Distributed Adaptive Model Rules for Mining Big Data Streams

Anh Thu Vu
Royal Institute of Technology
atvu@kth.se

Gianmarco De Francisci Morales
Yahoo Labs, Barcelona
gdfm@yahoo-inc.com

João Gama
University of Porto
jgama@fep.up.pt

Albert Bifet
HUAWEI Noah's Ark Lab
bifet.albert@huawei.com

***Abstract*—Decision rules are among the most expressive data mining models. We propose the first distributed streaming algorithm to learn decision rules for regression tasks. The algorithm is available in SAMOA (SCALABLE ADVANCED MASSIVE ONLINE ANALYSIS), an open-source platform for mining big data streams. It uses a hybrid of vertical and horizontal parallelism to distribute Adaptive Model Rules (AMRules) on a cluster. The decision rules built by AMRules are comprehensible models, where the antecedent of a rule is a conjunction of conditions on the attribute values, and the consequent is a linear combination of the attributes. Our evaluation shows that this implementation is scalable in relation to CPU and memory consumption. On a small commodity Samza cluster of 9 nodes, it can handle a rate of more than $30\,000$ instances per second, and achieve a speedup of up to 4.7x over the sequential version.**

I. INTRODUCTION

In this new media age, data is being generated from many of our daily activities as we interact with software. The data in a post to a social network, or a purchase with a credit card, or an access to the GPS, can all potentially produce useful information for interested parties. The recent advancements of mobile devices has further increased the rate and amount of data being generated, as people can generate data anywhere, anytime, using different devices and technologies. There is a lot of interest in extracting knowledge from this massive amount of data and using it, e.g., to choose a suitable business strategy, or to improve healthcare services. Moreover, many applications are required to be able to process incoming data and react to it in online using comprehensible prediction mechanisms. For example, when a bank monitors the transactions of its clients to detect frauds, it needs to identify and verify the fraud as soon as the transaction is performed and immediately block it or adjust the prediction mechanism.

The task of analyzing, extracting patterns, and predicting data from various sources, with massive volume and high incoming rate, is referred to as *big data stream mining*. SAMOA (SCALABLE ADVANCED MASSIVE ONLINE ANALYSIS) [1] is a platform designed to solve the challenges of *big data stream mining* tasks. It is a framework that eases the development of new distributed machine learning algorithms and the deployment of these implementations on top of state-of-the-art distributed stream processing engines (DSPEs). It is also a library of distributed data mining and machine learning algorithms that allows users to use or customize existing ones. By utilizing state-of-the-art DSPEs SAMOA enables distributed algorithms to scale with the volume and incoming rate of data. Easy development of new algorithms or customization of existing ones allow SAMOA's users to solve different stream mining tasks efficiently.

Work done while interning at Yahoo Labs.

In this paper, we present a distributed implementation of Adaptive Model Rules (AMRules) [2], an online decision rule algorithm for regression tasks. The goal of this work is to improve its scalability and enable it to handle big data streams. A comparison of AMRules with other algorithms for regression is out of the scope of this paper, and can be found in the works of Almeida et al. [2], Duarte and Gama [3].

The reasons for which we choose to parallelize AMRules are twofold. First, besides *classifcation* and *clustering*, *regression* is a third predominant task in machine learning; up to now, SAMOA lacked a distributed algorithm for the latter task. Second, the modularity of decision model rules make them suitable for a distributed implementation.

Regression analysis has been widely studied in statistics, pattern recognition, machine learning and data mining. It estimates a functional relationship between a continuous dependent variable and a set of independent variables. The most expressive data mining models for regression are model trees and regression rules. Model trees and model rules are among the ones with better performance [4]. One important aspect of rules, and the main advantage over trees, is modularity: each rule can be interpreted per se. Both trees and rules do automatic feature selection, are robust to outliers and irrelevant features, exhibit high degree of interpretability and structural invariance to monotonic transformation of the independent variables.

In summary, this paper makes the following contributions:

- We design the first distributed streaming algorithm for regression based on a state-of-the-art algorithm;
- We explore the different trade-offs involved in parallelizing AMRules and experiment with large real datasets;
- We make our implementation available as open-source software as part of SAMOA.

The paper is organized as follows. We discuss related work in Section II. In Section III, we present some preliminaries about SAMOA. Section IV describes the new distributed rule algorithms for evolving data streams. We report on our empirical evaluation of the new methods proposed in Section V. Finally, Section VI concludes the paper.

II. Related Work

Currently, there are only a limited number of regression algorithms for streaming data. One of them is Fast Incremental Model Trees with Drift Detection (FIMT-DD) [5]. FIMT-DD applies a modified version of the Hoeffding tree [6] to incrementally build a tree model from incoming instances. It uses the Page-Hinckley test to detect changes. If the test detects a change at a node, an alternative subtree is built simultaneously with the old one from that node. The performance of both subtrees is continuously monitored and compared. If its performance of the old subtree falls below the new one, the subtree is replaced.

A second notable algorithm is IBLStreams (Instance Based Learner on Streams) [7]. It is an instance-based learning algorithm (similar to k-NN) for both classification and regression. Instead of managing a decision model as in model-based algorithms, an instance-based algorithm stores and continuously updates a subset of examples, called the case base. These examples will only be processed when a prediction is actually required. Upon the arrival of a new example x_i, IBLStreams will add it to the case base and find examples to be removed among x_i's neighbors. The neighbors of x_i are determined by a distance function based on a *value difference metric* [8].

The Adaptive Model Rules (AMRules) [2], is the first rule-based learning algorithm for regression problems on streams. In AMRules the antecedent of a rule is a conjunction of conditions on the attribute values, and the consequent is a linear combination of the attributes. In order to maintain a regression model compatible with the most recent state of the process generating data, each rule uses a Page-Hinkley test to detect changes in this process and react to changes by pruning the rule set. Online learning might be strongly affected by outliers. AMRules is also equipped with outlier detection mechanisms to avoid model adaption using anomalous examples. Ensembles of AMRules have been studied by Duarte and Gama [3], Almeida et al. [9].

III. Preliminaries

SAMOA is an open-source distributed stream mining platform developed at Yahoo Labs [10]. It allows easy implementation and deployment of distributed streaming machine learning algorithms on supported DSPEs. Besides, it provides the ability to integrate new DSPEs into the framework and leverage their scalability for performing big data mining.

SAMOA is both a framework and a library. As a framework, it allows the algorithm developer to abstract from the underlying execution engine, and therefore reuse their code on different engines. It features a pluggable architecture that allows it to run on several distributed stream processing engines such as Samza, Storm and S4. This capability is achieved by designing a minimal API that captures the essence of modern DSPEs. This API also allows to easily write new bindings to port SAMOA to new execution engines. SAMOA takes care of hiding the differences of the underlying DSPEs in terms of API and deployment.

As a library, SAMOA contains implementations of state-of-the-art algorithms for distributed machine learning on streams. For classification, SAMOA provides a Vertical Hoeffding Tree (VHT), a distributed streaming version of a decision tree. For clustering, it includes an algorithm based on CluStream. The library also includes meta-algorithms such as bagging and boosting. The platform is useful in both research and real world deployments.

An algorithm in SAMOA is represented by a directed graph of operators that communicate via messages along streams which connect pairs of nodes. Borrowing the terminology from Storm, this graph is called a *Topology*. Each node in a Topology is a *Processor* that sends messages through a *Stream*. A Processor is a container for the code that implements the algorithm. At runtime, several parallel instances of a Processor are instantiated by the framework. A Stream can have a single source but several destinations (akin to a pub-sub system). A Topology is built by using a *TopologyBuilder*, which connects the various pieces of user code to the platform code and performs the necessary bookkeeping in the background.

A processor receives *Content Events* via a Stream. Algorithm developers instantiate a Stream by associating it with exactly one source Processor. When the destination Processor want to connect to a Stream, it needs to specify the *grouping* mechanism which determines how the Stream partitions and routes the transported Content Events. Currently there are three grouping mechanisms in SAMOA:

- *Shuffle grouping*, which routes the Content Events in a round-robin way among the corresponding Processor instances. This means each Processor instance receives the same number of Content Events from the stream.
- *Key grouping*, which routes the Content Events based on their *key*, i.e. the Content Events with the same key are always routed by the Stream to the same Processor instance.
- *All grouping*, which replicates the Content Events and broadcasts them to all downstream Processor instances.

AMRules. Decision rule learning is a category of machine learning algorithms whose goal is to extract a set of decision rules from the training data. These rules are later used to predict the unknown *label* values for test data. A rule is a logic expression of the form:

$$\textbf{IF } antecedent \textbf{ THEN } consequent$$

or, equivalently, $head \leftarrow body$, where *head* and *body* correspond to the *consequent* and *antecedent*, respectively.

The body of a rule is a conjunction of multiple clauses called *features*, each of which is a condition on an attribute of the instances. Such conditions consist of the identity of an attribute, a threshold value and an operator. For instance, the feature "$x < 5$" is a condition on attribute *x*, with threshold value *5* and operator *less-than* ($<$). An instance is said to be *covered* by a rule if its attribute values satisfy all the features in the rule body. The head of the rule is a function to be applied on the covered instances to determine the their label values. This function can be a constant or a function of the attributes of the instances, e.g.,

$$ax + b \leftarrow x < 5$$

AMRules is an algorithm for learning regression rules on streaming data. It incrementally constructs the rule model from the incoming data stream. The rule model consists of a set of *normal* rules (which is empty at the beginning), and a *default* rule. Each normal rule is composed of 3 parts: a *body* which is a list of features, a *head* with information to compute the prediction for those instance covered by the rule, and *statistics* of past instances to decide when and how to add a new feature to its body. The default rule is a rule with an empty *body*.

For each incoming instance, AMRules searches the current rule set for those rules that cover the instance. If an instance is not covered by any rule in the set, it is considered as being covered by the default rule. The heads of the rules are first used to compute the prediction for the instance they cover. Later, their statistics are updated with the attribute values and label value of the instance. There are two possible modes of operation: ordered and unordered. In ordered-rules mode, the rules are checked according to the order of their creation, and only the first rule is used for prediction and then updated. In unordered-rules mode, all covering rules are used and updated. In this work, we focus on the former which is more often used albeit more challenging to parallelize.

Each rule tries to expand its body after it receives N_m updates. In order to decide on the feature to expand, the rule incrementally computes a standard deviation reduction (SDR) measure [5] for each potential feature. Then, it computes the $ratio$ of the second-largest SDR value over the largest SDR value. This $ratio$ is used with a high confidence interval ϵ computed using the Hoeffding bound [11] to decide to expand the rule or not: if $ratio + \epsilon < 1$, the rule is expanded with the feature corresponding to the largest SDR value. Besides, to avoid missing a good feature when there are two (or more) equally good ones, rules are also expanded if the Hoeffding bound ϵ falls below a threshold. If the default rule is expanded, it becomes a *normal* rule and is added to the rule set. A new default rule is initialized to replace the previous one.

Each rule records its prediction error and applies a modified version of the Page-Hinkley test [12] for streaming data to detect changes. If the test indicates that the cumulative error has exceeded a threshold, the rule is evicted from the rule set.

The algorithm also employ outlier detection to check if an instance, although being covered by a rule, is an anomaly. If an instance is deemed as an anomaly, it is treated as if the rule does not cover it and is checked against other rules.

A simplified version of the pseudo code for training AMRules is shown in Algorithm 1.

Algorithm 1: Training algorithm of AMRules.

```
1  ________________________________________________
   Input: S: Stream of examples
2  ordered-set: boolean flag
3  $N_m$: minimum number of examples before attempting
   to expand
   Result: RS: set of decision rules
4  begin
5      Let $RS \leftarrow \{\}$
6      Let $defaultRule\ \mathcal{L} \leftarrow 0$
7      foreach example $x \in S$ do
8          foreach rule $r \in RS$ do
9              if $r$ covers example $x$ and $x$ is not an
               anomaly then
10                 Compute prediction error $e$
11                 Call PageHinkleyTest($e$)
12                 if change is detected then
13                     Remove the rule $r$
14                 else
15                     Update statistics of $r$
16                     if Number of examples in
                       $\mathcal{L}_r$ $mod$ $N_m = 0$ then
17                         $r \leftarrow ExpandRule(r)$
18                 if ordered-set then
19                     break
20          if no rule in $RS$ covers $x$ then
21              Update statistics of $defaultRule$
22              if Number of examples in $\mathcal{L}$ $mod$ $N_m = 0$
                then
23                  $RS \leftarrow$
                    $RS \cup ExpandRule(defaultRule)$
24 ________________________________________________
```

IV. Distributed AMRules

This section describes two possible strategies to parallelize AMRules that we propose.

A. Vertical Parallelism

In AMRules, each rule can evolve independently, as its expansion is based solely on the statistics of instances it covers. Also, searching for the best feature among all possible ones in an attempt to expand a rule is computationally expensive.

Given these observations, we decide to parallelize AMRules by delegating the training process of rules to multiple *learner processors*, each of which handles only a subset of the rules. Besides the learners, a *model aggregator processor* is also required to filter and redirect the incoming instances to the correct learners. The aggregator manages a set of simplified rules that have only head and body, i.e., do not keep statistics. The bodies are used to identify the rules that cover an instance, while the heads are used to compute the prediction. Each instance is forwarded to the designated learners by using the ID of the covering rule. At the learners, the corresponding rules' statistics are updated with the forwarded instance. This parallelization scheme guarantees that the rules created are the same as in the sequential algorithm. Figure 1 depicts the design of this vertically parallelized version of AMRules, or Vertical AMRules (VAMR for brevity).

The model aggregator also manages the statistics of the default rule, and updates it with instances not being covered by any other rule in the set. When the default rule is expanded and adds a new rule to the set, the model aggregator sends a message with the newly added rule to one of the learners, which will be responsible for its management. The assignment of a rule to a learner is done based on the rule's ID. All subsequent instances that are covered by this rule are forwarded to the same learner.

At the same time, learners update the statistics of each corresponding rule with each processed instance. When enough statistics have been accumulated and a rule is expanded, the new feature is sent to the model aggregator to update the

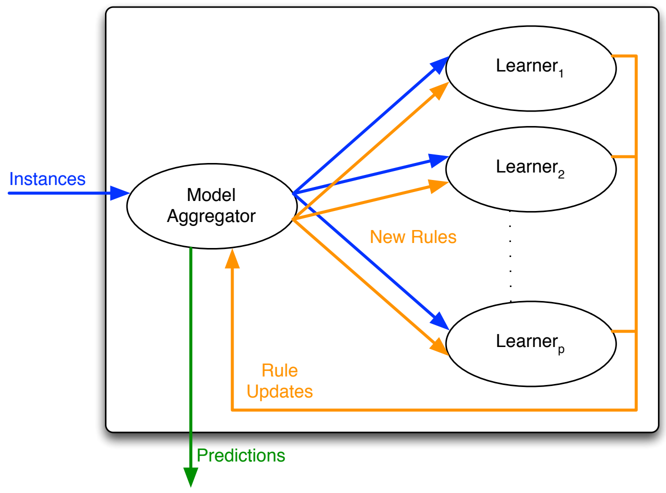


Fig. 1: Vertical AMRules (VAMR).

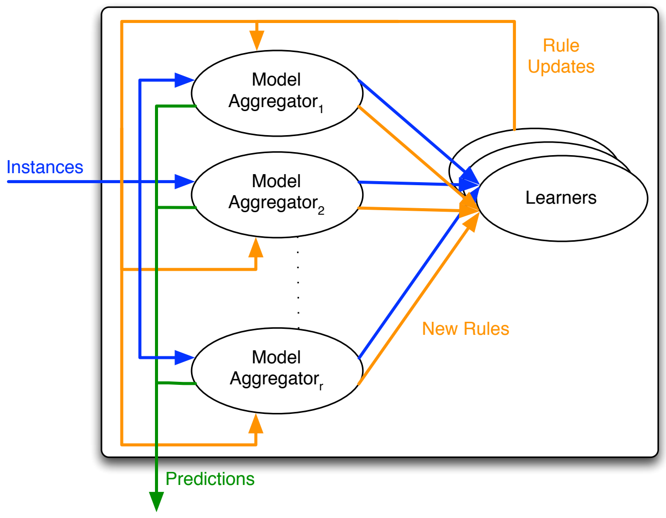


Fig. 2: AMRules with multiple horizontally parallelized Model Aggregators.

body of the rule. Learners can also detect changes and remove existing rules. In such an event, learners will inform the model aggregator with a message containing the removed rule ID.

As each rule is replicated in the model aggregator and in one of the learners, their bodies in model aggregator might not be up-to-date. The delay between rule expansion in the learner and model update in the aggregator depends mainly on the queue length at the model aggregator. The queue length, in turn, is proportional to the volume and speed of the incoming data stream. Therefore, instances that are in the queue before the model update event might be forwarded to a recently expanded rule which no longer covers the instance.

Coverage test is performed again at the learner, thus the instance is dropped if it was incorrectly forwarded. Given this additional test, and given that rule expansion can only increase the selectivity of a rule, when using unordered rules the accuracy of the algorithm is unaltered. However, in ordered-rules mode, these temporary inconsistencies might affect the statistics of other rules because the instance should have been forwarded to a different rule. We investigate the effect of such inconsistencies on the accuracy of the algorithm in Section V.

B. *Horizontal Parallelism*

As shown in Section V, a bottleneck in VAMR is the centralized model aggregator. Given that there is no straightforward way to vertically parallelize the execution of the model aggregator while maintaining the order of the rules, we explore an alternative based on horizontal parallelism. Specifically, we introduce multiple replicas of the model aggregator, so that each replica maintains the same copy of the rule set but processes only a portion of the incoming instances.

Horizontally parallelized model aggregator. The design of this scheme is illustrated in Figure 2. The basic idea is to extend VAMR and accommodate multiple model aggregators into the design. Each model aggregator still has a rule set and a default rule. The behavior of this scheme is similar to VAMR, except that each model aggregator now processes only a portion of the input data, i.e., the amount of instances each of them receives is inversely proportional to the number of model aggregators. This will affect the prediction statistics and, most importantly, the training statistics of the default rules.

Since each model aggregator processes only a portion of the input stream, each default rule is trained independently with different portions of the stream. Thus, these default rules will evolve independently and potentially create overlapping or conflicting rules. This fact also introduces the need for a scheme to synchronize and order the rules created by different model aggregators. Additionally, at the beginning, the scheme is less reactive compared to VAMR as it requires more instances for the default rules to start expanding. Besides, as the prediction function of each rule is adaptively constructed based on attribute values and label values of past instances, having only a portion of the data stream will lead to having less information and potentially lower accuracy. We show how

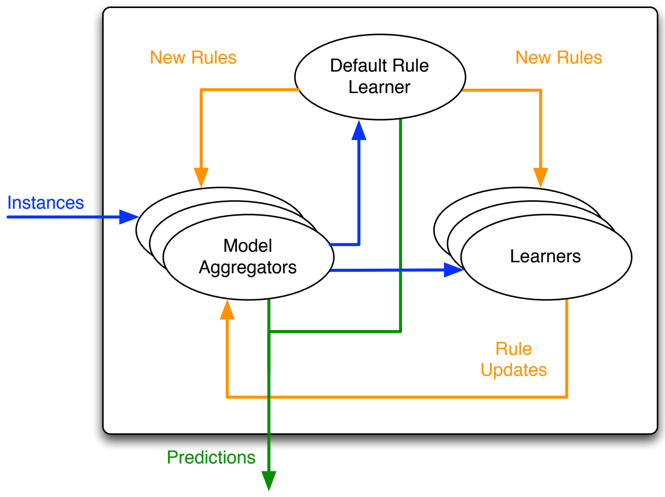


Fig. 3: Hybrid AMRules (HAMR) with multiple Model Aggregators and separate Default Rule Learner.

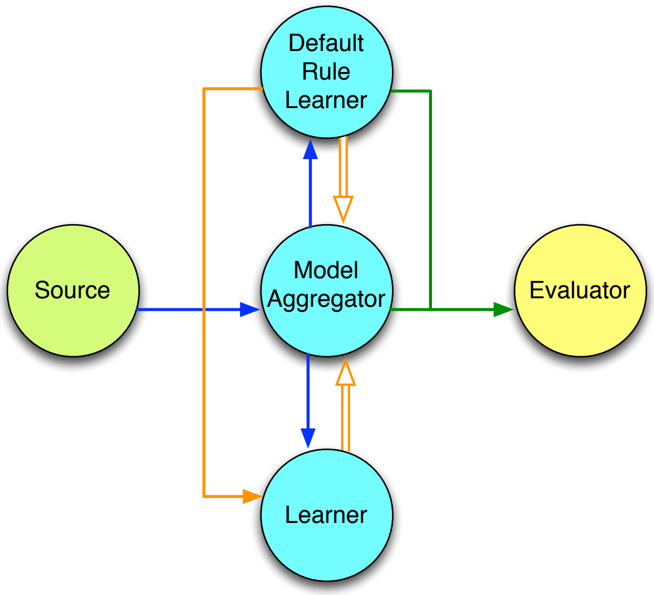


Fig. 4: Prequential evaluation task for HAMR. Single lines represent single messages (*key grouping*, *shuffle grouping*) while double lines represent broadcast messages (*all grouping*).

to address these issues next.

Centralized rule creation. In order to address the issues with distributed creation of rules, we move the default rule in model aggregators to a specialized default rule learner processor. With the introduction of this new component, some modifications are required in the model aggregators, but the behavior of the learners is still the same as in VAMR. However, as a result, all the model aggregators will be in synch.

As the default rule is now moved to the designated learner, those instances that are not covered by any rules are forwarded from the model aggregators to this learner. This specialized learner updates its statistics with the received instances and, when the default rule expands, it broadcasts the newly created rule to the model aggregators. The new rule is also sent to the assigned learner, as determined by the rule's ID.

The components of this scheme are shown in Figure 3. Henceforth, we will refer to this scheme as Hybrid AMRules (HAMR), as it is a combination of vertical and horizontal parallelism strategies.

V. Evaluation

We evaluate the performance of the 2 distributed implementations of AMRules, i.e., VAMR and HAMR, in comparison to the centralized implementation in MOA[1] (MAMR).

Evaluation methodology. We plug VAMR and HAMR into a *prequential evaluation task* [13], where each instance is first used to test and then to train the model. This evaluation task includes a *source* processor which provides the input stream and a *evaluator* processor which records the rate and accuracy of prediction results. The final task for HAMR is depicted in Figure 4. The parallelism level of the model is controlled by setting the number of learners p and the number of model aggregators r. The task for VAMR is similar but the default rule learner is excluded and model aggregator's parallelism level is always 1. Each task is repeated for five runs.

Datasets. We perform the same set of experiments with 3 different datasets, i.e., *electricity*, *airlines* and *waveform*. *Electricity* is a dataset from the UCI Machine Learning Repository [14] which records the electricity power consumption (in watt-hour) of a household from December 2006 to November 2010. The dataset contains more than 2 millions 12-attribute records. *Airlines* is another real dataset recording the arrival delay (in seconds) of commercial flights within the USA in year 2008[2]. It contains more than 5.8 millions records, each has 10 numerical attributes. On the other hand, *waveform* is generated using an artificial random generator. To generate an instance, it picks a random waveform among the 3 available ones. 21 attribute values for this instance are generated according to the chosen waveform. Another 19 noise attributes are generated and included in the new instance, making a total of 40 attributes for each instance. The label value is the index of the waveform (0, 1, or 2). Although this dataset does not fit perfectly to the definition of a regression task, it allows us to test our implementations with a high number of numerical attributes.

Setup. The evaluation is performed on an OpenStack[3] cluster of 9 nodes, each has 2 Virtual CPUs @ 2.3GHz and 4GB of RAM. All the nodes run Red Hat Enterprise Linux 6. The distributed implementations are evaluated on a Samza[4] cluster with Hadoop YARN 2.2[5] and Kafka 0.8.1[6]. The Kafka brokers coordinate with each other via a single-node ZooKeeper 3.4.3[7] instance. The replication factor of Kafka's streams in these experiments is 1. The performance of MAMR is measured on one of the nodes in the cluster.

A. *Throughput*

The throughput of several variants of AMRules is shown in Figure 5, 6 and 7, one for each dataset. HAMR-1 and HAMR-2 stand for *HAMR with 1 learner* and *HAMR with 2 learners*. The parallelism levels of VAMR represents the number of its learners, while in the case of HAMR it represents the number of model aggregators.

With *electricity* and *waveform*, the communication overhead in VAMR (to send instances from model aggregator to learners and evaluator) exceeds the throughput gained from delegating the training process to the learners and results in a lower overall throughput compared to MAMR's. However, with *airlines*, the performance of VAMR is better than MAMR. To verify that the training process for *airlines* is more computationally intensive than the other two datasets and, thus, VAMR is more effective for this dataset, we compare the statistics of rules and predicates creation for the 3 datasets. The number of

[1]http://moa.cms.waikato.ac.nz

[2]http://kt.ijs.si/elena_ikonomovska/data.html
[3]http://www.openstack.org
[4]http://samza.incubator.apache.org
[5]http://hadoop.apache.org
[6]http://kafka.apache.org
[7]http://zookeeper.apache.org

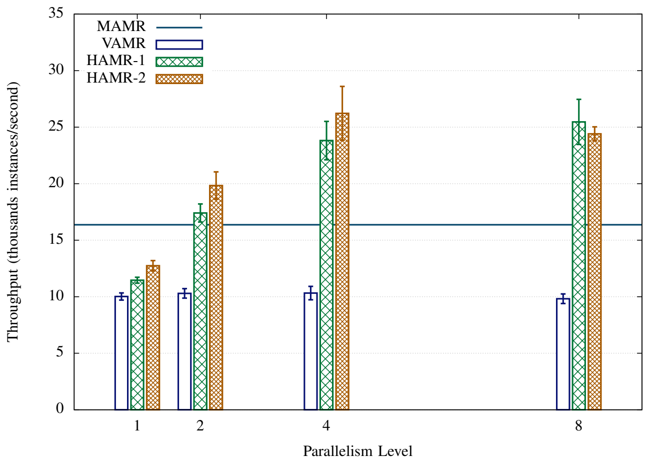


Fig. 5: Throughput of distributed AMRules with *electricity*.

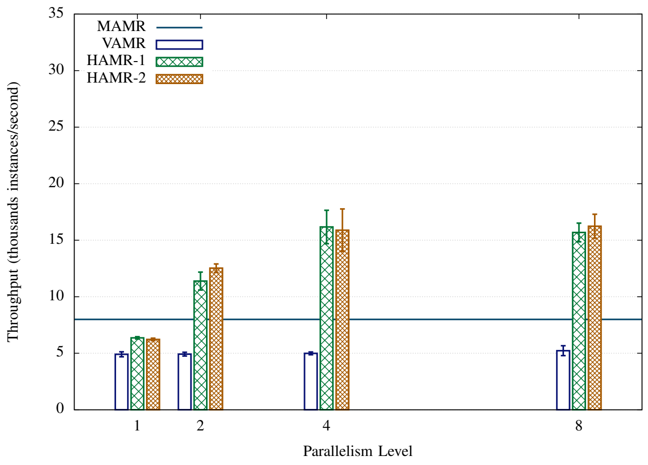


Fig. 7: Throughput of distributed AMRules with *waveform*.

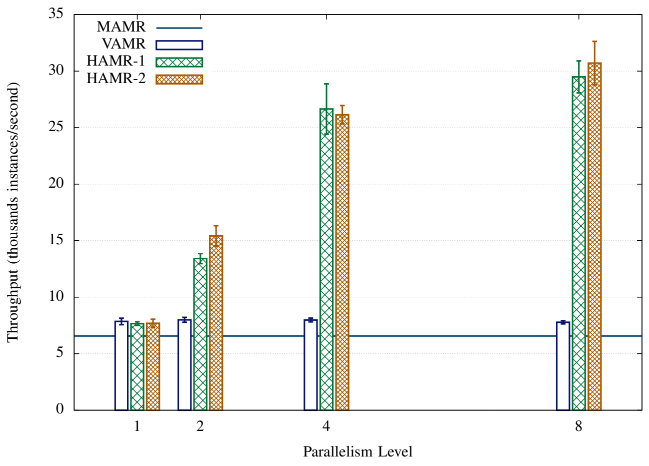


Fig. 6: Throughput of distributed AMRules with *airlines*.

rules created, rules removed, and features created (when a rule is expanded) by MAMR with the 3 datasets are presented in Table I. By subtracting the total number of rules removed from the total number of rules created, we can have an estimation of the average number of rules in the model for each dataset. A higher number of rules in the model and a higher number of features created suggest that the model is more complex and it takes more processing power to search for the best new feature when a rule attempts to expand.

TABLE I: Statistics for features and rules (wih MAMR) for different datasets.

	Electricity	Airlines	Waveform
Instances	2 049 280	5 810 462	1 000 000
# Attributes	12	10	40
Result message size (B)	891	764	1446
# Rules created	1203	2501	270
# Rules removed	1103	1040	51
Avg. # Rules	100	1461	219
# Features created	1069	10 606	1245

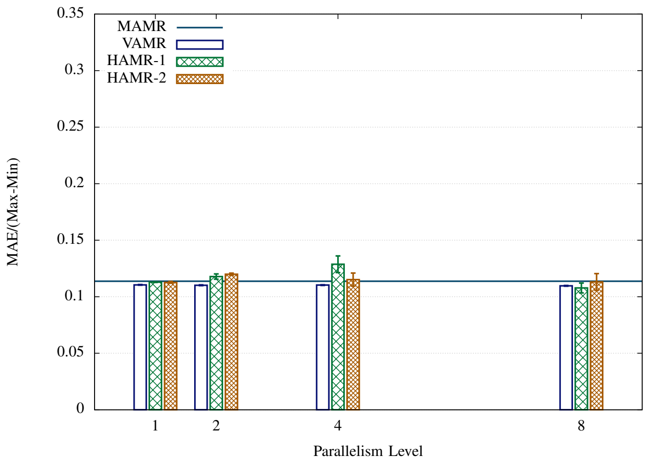
Although VAMR can perform better than MAMR for computationally intensive datasets, its throughput does not change with different parallelism level. This is due to the bottleneck at the model aggregator. The processing of each instance at the model aggregator consists of 3 steps: finding the covering rules, forwarding the instance to the corresponding learners and applying covering rules to predict the label value of the instance. Since the complexity of these 3 steps for an instance is constant, the final throughput is unaffected by the parallelism level.

The throughput of HAMR-1 and HAMR-2 exhibit a better scalability compared to VAMR. Up to parallelism level of 4, the throughput increases almost linearly with the number of model aggregators. However, there is no or little improvement when this number is increased from 4 to 8. As we measure this throughput at a single evaluator, we suspect that the bottleneck is in the maximum rate the evaluator can read from the output streams of the model aggregators and default rule learner. To investigate this issue, we plot the maximum throughput of HAMR against the size of messages from model aggregators and default rule learner to evaluator in Figure 8. The values of throughput of a single-partition Samza stream with messages of size 500B, 1000B and 2000B are used to compute the linear regression line (reference line) in the figure. The message size for different datasets is shown in Table I.

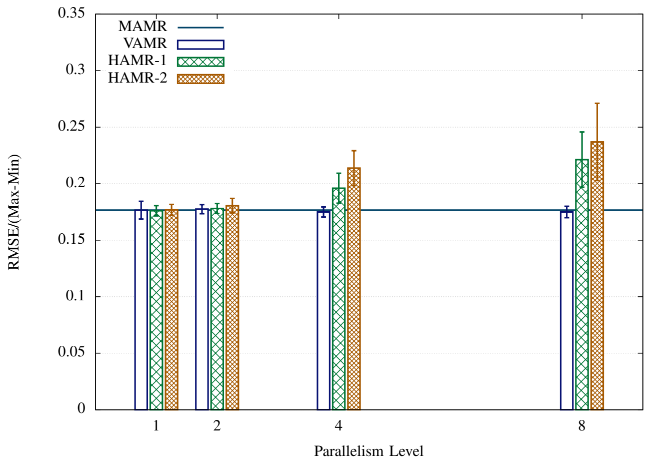
As reading is expected to be faster than writing in Samza and Kafka, the maximum rate the evaluator in HAMR can read from multiple stream partitions is expected to be higher than the throughput represented by the reference line. This fact is reflected in Figure 8 as the maximum throughput of HAMR for the 3 datasets constantly exceeds the reference line. However, the difference between them is relatively small. This result is a strong indicator that the bottleneck is the maximum reading rate of the evaluator. If there is no need to aggregate the result from different streams, this bottleneck can be eliminated.

B. Accuracy

We evaluate accuracy of the different implementations of AMRules in terms of *Mean Absolute Error* (MAE) and *Root Mean Square Error* (RMSE). Figure 9, 10 and 11 show the



(a) MAE



(b) RMSE

Fig. 9: MAE and RMSE of distributed AMRules with *electricity* dataset.

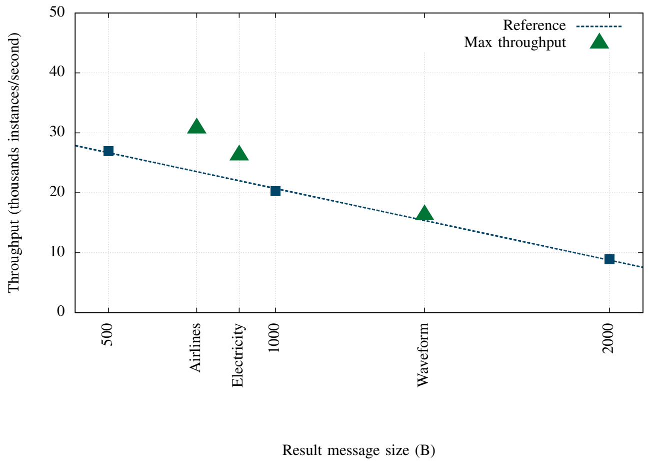


Fig. 8: Maximum throughput of HAMR vs message size.

MAE and RMSE for the three datasets, normalized by the range of label values in each dataset.

Most of the figures show that error values of distributed implementations presents very small fluctuations around the corresponding MAMR error line. However, there is a significant increment in the value of RMSE in HAMR with *electricity* dataset when the number of model aggregators is increased to 4 or 8. Moreover, larger variances, i.e., *standard error* is greater than 5% of the average MAE or RMSE, are also observed in the case of higher parallelism levels ($p \geq 4$) of HAMR. The probable cause is that when a rule in the model aggregators is out-of-sync with the corresponding one in the learners, i.e., model aggregators are not yet updated with a newly created rule, the number of instances that use an outdated model is multiplied by the throughput.

TABLE II: Memory consumption of MAMR for different datasets.

Dataset	Memory consumption (MB)	
	Avg.	Std. Dev.
Electricity	52.4	2.1
Airlines	120.7	51.1
Waveform	223.5	8

TABLE III: Memory consumption of VAMR for different datasets and parallelism levels.

Dataset	Parallelism	Memory Consumption (MB)			
		Model Aggregator		Learner	
		Avg.	Std. Dev.	Avg.	Std. Dev.
Electricity					
	1	266.5	5.6	40.1	4.3
	2	264.9	2.8	23.8	3.9
	4	267.4	6.6	20.1	3.2
	8	273.5	3.9	34.7	29
Airlines					
	1	337.6	2.8	83.6	4.1
	2	338.1	1.0	38.7	1.8
	4	337.3	1.0	38.8	7.1
	8	336.4	0.8	31.7	0.2
Waveform					
	1	286.3	5.0	171.7	2.5
	2	286.8	4.3	119.5	10.4
	4	289.1	5.9	46.5	12.1
	8	287.3	3.1	33.8	5.7

C. Memory

The reference memory consumption of MAMR for different datasets is presented in Table II. This table shows that *waveform* consumes the largest amount of memory, followed by *airlines* and then *electricity*.

Table III reports the memory consumption (*average* and *standard deviation*) of model aggregator and learner processors of VAMR for the 3 datasets with different parallelism levels. First of all, we notice a high memory consumption at the model aggregator. However this amount does not differ much for different datasets. This suggests that, at the model aggregator, there is a constant overhead in memory usage but the memory consumption due to the growing decision model is small.

Second, the memory consumption per learner decreases as more learners are used. As there is definitely some memory overhead for each learner instance, there is no significant reduction of memory usage per learner when the parallelism level goes from 4 to 8. However, the result indicates that we can spread a large decision model over multiple learners and make it possible to learn very large models whose size exceeds the memory capacity of a single machine.

VI. Conclusion

The increasing availability of data is creating new opportunities for machine learning applications. However, these applications are facing challenges brought about by data's volume, velocity and variety. SAMOA, a framework for mining big data streams in distributed environments, is designed to address these three challenges and allow extraction of useful knowledge from the available data.

In this paper, we developed and evaluated two variants of distributed AMRules. A huge advantage of decision rules is comprehensibility, required in many business decision making applications. We begin by pipelining the processing of each instances into two steps: *training* and *predicting* and assigning these steps to learner and model aggregator processors in VAMR. This approach has proved to increase the throughput for "complex" datasets. Besides, VAMR also provides memory scalability as the memory consumption of the model (the rule set) is spread among multiple learners. However, VAMR is not scalable in terms of throughput due to the bottleneck at the single model aggregator. To address this issue, we developed HAMR, an extended version of VAMR with multiple replicated model aggregators. HAMR is shown to be scalable as it can improve the throughput proportionally to the number of model aggregators while maintaining good accuracy. With a commodity cluster of 9 nodes, we achieved a throughput of more than 30 000 instances/second and a speedup of up to 4.7x over the sequential version.

One of the causes for the reduction of accuracy in VAMR and HAMR is the existence of cycles in the topologies, i.e., model aggregators send instances to learners and learners send back update messages to model aggregator. These cycles delay the update of the rules in model aggregators. Although the reduction of accuracy in our evaluation is negligible, it is expected to grow as the the throughput increases. Therefore, a design with an acyclic topology is desired to improve the accuracy, especially with higher parallelism level or higher throughput. Alternatively, support for priority in streams could also alleviate this problem.

This work can be extended by studying how to implement distributed decision rules in the more challenging settings of multi-target learning and structured learning, where instead of predicting only one attribute value, we need to predict a completes set of output attribute values.

Acknowledgment

Authors acknowledge financial support European Commission through the project MAESTRA (Grant number ICT-2013-612944).

VII. References

[1] G. De Francisci Morales and A. Bifet, "SAMOA: Scalable Advanced Massive Online Analysis," *JMLR: Journal of Machine Learning Research*, 2014. [Online]. Available: http://samoa-project.net

[2] E. Almeida, C. Ferreira, and J. Gama, "Adaptive model rules from data streams," in *ECML-PKDD '13: European Conference on Machine Learning and Knowledge Discovery in Databases*, 2013, pp. 480–492.

[3] J. Duarte and J. Gama, "Ensembles of Adaptive Model Rules from High-Speed Data Streams," in *BigMine '14: 3rd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, 2014, pp. 198–213.

[4] J. Fürnkranz, D. Gamberger, and N. Lavrac, *Foundations of Rule Learning*, ser. Cognitive Technologies. Springer, 2012.

[5] E. Ikonomovska, J. Gama, and S. Džeroski, "Learning model trees from evolving data streams," *Data Mining and Knowledge Discovery*, vol. 23, no. 1, pp. 128–168, 2011.

[6] P. Domingos and G. Hulten, "Mining high-speed data streams," in *KDD '00: 6th International Conference on Knowledge Discovery and Data Mining*, 2000, pp. 71–80.

[7] A. Shaker and E. Hüllermeier, "IBLStreams: A system for instance-based classification and regression on data streams," *Evolving Systems*, vol. 3, no. 4, pp. 235–249, 2012.

[8] C. Stanfill and D. Waltz, "Toward memory-based reasoning," *Communications of the ACM*, vol. 29, no. 12, pp. 1213–1228, 1986.

[9] E. Almeida, P. Kosina, and J. Gama, "Random rules from data streams," in *SAC '13: 28th Annual ACM Symposium on Applied Computing*, 2013, pp. 813–814.

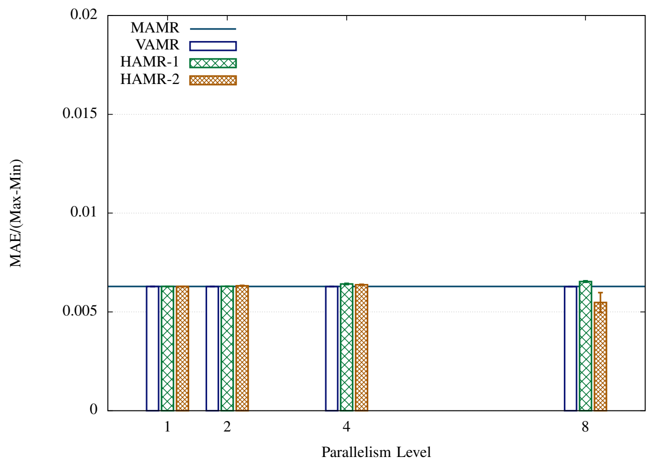
[10] G. De Francisci Morales, "SAMOA: A Platform for Mining Big Data Streams," in *RAMSS '13: 2nd International Workshop on Real-Time Analysis and Mining of Social Streams @WWW '13*, 2013.

[11] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.

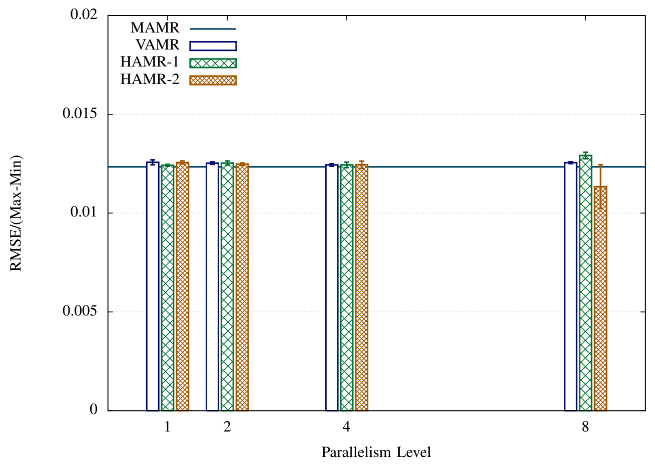
[12] E. Page, "Continuous inspection schemes," *Biometrika*, pp. 100–115, 1954.

[13] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Machine Learning*, vol. 90, no. 3, pp. 317–346, 2013.

[14] K. Bache and M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

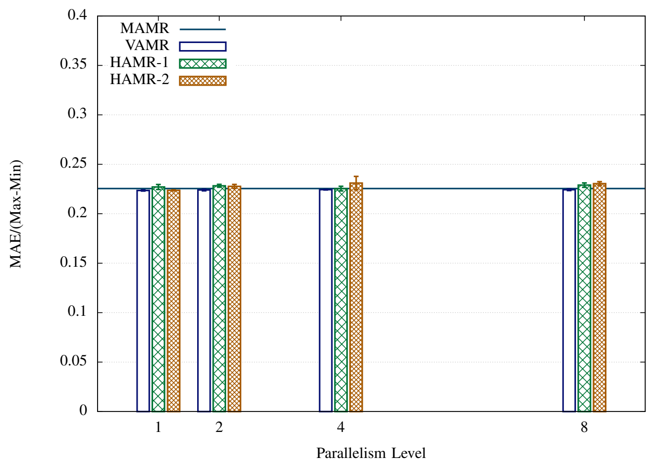


(a) MAE

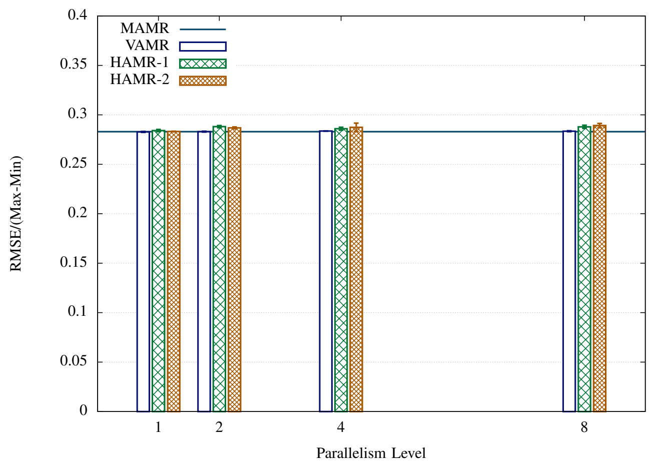


(b) RMSE

Fig. 10: MAE and RMSE of distributed AMRules with *airlines* dataset.



(a) MAE



(b) RMSE

Fig. 11: MAE and RMSE of distributed AMRules with *waveform* random generator.