling module for cooperating with other devices, and policy processing modules.

Some work in network intrusion detection has been done in the domain of mobile ad hoc networks (MANETs) [47, 18], where nodes communicate over a wireless medium. In MANETs, the topology is dynamic, and nodes must cooperate in order to route messages to their proper destinations. Because of the open communication medium, dynamic topology, and cooperative nature, MANETs are especially prone to network intrusions, and present difficulties for distributed intrusion detection.

To protect against intrusions, Zhang *at al* have proposed several intrusion detection techniques [46, 47]. In their proposed architecture, each node in the network participates in detection and response, and each is equipped with a local detection engine and a cooperative detection engine. The local detection engine is responsible for detecting intrusions from the local audit data. If a node has strong evidence that an intrusion is taking place, it can initiate a response to the intrusion. However, if the evidence is not sufficiently strong, it can initiate a global intrusion detection procedure through the cooperative detection engine. The nodes only cooperate by sharing their detection states, not their audit data, and so it is difficult for each node to build an accurate global model of the network with which to detect intrusions. In this case, intrusions detectable only at the global level (e.g. ip sweeps) will be missed. However, the authors do point out that they only use local data since the remote nodes may be compromised and their data may not be trustworthy.

In another paper [18], Huang and Lee present an alternative approach to intrusion detection in MANETs. In this work, the intrusions to be detected are attacks against the structure of the network itself. Such intrusions are those that corrupt routing tables and protocols, intercept packets, or launch network-level denial-of-service attacks. Since MANETs typically operate on battery power, it may not be cost effective for each node to constantly run its own intrusion detection system, especially when there is a low threat level. The authors propose that a more effective approach would be for a cluster of nodes in a MANET to elect one node as a monitor (the *clusterhead*) for the entire cluster. Using the assumption that each node can overhear network traffic in its transmission range, and that the other cluster members can provide (some of) the features (since the transmission ranges of the clusterhead and the other

cluster members may not overlap, the other cluster members may have statistics on portions of the cluster not accessible to the clusterhead), the clusterhead is responsible for analyzing the flow of packets in its cluster in order to detect intrusions and initiate a response. In order for this intrusion detection approach to be effective, the election of the clusterhead must be fair, and each clusterhead must serve an equal amount of time. The first requirement ensures that the election of the clusterhead is unbiased (i.e. a compromised node cannot tilt the election in its favor), and the second requirement ensures that a compromised node cannot force out the current clusterhead nor remain as clusterhead for an unlimited period of time. There is a good division of labor, as the clusterhead is the only member of the cluster that must run the intrusion detection system; the other nodes need only collect data and send it to the clusterhead. However, a limitation of this approach is that not all intrusions are visible at the global level, especially given the feature set the detection system uses (statistics on the network topology, routes, and traffic). Such local intrusions include exploits of services running on a node, which may only be discernible using the content of the traffic.

3. Clustering

The goal in clustering is to partition a set of points into groups such that points within a group are similar in some sense and points in different groups are dissimilar in the same sense. In the context of distributed streams, one would want to process the data streams in a distributed fashion, while communicating the summaries, and to arrive at global clustering of the data points. Guha *et al* [17], present an approach for clustering data streams. Their approach produces a clustering of the points seen using small amounts of memory and time. The summarized data consists of the cluster centers together with the number of points assigned to that cluster. The k-median algorithm is used as the underlying clustering mechanism. The resulting clustering is a constant factor approximation of the true clustering. As has been shown in [16], this algorithm can be easily extended to operate in a distributed setting. Essentially, clusterings from each distributed site can be combined and clustered to find the global clustering with the same approximation factor. From a qualitative stand point, in many situations, k-median clusters are known to be less desirable than those formed by other clustering techniques. It would be interesting to see if other clustering algorithms that produce more desirable clusterings can be extended with the above methodology to operate over distributed streams.

Januzaj *et al* [21], present a distributed version of the density-based clustering algorithm, DBSCAN. Essentially, each site builds a local density-based clustering, and then communicates a summary of the clustering to a central site. The central site performs a density-based clustering on the summaries

obtained from all sites to find a global clustering. This clustering is relayed back to the distributed sites that update their local clusterings based on the discovered global clustering. While this approach is not capable of processing dynamic data, in [13], the authors have shown that density based clustering can be performed incrementally. Therefore, a distributed and incremental version of DBSCAN can potentially be devised. However, like the distributed version presented by Januzaj *et al*, we cannot provide a guarantee on the quality of the result.

Beringer and Hullermeir consider the problem of clustering parallel data streams [5]. Their goal is to find correlated streams as they arrive synchronously. The authors represent the data streams using exponentially weighted sliding windows. The discrete Fourier transform is computed incrementally, and k-Means clustering is performed in this transformed space at regular intervals of time. Data streams belonging to the same cluster are considered to be correlated. While the processing is centralized, the approach can be tailored to correlate distributed data streams. Furthermore, the approach is suitable for online streams. It is possible that this approach can be extended to a distributed computing environment. The Fourier coefficients can be exchanged incrementally and aggregated locally to summarize remote information. Furthermore, one can potentially produce approximate results by only exchanging the significant coefficients.

4. Frequent itemset mining

The goal in frequent itemset mining is to find groups of items or values that co-occur frequently in a transactional data set. For instance, in the context of market data analysis, a frequent two itemset could be $\{beer, chips\}$, which means that people frequently buy beer and chips together. The goal in frequent itemset mining is to find all itemsets in a data set that occur at least x number of times, where x is the minimum support parameter provided by the user.

Frequent itemset mining is both CPU and I/O intensive, making it very costly to completely re-mine a dynamic data set any time one or more transactions are added or deleted. To address the problem of mining frequent itemsets from dynamic data sets, several researchers have proposed incremental techniques [10, 11, 14, 30, 43, 44]. Incremental algorithms essentially re-use previously mined information and try to combine this information with the fresh data to efficiently compute the new set of frequent itemsets. However, it can be the case that the database may be distributed over multiple sites, and is being updated at different rates at each site, which requires the use of distributed asynchronous frequent itemset mining techniques.

Otey *et al* [38], present a distributed incremental algorithm for frequent itemset mining. The approach is capable of incrementally finding maximal

frequent itemsets in dynamic data. Maximal frequent itemsets are those that do not have any frequent supersets, and the set of maximal frequent itemsets determines the complete set of frequent itemsets. Furthermore, it is capable of mining frequent itemsets in a distributed setting. Distributed sites can exchange their local maximal frequent itemsets to obtain a superset of the global maximal frequent itemsets. This superset is then exchanged between all nodes so that their local counts may be obtained. In the final round of communication, a reduction operation is performed to find the exact set of global maximal frequent itemsets.

Manku and Motwani [35], present an algorithm for mining frequent itemsets over data streams. In order to mine all frequent itemsets in constant space, they employ a down counting approach. Essentially, they update the support counts for the discovered itemsets as the data set is processed. Furthermore, for all the discovered itemsets, they decrement the support count by a specific value. As a result, itemsets that occur rarely will have their count set to zero and will be eventually eliminated from list. If they reappear later, their count is approximated. While this approach is tailored to data streams, it is not distributed. The methodology proposed in [38] can potentially be applied to this algorithm to process distributed data streams.

Manjhi *et al* [34], extend Manku and Motwani's approach to find frequent items in the union of multiple distributed streams. The central issue is how to best manage the degree of approximation performed as partial synopses from multiple nodes are combined. They characterize this process for hierarchical communication topologies in terms of a precision gradient followed by synopses as they are passed from leaves to the root and combined incrementally. They studied the problem of finding the optimal precision gradient under two alternative and incompatible optimization objectives: (1) minimizing load on the central node to which answers are delivered, and (2) minimizing worst-case load on any communication link. While this approach targets frequent items only, it would be interesting to see if it can be extended to find frequent itemsets.

5. Classification

Hulten and Domingos [19], present a one-pass decision tree construction algorithm for streaming data. They build a tree incrementally by observing data as it streams in and splitting a node in the tree when a sufficient number of samples have been seen. Their approach uses the Hoeffding inequality to converge to a sample size. Jin and Agrawal revisit this problem and present solutions that speed up split point calculation as well as reduce the desired sample size to achieve the same level of accuracy [22]. Both these approaches are not capable of processing distributed streams.

Kargupta and Park present an approach for aggregating decision trees constructed at distributed sites [26]. As each decision tree can be represented as a numeric function, the authors propose to transmit and aggregate these trees by using their Fourier representations. They also show that the Fourier-based representation is suitable for approximating a decision tree, and thus, suitable for transmission in bandwidth-limited mobile environments. Coupled with a streaming decision tree construction algorithm, this approach should be capable of processing distributed data streams.

Chen *et al* [8], present a collective approach to mine Bayesian networks from distributed heterogeneous web-log data streams. In their approach, they learn a local Bayesian network at each site using the local data. Then each site identifies the observations that are most likely to be evidence of coupling between local and non-local variables and transmits a subset of these observations to a central site. Another Bayesian network is learned at the central site using the data transmitted from the local sites. The local and central Bayesian networks are combined to obtain a collective Bayesian network, that models the entire data. This technique is then suitably adapted to an online Bayesian learning technique, where the network parameters are updated sequentially based on new data from multiple streams. This approach is particularly suitable for mining applications with distributed sources of data streams in an environment with non-zero communication cost (e.g. wireless networks).

6. Summarization

Bulut and Singh [6], propose a novel technique to summarize a data stream incrementally. The summaries over the stream are computed at multiple resolutions, and together they induce a unique Wavelet-based approximation tree. The resolution of approximations increases as we move from the root of the approximation tree down to its leaf nodes. The tree has space complexity $O(logN)$, where N denotes the current size of the stream. The amortized processing cost for each new data value is $O(1)$. These bounds are currently the best known for the algorithms that work under a biased query model where the most recent values are of a greater interest. They also consider the scenario in which a central source site summarizes a data stream at multiple resolutions. The clients are distributed across the network and pose queries. The summaries computed at the central site are cached adaptively at the clients. The access pattern, i.e. reads and writes, over the stream results in multiple replication schemes at different resolutions. Each replication scheme expands as the corresponding read rate increases, and contracts as the corresponding write rate increases. This adaptive scheme minimizes the total communication cost and the number of inter-site messages. While the summarization process is centralized, it can potentially

be used to summarize distributed streams at distributed sites by aggregating wavelet coefficients.

The problem of pattern discovery in a large number of co-evolving streams has attracted much attention in many domains. Papadimitriou *et al* introduce SPIRIT (Streaming Pattern dIscoveRy in multIple Time-series) [40], a comprehensive approach to discover correlations that effectively and efficiently summarize large collections of streams. The approach uses very less memory and both its memory requirements and processing time are independent of the stream length. It scales linearly with the number of streams and is adaptive and fully automatic. It dynamically detects changes (both gradual and sudden) in the input streams, and automatically determines the number of hidden variables. The correlations and hidden variables discovered have multiple uses. They provide a succinct summary to the user, they can help to do fast forecasting and detect outliers, and they facilitate interpolations and handling of missing values. While the algorithm is centralized, it targets multiple distributed streams. The approach can potentially be used to summarize streams arriving at distributed sites.

Babcock and Olston [3], study a useful class of queries that continuously report the k largest values obtained from distributed data streams ("top-k monitoring queries"), which are of particular interest because they can be used to reduce the overhead incurred while running other types of monitoring queries. They show that transmitting entire data streams is unnecessary to support these queries. They present an alternative approach that significantly reduces communication. In their approach, arithmetic constraints are maintained at remote stream sources to ensure that the most recently provided top-k answer remains valid to within a user-specified error tolerance. Distributed communication is only necessary on the occasion when constraints are violated.

7. Mining Distributed Data Streams in Resource Constrained Environments

Recently, there has been a lot of interest in environments that demand distributed stream mining where resources are constrained. For instance, in the sensor network domain, due to energy consumption constraints, excessive communication is undesirable. One can potentially perform more computation and less communication to perform the same task with reduced energy consumption. Consequently, in such scenarios, data mining algorithms (specifically clustering and classification) with tunable computation and communication requirements are needed [24, 39].

A similar set of problems have recently been looked at in the network intrusion detection community. Here, researchers have proposed to offload computation related to monitoring and intrusion detection on to the network interface

card (NIC) [37] with the idea of enhancing reliability and reducing the constraints imposed on the host processing environment. Initial results in this domain convey the promise of this area but there are several limiting criteria in current generation NICs (e.g. programming model, lack of floating point operations) that may be alleviated in next generation NICs.

Kargupta *et al* present Mobimine [27], a system for intelligent analysis of time-critical data using a Personal Data Assistant (PDA). The system monitors stock market data and signals interesting stock behavior to the user. Stocks are interesting if they may positively or negatively affect the stock portfolio of the user. Furthermore, to assist in the user's analysis, they transmit classification trees to the user's PDA using the Fourier spectrum-based approach presented earlier. As discussed previously, this Fourier spectrum-based representation is well suited to environments that have limited communication bandwidth.

The Vehicle Data Stream Mining System (VEDAS) [25], is a mobile and distributed data stream mining/monitoring application that taps into the continuous stream of data generated by most modern vehicles. It allows continuous on-board monitoring of the data streams generated by the moving vehicles, identifying the emerging patterns, and reporting them to a remote control center over a low-bandwidth wireless network connection. The system offers many possibilities such as real-time on-board health monitoring, drunk-driving detection, driver characterization, and security related applications for commercial fleet management. While there has been initial work in such constrained environments, we believe that there is still a lot to be done in this area.

8. Systems Support

A distributed stream mining system can be complex. It typically consists of several sub-components such as the mining algorithms, the communication sub-system, the resource manager, the scheduler, etc. A successful stream mining system must adapt to the dynamics of the data and best use the available set of resources and components. In this section, we will briefly summarize efforts that target the building of system support for resource-aware distributed processing of streams.

When processing continuous data streams, data arrival can be bursty, and the data rate may fluctuate over time. Systems that seek to give rapid or real-time query responses in such an environment must be prepared to deal gracefully with bursts in data arrival without compromising system performance. Babcock *et al* [1] show that the choice of an operator scheduling strategy can have significant impact on the run-time system memory usage. When data streams are bursty, the choice of an operator scheduling strategy can result in significantly high run-time memory usage and poor performance. To minimize memory utilization at peak load, they present Chain scheduling, an adaptive, load-aware

scheduling of query operators to minimize resource consumption during times of peak load. This operator scheduling strategy for data stream systems is near-optimal in minimizing run-time memory usage for single-stream queries involving selections, projections, and foreign-key joins with stored relations. At peak load, the scheduling strategy selects an operator path (a set of consecutive operators) that is capable of processing and freeing the maximum amount of memory per unit time. This in effect results in the scheduling of operators that together are both selective and have a high aggregate tuple processing rate.

The aforementioned scheduling strategy is not targeted at the processing of distributed streams. Furthermore, using the Chain operator scheduling strategy has an adverse affect on response time and is not suitable for data mining applications that need to provide interactive performance even under peak load. In order to mine data streams, we need a scheduling strategy that supports both response time and memory-aware scheduling of operators. Furthermore, when scheduling a data stream mining application with dependent operators in a distributed setting, the scheduling scheme should not need to communicate a significant amount of state information. Ghoting and Parthasarathy [16], propose an adaptive operator scheduling technique for mining distributed data streams with response time guarantees and bounded memory utilization. The user can tune the application to the desired level of interactivity, thus facilitating the data mining process. They achieve this through a step-wise degradation in response time beginning from a schedule that is optimal in terms of response time. This sacrifice in response time is used towards optimal memory utilization. After an initial scheduling decision is made, changes in system state may force a reconsideration of operator schedules. The authors show that a decision as to whether a local state change will affect the global operator schedule can be made locally. Consequently, each local site can proceed independently, even under minor state changes, and a global assignment is triggered only when it is actually needed.

Plale considers the problem of efficient temporal-join processing in a distributed setting [41]. In this work, the author's goal is to optimize the join processing of event streams to efficiently determine sets of events that occur together. The size of the join window cannot be determined apriori as this may lead to missed events. The author proposes to vary the size of the join window depending on the rate of the incoming stream. The rate of the incoming stream gives a good indication of how many previous events on the stream can be dropped. Reducing the window size also helps reduce memory utilization. Furthermore, instead of forwarding events into the query processing engine on a first-come first-serve basis, the author proposes to forward the earliest event first to further improve performance, as this facilitates the earlier determination of events that are a part of the join result.

Chen *et al* present GATES [7], a middleware for processing distributed data streams. This middleware targets data stream processing in a grid setting and is built on top of the Open Grid Services Architecture. It provides a high level interface that allows one to specify a stream processing algorithm as a set of pipelined stages. One of the key design goals of GATES is to support self adaption under changing conditions. To support self adaptation, the middleware changes one or more of the sampling rate, the summary structure size, or the algorithm used, based on changing conditions of the data stream. For instance, if the stream rate increases, the system reduces the sampling rate accordingly to maintain a real-time response. If we do not adapt the sampling rate, we could potentially face increasing queue sizes, resulting in poor performance. To support self adaptation, the programmer needs to provide the middleware with parameters that allow it to tune the application at runtime. The middleware builds a simple performance model that allows it to predict how parameter changes help in performance adaptation in a distributed setting.

Chi *et al* [12] present a load shedding scheme for mining multiple data streams, although the computation is not distributed. They assume that the task of reading data from the stream and building feature values is computationally expensive and is the bottleneck. Their strategies decide on how to expend limited computation for building feature values for data on multiple streams. They decide on whether to drop a data item on the stream based on the historic utility of the items produced by the stream. If they choose not to build feature values for a data item, they simply predict feature values based on historical data. They use finite memory Markov chains to make such predictions. While the approach presented by the authors is centralized, load shedding decisions can be trivially distributed.

Conclusions and Future Research Directions

In this chapter, we presented a summary of the current state-of-the-art in distributed data stream mining. Specifically, algorithms for outlier detection, clustering, frequent itemset mining, classification, and summarization were presented. Furthermore, we briefly described related applications and systems support for distributed stream mining.

First, the distributed sources of data that need to be mined are likely to span multiple organizations. Each of these organizations may have heterogeneous computing resources. Furthermore, the distributed data will be accessed by multiple analysts, each potentially desiring the execution of a different mining task. The various distributed stream mining systems that have been proposed to date do not take the variability in the tasks and computing resources into account. To facilitate execution and deployment in such settings, a plug and play system design that is cognizant of each organization's privacy is necessary.

A framework in which services are built on top of each other will facilitate rapid application development for data mining. Furthermore, these systems will need to be integrated with existing data grid and knowledge grid infrastructures [9] and researchers will need to design middleware to support this integration.

Second, next generation computing systems for data mining are likely to be built using off-the-shelf CPUs connected using a high bandwidth interconnect. In order to derive high performance on such systems, stream mining algorithms may need to be redesigned. For instance, next generation processors are likely to have multiple-cores on chip. As has been shown previously [15], data mining algorithms are adversely affected by the memory-wall problem. This problem will likely be exacerbated on future multi-core architectures. Therefore, stream mining algorithms at each local site will need to be redesigned to derive high performance on next generation architectures. Similarly, with innovations in networking technologies, designs that are cognizant of high performance interconnects (like Infiniband) will need to be investigated.

Third, as noted earlier, in many instances, environments that demand distributed stream mining are resource constrained. This in turn requires the development of data mining technology that is tailored to the specific execution environment. Various tradeoffs, e.g. energy vs. communication, communication vs. redundant computation etc., must be evaluated on a scenario-by-scenario basis. Consequently, in such scenarios, data mining algorithms with tunable computation and communication requirements will need to be devised. While initial forays in this domain have been made, a systematic evaluation of the various design tradeoffs even for a single application domain has not been done. Looking further into the future, it will be interesting to evaluate if based on specific solutions a more abstract set of interfaces can be developed for a host of application domains.

Fourth, new applications for distributed data stream mining are on the horizon. For example, RFID (radio frequency identification) technology is expected to significantly improve the efficiency of business processes by allowing automatic capture and identification. RFID chips are expected to be embedded in a variety of devices, and the captured data will likely be ubiquitous in the near future. New applications for these distributed streaming data sets will arise and application specific data mining technology will need to be designed.

Finally, over the past few years, several stream mining algorithms have been proposed in the literature. While they are capable of operating in a centralized setting, many are not capable of operating in a distributed setting and cannot be trivially extended to do so. In order to obtain exact or approximate (bounded) results in a distributed setting, the amount of state information that needs to be exchanged is usually excessive. To facilitate distributed stream mining algorithm design, instead of starting from a centralized solution, one needs to start with a distributed mind-set right from the beginning. Statistics or summaries

that can be efficiently maintained in a distributed and incremental setting should be designed and then specific solutions that use these statistics should be devised. Such a design strategy will facilitate distributed stream mining algorithm design.

References

[1] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain: Operator scheduling for memory minimization in data stream systems. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2003.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, 2002.

[3] B. Babcock and C. Olston. Distributed top k monitoring. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2003.

[4] V. Barnett and T. Lewis. *Outliers in Statistical Data*. John Wiley and Sons, 1994.

[5] J. Beringer and E. Hullermeier. Online clustering of parallel data streams. *Data and Knowledge Engineering*, 2005.

[6] A. Bulut and A. Singh. SWAT: Hierarchical stream summarization in large networks. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2003.

[7] L. Chen, K. Reddy, and G. Agrawal. GATES: A grid-based middleware for processing distributed data streams. In *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*, 2004.

[8] R. Chen, D. Sivakumar, and H. Kargupta. An approach to online bayesian network learning from multiple data streams. In *Proceedings of the International Conference on Principles of Data Mining and Knowledge Discovery*, 2001.

[9] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific data sets, 2001.

[10] D. Cheung, J. Han, V. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 1996.

[11] D. Cheung, S. Lee, and B. Kao. A general incremental technique for maintaining discovered association rules. In *Proceedings of the International Conference on Database Systems for Advanced Applications*, 1997.

[12] Y. Chi, P. Yu, H. Wang, and R. Muntz. Loadstar: A load shedding scheme for classifying data streams. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2005.

[13] M. Ester, H. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1998.

[14] V. Ganti, J. Gehrke, and Raghu Ramakrishnan. Demon–data evolution and monitoring. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2000.

[15] A. Ghoting, G. Buehrer, S. Parthasarathy, D. Kim, A. Nguyen, Y. Chen, and P. Dubey. A characterization of data mining algorithms on a modern processor. In *Proceedings of the ACM SIGMOD Workshop on Data Management on New Hardware*, 2005.

[16] A. Ghoting and S. Parthasarathy. Facilitating interactive distributed data stream processing and mining. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, 2004.

[17] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the Symposium on Foundations of Computer Science(FOCS)*, 2000.

[18] Yi-An Huang and Wenke Lee. A cooperative intrusion detection system for ad hoc networks. In *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*, 2003.

[19] G. Hulten and P. Domingos. Mining high speed data streams. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2000.

[20] Sotiris Ioannidis, Angelos D. Keromytis, Steven M. Bellovin, and Jonathan M. Smith. Implementing a distributed firewall. In *ACM Conference on Computer and Communications Security*, 2000.

[21] E. Januzaj, H. Kriegel, and M. Pfeifle. DBDC: Density based distributed clustering. In *Proceedings of the International Conference on Extending Data Base Technology (EDBT)*, 2004.

[22] R. Jin and G. Agrawal. Efficient decision tree construction on streaming data. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.

[23] M. Kantarcioglu and C. Clifton. Privacy-preserving distributed mining of association rules on horizontally partitioned data. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2002.

[24] H. Kargupta. Distributed data mining for sensor networks. In *Tutorial presented at ECML/PKDD*, 2004.

[25] H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa, and D. Handy. Vedas: A mobile and distributed data stream mining system for real-time vehicle monitoring. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, 2004.

[26] H. Kargupta and B. Park. A fourier spectrum based approach to represent decision trees for mining data streams in mobile environments. *IEEE Transactions on Knowledge and Data Engineering*, 2004.

[27] H. Kargupta, B. Park, S. Pittie, L. Liu, D. Kushraj, and K. Sarkar. MobiMine: Monitoring the stock market from a PDA. *SIGKDD Explorations*, 2002.

[28] Hillol Kargupta, Souptik Datta, Qi Wang, and Krishnamoorthy Sivakumar. On the privacy preserving properties of random data perturbation techniques. In *Proceedings of the International Conference on Data Mining (ICDM)*, 2003.

[29] E. Knorr and R. T. Ng. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1998.

[30] S. Lee and D. Cheung. Maintenance of discovered association rules: When to update? In *Proceedings of the Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1997.

[31] Wenke Lee, Rahul A. Nimbalkar, Kam K. Yee, Sunil B. Patil, Pragneshkumar H. Desai, Thuan T. Tran, and Salvatore J. Stolfo. A data mining and CIDF based approach for detecting novel and distributed intrusions. *Lecture Notes in Computer Science*, 2000.

[32] Yehuda Lindell and Benny Pinkas. Privacy preserving data mining. *Lecture Notes in Computer Science*, 1880:36, 2000.

[33] M. Locasto, J. Parekh, S. Stolfo, A. Keromytis, T. Malkin, and V. Misra. Collaborative distributed intrusion detection. Technical report, Columbia University, 2004.

[34] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2005.

[35] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2002.

[36] M. Otey, A. Ghoting, and S. Parthasarathy. Fast distributed outlier detection in mixed attribute data sets. *Data Mining and Knowledge Discovery Journal*, 2006.

[37] M. Otey, S. Parthasarathy, A. Ghoting, G. Li, S. Narravula, and D. Panda. Towards nic-based instrusion detection. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2003.

[38] M. Otey, C. Wang, S. Parthasarathy, A. Veloso, and W. Meira. Mining frequent itemsets in distributed and dynamic databases. In *Proceedings of the International Conference on Data Mining (ICDM)*, 2003.

[39] T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Distributed deviation detection in sensor networks. *SIGMOD Record*, 2003.

[40] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time series. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2005.

[41] B. Plale. Learning run time knowledge about event rates to improve memory utilization in wide area stream filtering. In *Proceedings of the International Symposium on High Performance Distributed Computing (HPDC)*, 2002.

[42] P. Porras and P. Neumann. EMERALD: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of the National Information Systems Security Conference*, 1997.

[43] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka. An efficient algorithm for the incremental updation of association rules in large databases. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 1997.

[44] A. Veloso, W. Meira Jr., M. B. De Carvalho, B. Possas, S. Parthasarathy, and M. J. Zaki. Mining frequent itemsets in evolving databases. In *Proceedings of the SIAM International Conference on Data Mining*, 2002.

[45] Wei Yu, Dong Xuan, and Wei Zhao. Middleware-based approach for preventing distributed deny of service attacks. In *Proceedings of the IEEE Military Communications Conference*, 2002.

[46] Yongguang Zhang and Wenke Lee. Intrusion detection in wireless ad-hoc networks. In *Mobile Computing and Networking*, 2000.

[47] Yongguang Zhang, Wenke Lee, and Yi-An Huang. Intrusion detection techniques for mobile wireless networks. *Wireless Networking*, 9(5):545–556, 2003.