Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 2482 – 2491

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

A Design Methodology for Distributed Adaptive Stream Mining Systems

Stephen Won[a], Inkeun Cho[a], Kishan Sudusinghe[a], Jie Xu[b], Yu Zhang[b], Mihaela van der Schaar[b], Shuvra S. Bhattacharyya[a,]

[a] *University of Maryland, College Park, USA*
[b] *University of California, Los Angeles, USA*

Abstract

Data-driven, adaptive computations are key to enabling the deployment of accurate and efficient stream mining systems, which invoke suitably configured queries in real-time on streams of input data. Due to the physical separation among data sources and computational resources, it is often necessary to deploy such stream mining systems in a distributed fashion, where local learners have access to disjoint subsets of the data that is to be mined, and forward their intermediate results to an ensemble learner that combines the results from the local learners. In this paper, we develop a design methodology for integrated design, simulation, and implementation of dynamic data-driven adaptive stream mining systems. By systematically integrating considerations associated with local embedded processing, classifier configuration, data-driven adaptation and networked communication, our approach allows for effective assessment, prototyping, and implementation of alternative distributed design methods for data-driven, adaptive stream mining systems. We demonstrate our results on a dynamic data-driven application involving patient health care monitoring.

Keywords: Adaptive stream mining, dataflow graphs, distributed signal processing.

1. Introduction

In this paper, we develop and demonstrate methods for integrated design, prototyping, and implementation of distributed systems for adaptive stream mining. In the novel class of stream mining systems addressed by this work, classifiers operate within embedded systems at sources of vertically distributed data, and interact through communication networks, which allow for their cooperation and adaptation against the incoming data stream to achieve global objectives on prediction/mining accuracy, real-time performance and resource management. Global objectives of resource management and performance for a given threshold of accuracy are achieved by making data-driven decisions at run-time, communicating with the distributed nodes effectively, and adapting across relevant classifier modes. Broad classes of dynamic data-driven application areas that will benefit from such distributed stream mining systems include complex adaptive systems with resilient autonomy; collaborative/cooperative control; autonomous reasoning and learning; sensor-based processing; and ad-hoc, agile networks.

Here, by *vertically distributed* data, we mean that each local learner (e.g., see Figure 1) accesses a subset of the feature space of all instances in the data stream. For example, a bank, a hospital, and an insurance company

Email address: ssb@umd.edu (Shuvra S. Bhattacharyya)

Selection and peer review under responsibility of the organizers of the 2013 International Conference on Computational Science
doi:10.1016/j.procs.2013.05.425

collect different kinds of information about the same customer. A bank has customer information such as the average monthly deposit, and account balance; a hospital has access to medical information; and an insurance company has access to policy information. A statistical analysis evaluating the customer's credit risk for life insurance from all three sources (i.e., these three institutions) is more informative than separate analyses from individual data sources.

However, due to HIPAA (Health Insurance Portability and Accountability) laws, medical data cannot be released or shared for any purpose without appropriate anonymization [1]. Similar U.S. regulations also exist which prohibit the sharing of financial account information from banks and insurance companies. Such privacy constraints thus prevent the raw data to be fully shared among different local learners and no local learner can have full access to the entire dataset (e.g., the complete information about a customer in the previous example). The analysis requires information from all related distributed nodes, and must not only consider the accuracy of the information but also the time to gather all necessary information by communicating with distributed nodes and managing the available resources within the current network. Yet, not all statistical analysis requires to manage resources or adhere to performance constraints. Often there is a trade-off between these two constraints based on the statistical query. Hence, an ensemble learner drives the constraints for data mining in local learners based on the type of query at run-time by making decisions based on the data and network strength observed.

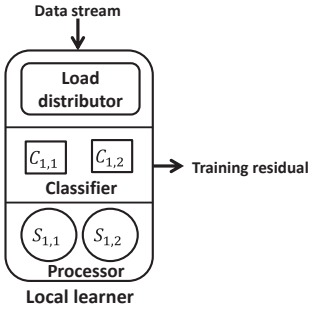

Fig. 1. Local learner structure.

Another important application example for this kind of distributed stream mining system is the identification of climate anomalies for a specific area (e.g., wildfires, droughts, floods, insect/pest damage, wind damage, logging, etc.) by analyzing a number of distributed datasets collected from satellites, ground sensors and other monitoring devices, which contain various observations and measurements about the environment for this area. The massive amount of data collected by these monitoring devices (e.g., one terabyte each night) makes it infeasible to collect all the data into a centralized place to be analyzed and hence, a distributed data mining system based on vertically distributed data is also imperative here. However, in such an application, each local learner has access to only a subset of information. The data collected through the distributed nodes needs to be processed cooperatively, possibly with the aid of a centralized agent, such that accurate decisions can be made dynamically under relevant real-time constraints. The decisions made are then routed back to the distributed nodes to guide subsequent data-driven classification and mining tasks.

Figure 2 illustrates our joint system-application approach. Local learners exchange training information and residuals with the ensemble learner or higher level local learners. Each local learner is comprised of several classifiers using a portion of computational resources as shown in Figure 1. The ensemble learner takes the output of several local learners to make a final classification on a data case while adhering to available resources and performance limitations.

Due to their distributed and data-driven nature and the intensive real-time computations that must be processed on the individual distributed system nodes, the stream mining systems that we target in this work involve cross-cutting challenges in their effective data-driven design, modeling, and implementation. In this paper, we introduce a new modeling and simulation framework, called NL-SIM, to help address these challenges by enabling rapid prototyping of designs prior to implementation. NL-SIM applies advanced methods from signal processing

oriented dataflow graphs as a formal foundation for design; data-driven and distributed system integration; and simulation (e.g., see [2] for reviews of various key topics in the general area of signal processing oriented dataflow graphs).

Furthermore, to concretely demonstrate the novel class of stream mining systems introduced in this paper along with our advances for modeling and simulation of such systems, we present a detailed case study of a practical, distributed stream mining system for the application area of patient health care monitoring. Results from our experimentation on this case study demonstrate the effectiveness of our modeling and simulation techniques in assessing the performance of and communication cost between distributed learners, assessing design trade-offs among alternative distributed stream mining configurations, and validating overall distributed system functionality.

Throughout this work, we apply the DDDAS paradigm by switching between classifier configurations and operational modes dynamically based on data-driven, run-time decisions that are made by the ensemble learner and communicated to the associated local learners. Dynamic changes in classifier configurations are limited to each affected local learner, whereas, different adaptive stream mining configurations can affect subsets of local learners and ensemble learners within the associated design subsystems.

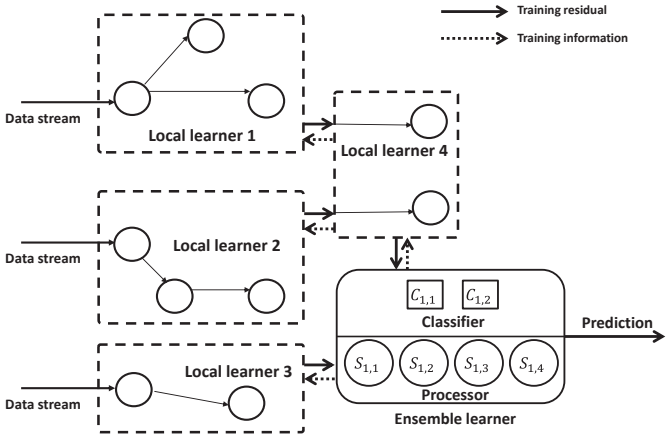

Fig. 2. Distributed stream mining approach.

2. Related Work

The class of distributed stream mining systems addressed in this paper can be viewed as part of the general field of *BigData analytics*, where data is often produced by various distributed data sources producing complementary data sets [3]. Such data sets are characterized by extremely large volumes of data, diverse forms of information (e.g., the different kinds of customer information involved in the example discussed in Section 1), and the distributed nature of the physical sources for the data.

Distributed data mining techniques have been proposed in the literature to process such distributed data sets [3]. Different from traditional centralized data mining systems where a single learner has full access to the global dataset [4], distributed data mining systems typically use ensemble learning techniques consisting of a hierarchy of multiple local learners operating on subsets of the global dataset at the lowest level of the hierarchy, and one or more ensemble learners combining the outputs of all the local learners [5].

In this paper, we explore challenges in the design, modeling, and implementation of such hierarchical, ensemble learning techniques, which require efficient and reliable networked integration of real-time subsystems for local and ensemble learning. Also, different from existing distributed data mining systems where the distributed datasets are usually homogeneous (i.e., producing data in the same form and containing the same set of features), the distributed stream mining techniques proposed in this paper analyze data from heterogeneous sources and having diverse forms of representation. Extending to heterogeneous systems requires more collaboration between

local learners and ensemble learners in the system hierarchy. This added complexity, in conjunction with the need to operate under strict performance and resource constraints, demands significant data-driven decision making at run-time. Furthermore, the dynamic data-driven system methods that we introduce provide novel capabilities toward developing analytics for real-time streaming data.

To design and experiment with our targeted class of distributed stream mining systems, we introduce a novel simulation framework called NL-SIM. The primary distinguishing characteristic of the NL-SIM framework and the underlying modeling and simulation techniques that we present in this paper is their basis in dataflow concepts for the node-level behavior, and their capability of handling network and data traffic conditions to model and simulate interactions between node-level dataflow graphs (local/ensemble classifier subsystems). Such front-end simulation is useful to prototype real-time systems on embedded and distributed platforms, and to assess performance under a wide range of run-time scenarios from an early stage in the design process.

Such an approach is also useful for efficiently validating functional correctness before proceeding to the detailed platform-specific optimization and tuning needed for developing a deployable implementation. This stands in contrast to execution-sequence based co-simulators, which lack dataflow semantics or are restricted to static schedules throughout the network. It also stands in contrast to hybrid system simulators (e.g., see [6]), which focuses on interactions between continuous time and discrete time dynamics. While they may be used effectively to develop specific components, such approaches do not scale well to the complexities in system-level DDDAS design — e.g., to those aspects associated with the heterogeneous nature of data sources in real-time stream mining applications.

Other embedded system design environments have been developed with the objective of integration with advanced network simulation capabilities. For example, NMLab is a co-simulation framework between MATLAB and ns-2 [7]. PiccSIM is a co-simulator with a graphical user interface that also integrates MATLAB/Simulink and ns-2 [8]. SystemC-NS-2 uses the simulation environments of SystemC, a C++ class library used to create system models at different abstraction levels, and ns-2 [9]. Unique features of NL-SIM compared to these previously developed tools are its rigorous application and enforcement of formal dataflow modeling techniques, and NL-SIM's integration with advanced libraries for dataflow modeling, scheduling, and synthesis. These libraries are geared towards implementing diverse kinds of signal processing applications on heterogeneous platforms (e.g., see [10]), which provides important capabilities for handling distributed, heterogeneous system design scenarios in distributed stream mining systems.

3. NL-SIM

In this section, we present our proposed design flow for modeling, simulation, and implementation of dynamic data-driven adaptive stream mining systems. Figure 3 illustrates the steps in this design flow. When designing a distributed, adaptive stream mining system, a designer needs to carefully address issues involving data-driven adaptation, scalability, and resource constraints in an integrated manner. The design flow presented in Figure 3 is developed so that systems can be rigorously assessed and optimized in terms of these key concerns.

To convey how dataflow concepts are tightly integrated into the proposed design methodology, we first discuss the *NS-2 LIDE SIMulation environment* (*NL-SIM*), and then present the design of cooperating local learner and ensemble learner systems using our methodology.

3.1. NL-SIM Simulation Environment

NL-SIM, is a co-simulation tool that systematically integrates the lightweight dataflow environment (LIDE) for embedded signal processing [11] with the Network Simulator (ns-2) tool for efficient and accurate network simulation [12]. NL-SIM provides a flexible design environment that allows designers to simulate real-time embedded processing at the node level, communication protocols at the network level, and the complex interactions between these two levels. This integrated simulation framework provides important capabilities for distributed stream mining applications, where real-time processing for individual learners must be designed, integrated and tested in the context of higher level frameworks that employ networked, cooperative learning, as described in Section 1.

NL-SIM applies techniques for modeling and integration that were introduced in the recently-developed NT-SIM environment [13]. NL-SIM differs from NT-SIM in that the *lightweight dataflow* environment is used instead

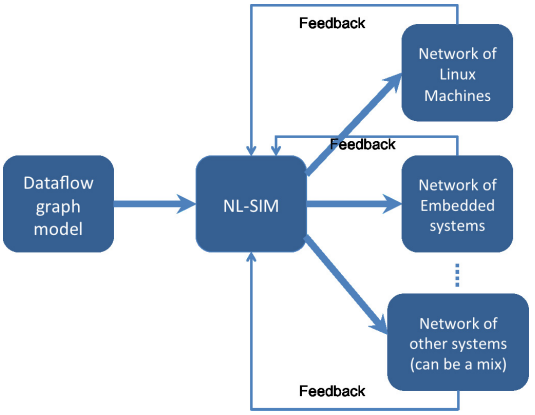

Fig. 3. Proposed design flow for dynamic data-driven adaptive stream mining systems.

of the *targeted dataflow interchange format* [10] for dataflow modeling. Compared to the targeted dataflow interchange format, lightweight dataflow enables efficient experimentation with different kinds of dataflow modeling techniques with minimal constraints on specialized tools or libraries. It is important to consider a broad class of dataflow modeling techniques when designing dynamic data-driven applications so that the applied models can be customized effectively to the unique constraints and dynamic characteristics of the targeted applications.

Distinguishing features of NL-SIM that are useful for distributed stream mining systems, such as those targeted in this paper, include different modes of execution that are optimized for data classification and parameter updates; an operational context to maintain and dynamically adapt parameter values across execution; and efficient retargetability across different kinds of embedded platforms, including graphics processing units, programmable digital signal processors, and field programmable gate arrays. This section provides an overview of key features in NL-SIM and emphasizes its application to distributed stream mining systems. For more comprehensive background on NL-SIM, we refer the reader to [14].

The architecture of NL-SIM is designed to preserve dataflow principles provided by the lightweight dataflow environment throughout all of the lightweight dataflow based subsystems, including interactions that occur at the interfaces of each subsystem. Such dataflow principles can be applied to provide provable bounds on buffer memory requirements, ensure deadlock free operation, and facilitate a variety of high level analysis and optimization techniques for deriving efficient implementations (e.g., see [2]). In NL-SIM, the designer is responsible for distributing the actors across the nodes in the network graph. Here, by an *actor*, we mean a vertex in a dataflow graph, which represents a functional component (hardware or software module) of arbitrary complexity.

An ensemble learner system can be developed using NL-SIM in a hierarchical manner: actor design and dataflow graph subsystems using lightweight dataflow environment and network simulation using ns-2. First-in, first-out (FIFO) communication channels act as bridges between actors in a dataflow graph while interfaces with ns-2 act as bridges between dataflow graphs placed on different network nodes.

Figure 4 illustrates the process of designing stream mining systems using NL-SIM. Given a stream mining application modeled by dataflow, designers create actor and FIFO implementations along with associated test suites using the lightweight dataflow environment programming model [11]. The lightweight dataflow design environment is responsible for scheduling and buffer mapping of the application. Designers then model the network using the ns-2 environment with each node representing a subset of actors in the application. Afterwards, the designer can validate the functionality and assess the performance of the application. For additional details on NL-SIM, we refer the reader to [14].

In the remainder of this section we elaborate on key aspects in the process of modeling and simulating distributed stream mining systems — in particular, the novel class of stream mining systems introduced in Section 1 — using NL-SIM.

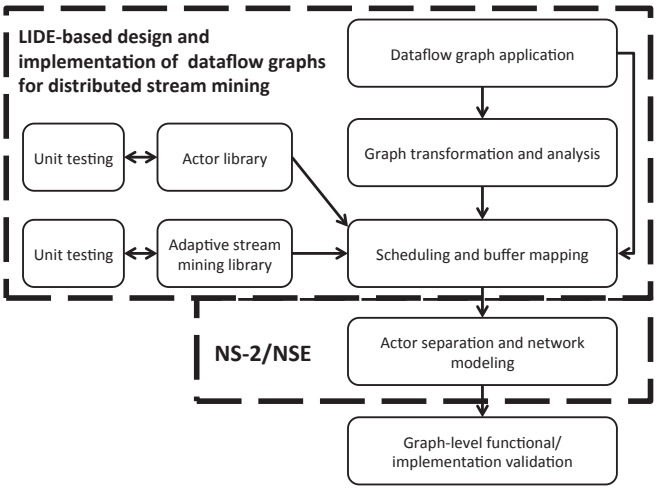

Fig. 4. Design process using LIDE.

3.2. Dataflow Modeling

Each actor is designed in the lightweight dataflow environment. The ensemble system is decomposed into the ensemble and local learner units, which are modeled with distinct actors. The designer is responsible for specifying the inputs, outputs, required parameters, and execution modes of the actors. This information is stored in a data structure called the *operational context*, which is managed dynamically throughout the execution of the associated actor.

In the modeling of classifier subsystems, each execution mode is related to a well-defined configuration in the associated learning algorithm. Such *modes* correspond to the basic unit of actor design in *core functional dataflow* (*CFDF*) [15], which is the specific dataflow model of computation that underlies the lightweight dataflow environment. For local learner units, representative execution modes include reading the data, calculating the predicted class of the data sample, sending relevant data to the ensemble learner, and receiving the updated weights and residual from the ensemble learner.

A dynamic data-driven system can be constructed by switching among such operational modes based on feedback or commands received from the ensemble learner. Local learners communicate run-time parameters that enable the ensemble learner to drive subsequent changes in system- or subsystem-level operating modes. In our case study, which we introduce in Section 4, the ensemble learner contains execution modes that include receiving information from the local learners, calculating new weights and residuals, sending the relevant updated information to the local learners, and reporting training statistics once a significant number of data samples has been accounted for.

Actors are split onto different network nodes depending on their roles in the ensemble system. Each local learner is responsible for predicting the class of the data sample based on its observed features. The ensemble learner is responsible for the update of weights and reporting of the training statistics. In our case-study, each local and ensemble learner is distributed on separate network nodes depending on its role in the ensemble system and resource availability. At present, this separation is done manually through a driver function, written by the designer, which executes the actors in the ensemble network. A useful direction for future work is the automated generation of such driver functions, and their underlying task schedules, based on given performance constraints.

3.3. Prototyping

In this section we elaborate on key aspects in the process of modeling and simulating distributed stream mining systems — in particular, the novel class of stream mining systems introduced in Section 1 — using NL-SIM.

When building the network each agent is attached to a node corresponding to a local or ensemble learner. When there is a node that acts as an ensemble learner to multiple local learners, then the agents are connected, based on ns-2 conventions, using a Tcl script that specifies the network topology on *Network Simulator Emulator* (*NSE*). Once the connections have been specified, network simulator emulator can be run to set up a network that can capture the information exchanged between the ensemble and local learners. Since information is exchanged between the ensemble and local learners, duplex links are set up between nodes to allow communication going in both directions.

After the actors, dataflow graph subsystems, and network have been specified in NL-SIM, the ensemble system can be simulated. The Tcl script for the network is run using network simulator emulator. This allows connections to be made between the lightweight dataflow environment and ns-2 environments. Separate test files are required for each ensemble system node. After executables have been generated for each node in the ensemble system, they can be run concurrently. We have validated the functional accuracy of NL-SIM simulation by comparing with simulation of MATLAB code related to ensemble system training. The training statistics were compared using outputs from 12 classifiers to ensure functional accuracy. Network statistics were measured using the features of ns-2 that are embedded within NL-SIM.

4. Application Case Study

In this section, we present a case study of a practical distributed stream mining system that involves ensemble learning subsystems operating on vertically-distributed data. Through this case study, we concretely demonstrate our proposed design methodology for the class of dynamic, data-driven stream mining systems that is targeted in this work. We also demonstrate the capabilities of NL-SIM for integrated design, modeling, and implementation of such systems. In this case study, we employ a distributed Linux environment as a concrete platform for dynamic, data-driven stream mining implementation. The Linux environment consists of a network of four Intel-based computing platforms that employ virtual machines (Oracle Virtual Box 4.2.6) running Ubuntu Linux.

The application in our case study is a patient care system in a hospital, where the objective is to detect whether or not a given patient needs critical care (i.e., a binary classification label) based on monitoring results collected from a bedside monitor. The monitor provides signals pertaining to multiple features of a patient, such as the heartbeat rate, and blood pressure.

We consider 14 features — i.e., a patient instance in our targeted system can be considered to include 14 features. We model three distributed learning systems— a cooperative system, a partially correlated system, and a DDDAS-based stream mining system — that are constructed from local learning subsystems.

In the cooperative system (e.g., see Figure 4), there are two local learners employed. Each local learner makes predictions based on the classifiers it represents, and sends its predictions to an ensemble learner. The ensemble learner then makes an overall (global) prediction based on the prediction results that it receives.

In the partially correlated system (again, see Figure 4), two groups of local learners are employed, where each local learner observes 7 features of a patient instance. Like the cooperative system, each local learner makes predictions based on the predictions of the classifiers it represents. However, local learners do not exchange messages within groups when updating weights. The ensemble learner combines the results of these groups of local learners to make its global prediction.

We consider the trade-off between execution time (computation and communication time) cost, and resource cost in the cooperative system and partially correlated system. The developed dynamic, data-driven stream mining systems switch among different configurations to tune the trade-off between execution time and resource cost subject to dynamically varying operational constraints. Here, we define the *resource cost* of an operating configuration as the number of processing nodes that is dedicated to the configuration. For our target platform, which is a distributed Linux environment (as described above), the resource cost corresponds to the number of Linux machines that is made available to the configuration.

The ensemble learners are the primary subsystems that provide the dynamic, data-driven features of these systems (i.e., the partially correlated and cooperative systems that we have experimented with). Each system starts in a single mode (initial mode), and continuously monitors its constraints on execution time and resource requirements as it classifies data. As these constraint inputs to the system change, the system autonomously adapts its processing configuration to optimize classification accuracy subject to the new constraints.

When modeling these systems using NL-SIM, we separate the individual local learners and the ensemble learners into three distinct nodes on the emulated networks. This results in three nodes: one node contains the ensemble learner (equivalent to a centralized server) and the other nodes contain the local learners (based on 6 classifiers each).

The training procedures for the ensemble and local learners were developed using the lightweight dataflow environment. The ensemble learner is responsible for updating a weight vector to determine the effect of each classifier from each local learner. After updating the weights, the ensemble learner sends the local learners the updated weights and the training residuals. The local learners are responsible for calculating the gradients to update the weights. The local learners also send their predictions to the ensemble learner actor to update the weights. As this is an online learning algorithm, this is repeated for each of the training cases.

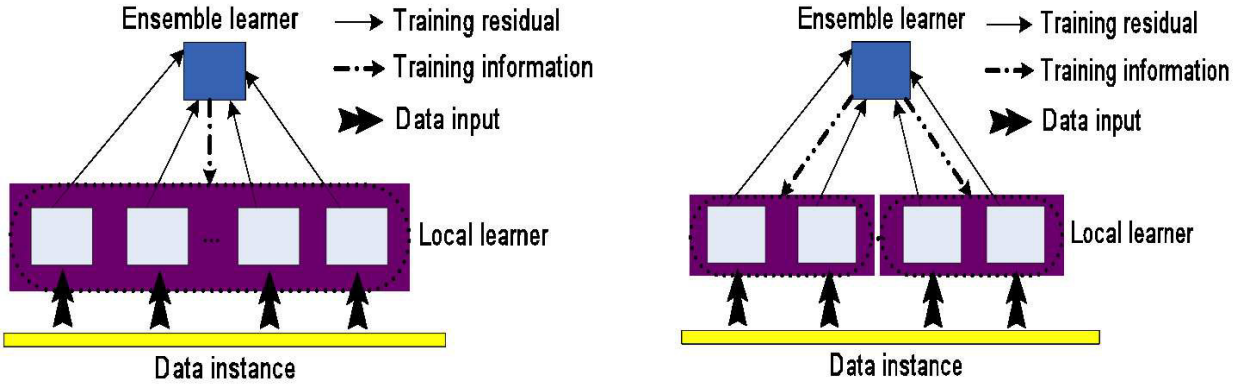

Fig. 5. Networks for cooperative learning (left) and partially correlated learning (right).

During the prototyping phase of the design flow, we test and validate the system functionally in the context of a networked system design. The connections between the ensemble and local learners are made based on the TCP protocol. The network and the connections between nodes are specified using NSE (the emulation capabilities of ns-2). Each node captures packets sent and received from the actors through *packet capture* network objects in NSE. For the cooperative training system, the network is only a duplex link between two nodes. For the partially correlated training system, the network is comprised of two duplex links connecting the local learners with the ensemble learner.

5. Results

Using NL-SIM, we first designed, modeled, and simulated the dynamic, data-driven stream mining systems — i.e., the cooperative and partially correlated systems — described in Section 4. After validating the functionality of these designs using NL-SIM, we implemented them as distributed systems on the four-node Linux platform described in Section 4. In our experiments, the ensemble learner is implemented on a server machine, which is capable of handling multiple training requests from local learners simultaneously.

Figure 6 shows an overview of the network testbed for our implemented stream mining systems. We assume that each local learner has its associated data set residing locally on the processing node in which it executes, so that there is minimal delay in accessing the data set. Thus, execution time results are dependent primarily on the how the local learners are distributed on processing nodes, and on the applied learning mode (partially correlated versus cooperative learning).

We consider three different forms of resource allocation in our experiments — fair, lightly unfair, and unfair distribution. These three forms of distribution are illustrated in Figure 7 .

To assess the the performance of the implemented stream mining systems, we define T_{tr} as the time required to train using a given data set. We experiment with systems that each involve three local learners in conjunction

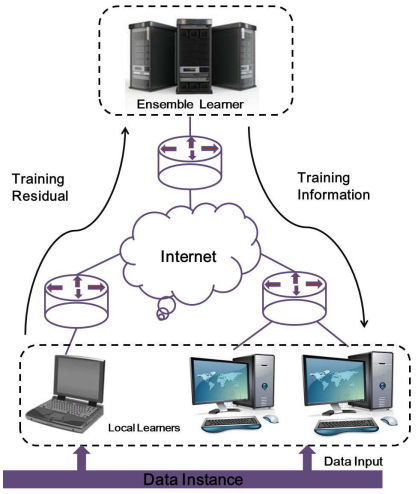

Fig. 6. Network testbed for experimentation with dynamic, data-driven stream mining implementations.

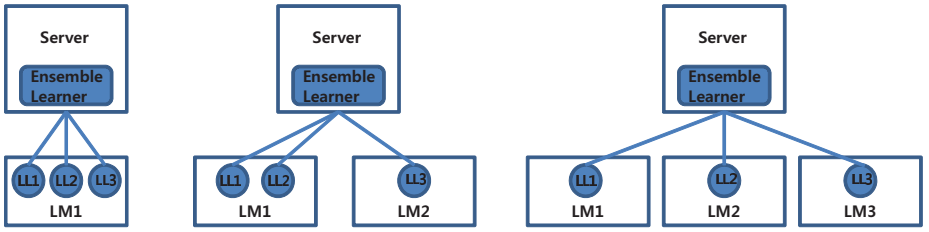

Fig. 7. Forms of resource allocation.

with an ensemble learner.

In the case of fair distribution, the three local learners are deployed on three different local machines. This provides a balanced load distribution, so that T_{tr} is dependent primarily on the learning mode. In the case of lightly unfair distribution, the training time is dependent on both the learning mode and on the given resource constraints, which serialize execution across two of the local learners. In the case of unfair distribution, which may be necessitated due to higher priority tasks in concurrent applications or due to hardware failures, the execution time is determined primarily by the serialization of the local learners.

Figure 8 shows the training times T_{tr} for the cooperative and partially-correlated systems as functions of the number of processing units allocated. These results provide profiles that can be employed at run-time to systematically guide data-driven reconfiguration. Such profile- and data-driven reconfiguration can be employed to effectively adapt system configurations based on different combinations of resource and performance constraints. Additionally, underlying trade-offs between cooperative and partially-correlated learning can be embedded selectively within the applied profiles (stored configurations) so that different forms of learning can be switched dynamically in conjunction with the task distributions, and their associated levels of processing speed and resource utilization.

In Figure 8 , we expect that the execution time differences across the different system configurations is relatively small because of the relatively small scale of the stream mining systems employed and of the associated training data. Experimentation with larger scale systems, involving larger numbers of classifiers, and larger scale distributed platforms is a useful direction for further development of this work. Extending the proposed design framework to take energy consumption into account (e.g., in the case that certain nodes are mobile or battery-powered) is another useful direction for further study.

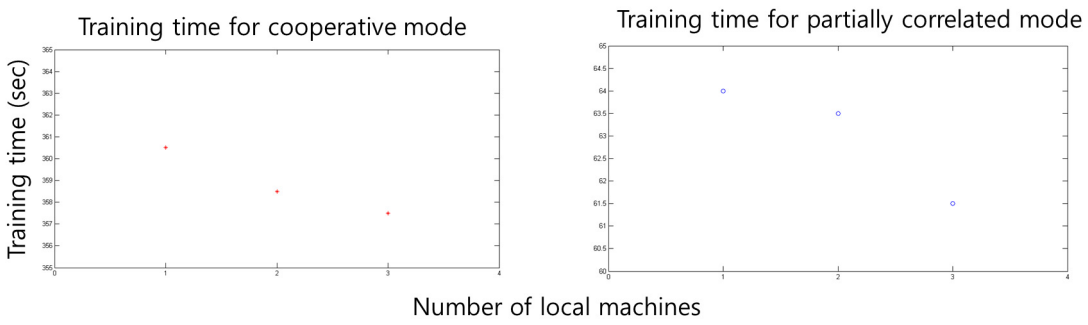

Fig. 8. Training time for cooperative mode (left) and partially-correlated mode (right)

6. Conclusion

In this paper, we have introduced a design framework for distributed, adaptive stream mining. As part of this framework, we have introduced NL-SIM as a co-simulation tool that integrates dataflow modeling principles for embedded signal processing with the advanced network simulation capabilities of NSE. We have presented a design methodology that applies NL-SIM to develop distributed implementations of dynamic, data-driven stream mining systems, and to derive alternative system configurations and associated profiles that can be applied at run-time to adapt high level system attributes, including stream mining modes (e.g, cooperative versus partially-correlated learning), and task distributions across the deployed processing nodes. We have demonstrated our proposed methods through a distributed stream mining case study involving a practical healthcare application.

Acknowledgements

This research is supported by the US Air Force Office of Scientific Research under the DDDAS Program.

References

[1] S. Calloway, L. Venegas, The new HIPAA law on privacy and confidentiality, Nursing Administration Quarterly 26 (4) (2002) 40–54.
[2] S. S. Bhattacharyya, E. Deprettere, R. Leupers, J. Takala (Eds.), Handbook of Signal Processing Systems, Springer, 2010.
[3] B. Park, H. Kargupta, Distributed data mining: Algorithms, systems, and applications, in: N. Ye (Ed.), Data Mining Handbook, Lawrence Erlbaum Associates, 2004.
[4] M. Kantardzic, Data Mining: Concepts, Models, Methods, and Algorithms, 2nd Edition, Wiley–IEEE Press, 2011.
[5] P. K. Chan, S. J. Stolfo, Experiments on multistrategy learning by meta-learning, in: Proceedings of the International Conference on Information and Knowledge Management, 1993, pp. 314–323.
[6] A. Chutinan, B. H. Krogh, Computational techniques for hybrid system verification, IEEE Transactions on Automatic Control 48 (1) (2003) 64–75.
[7] O. Heimlich, R. Sailer, L. Budzisz, NMLab: A co-simulation framework for Matlab and ns-2, in: Proceedings of the International Conference on Advances in System Simulation, 2010, pp. 152–157.
[8] T. Kohtamaki, M. Pohjola, J. Brand, L. M. Eriksson, PiccSIM toolchain — design, simulation and automatic implementation of wireless networked control systems, in: Proceedings of the International Conference on Networking, Sensing and Control, 2009, pp. 49–54.
[9] F. Fummi, G. Perbellini, P. Gallo, M. Poncino, S. Martini, F. Ricciato, A timing-accurate modeling and simulation environment for networked embedded systems, in: Proceedings of the Design Automation Conference, 2003, pp. 42–47.
[10] C. Shen, S. Wu, N. Sane, H. Wu, W. Plishker, S. S. Bhattacharyya, Design and synthesis for multimedia systems using the targeted dataflow interchange format, IEEE Transactions on Multimedia 14 (3) (2012) 630–640.
[11] C. Shen, W. Plishker, H. Wu, S. S. Bhattacharyya, A lightweight dataflow approach for design and implementation of SDR systems, in: Proceedings of the Wireless Innovation Conference and Product Exposition, Washington DC, USA, 2010, pp. 640–645.
[12] K. Fall, K. Varadhan, The ns Manual (formerly ns Notes and Documentation) (November 2011).
[13] S. Won, C. Shen, S. S. Bhattacharyya, NT-SIM: A co-simulator for networked signal processing applications, in: Proceedings of the European Signal Processing Conference, Bucharest, Romania, 2012, pp. 1094–1098.
[14] S. Won, A networked dataflow simulation environment for signal processing and data mining applications, Master's thesis, Department of Electrical and Computer Engineering, University of Maryland, College Park (2012).
[15] W. Plishker, N. Sane, M. Kiemb, K. Anand, S. S. Bhattacharyya, Functional DIF for rapid prototyping, in: Proceedings of the International Symposium on Rapid System Prototyping, Monterey, California, 2008, pp. 17–23.