Incremental Data Partitioning of RDF Data in SPARK

Giannis Agathangelos[1], Georgia Troullinou[1], Haridimos Kondylakis[1], Kostas Stefanidis[2], and Dimitris Plexousakis[1]

[1] FORTH-ICS, Heraklion, GR, {jagathan,troulin,kondylak,dp}@ics.forth.gr
[2] University of Tampere, Tampere, FI, kostas.stefanidis@uta.fi

Abstract. Significant efforts have been dedicated recently to the development of architectures for storing and querying RDF data in distributed environments. Several approaches focus on data partitioning, which are able to answer queries efficiently, by using a small number of computational nodes. However, such approaches provide static data partitions. Given the increase on the continuous and rapid flow of data, nowadays there is a clear need to deal with streaming data. In this work, we propose a framework for incremental data partitioning by exploiting machine learning techniques. Specifically, we present a method to learn the structure of a partitioned database, and we employ two machine learning algorithms, namely Logistic Regression and Random Forest, to classify new streaming data.

1 Introduction

The recent explosion of the Data Web and the associated Linked Open Data (LOD) initiative have led to an enormous amount of widely available RDF datasets [2, 4, 5] . To efficiently store, manage and query these ever increasing RDF data, new clustered RDF database systems are constantly developed and produced [1], whereas when focusing on streaming data, incremental partitioning approaches are of crucial importance. A common way of incremental partitioning is to follow hash partitioning. For example, [8] adopts hash partitioning on triples subjects using MapReduce. [9] applies a graph partitioning approach for streaming RDF data. Query driven partitioning [3] leverages query knowledge to partition data so as to answer queries by single node computations.

Our approach combines the classical predicate and subject based partitioning along with the query workload knowledge. With this combination, we maximize the intra node execution when it comes to the chosen queries, but also other similar queries that contain combinations of predicate and subject categories we have seen so far. We manage and partition the incoming data incrementally, using machine learning techniques. Specifically, we demonstrate a method to learn the structure of a partitioned knowledge base eliciting its properties, and then classify the new streaming data to the appropriate computational nodes. We performed preliminary experiments using Logistic Regression and Random Forest for classification, and show the effectiveness of these algorithms in the incremental partition procedure.

2 Incremental Partitioning

Data partitioning is thorny issue in distributed RDF data storage. A step further, the classification of new incoming data to a distributed database should also follow the same

policy in order to maintain the efficiency of the computational environment. Our goal is to extract the properties of partitioned data and learn from this structure in order to classify effectively streaming data respecting the existing distribution. The architecture of our incremental partitioning framework consists of two major components:

Data Manager: This component manages the distributed environment that consists of a set of computational nodes that interact to issue queries on existing data. As an RDF dataset is a collection of triples, usually in distributed environments, triples are partitioned across a cluster of machines and at querying, graph patterns are queried in parallel. Existing approaches try to minimize inter-machine communication during querying processing e.g. via vertical partitioning , partitioning triples based on their subject , or by combining different parts of the triples [6]. These techniques guarantee that all triples sharing a common property, i.e. a predicate, are stored on the same machine. We assume in our environment, the partitioning is performed using a combination of subject and predicate. Thus, the triples that contain the same combination of predicate and the corresponding instances of the domain classes can be accessed locally.

Incremental Partitioner: This component deals with incremental incoming data, selecting the appropriate computational node to store the corresponding triples. The functionality offered, is based on a machine learning classifier that assimilates the structure of the distributed database and assigns effectively the new triples to partitions.

Dataset Creation. A basic goal of a distributed database is to answer queries using a small number of computational nodes. Thus, triples found in same queries should be stored in the same machine. Based on this idea, we construct the dataset for training a classifier. The interesting part of this procedure is how to transform data and queries to samples, features and categories. Since efficient and effective query answering is the main goal of the partitioning process, queries should guide data distribution as well. As such, we select the user queries to represent the features of our dataset. In turn, the triples of our knowledge base will be the samples. Specifically, each sample is represented by a vector of binary values that corresponds to the existence of the specific sample/triple in each particular feature/query. Figure 1 depicts the stages for the construction of the final dataset for the classifier. Parsing each user query, we collect subjects and predicates that appear in each triple pattern and create the corresponding matrix. Each entry represents a pair (of a predicate and the corresponding domain class) that its instances are identified within the queries. In case that one subject or a predicate uses a variable, we generalize to every possible combination of the corresponding subject class with all predicates that have this class as a domain and vice versa. Thus, we use the produced pairs (of query predicates/domains) to create the first form of our dataset. However, we are interesting in creating a set of samples derived from the (instance) triples and the associated label/category for each triple. So, in a second step, all triples that have the predicate and the instances of its domain classes are represented by the corresponding feature vector of their domain-predicate pair. Then, each sample is assigned to the computational node to which the corresponding triple belongs.

Classification. Next, we train the classifier and estimate its expected performance. Since the train procedure is crucial for the performance of the predictor, we have to select the qualified parameters of the classifier that maximize its accuracy. This selection was done with the procedure of k-Fold Cross-Validation [7].

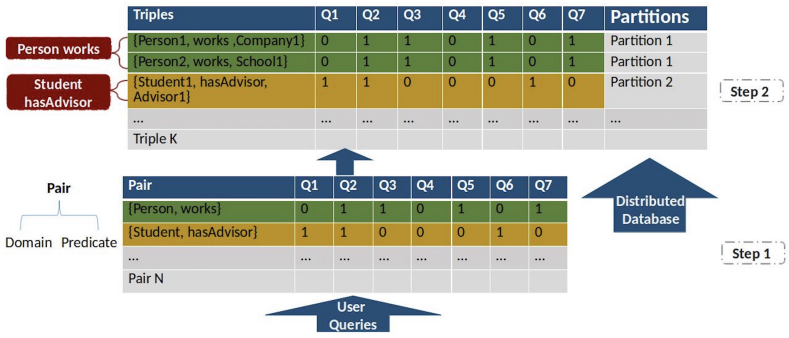

Fig. 1: Dataset construction.

3 Preliminary Evaluation & Conclusion

To evaluate our approach, we used a part of the 3.8 version of DBpedia. To create our vector space, we exploited the query logs from the corresponding DBpedia endpoints and got access to more than 50K user queries (features) for a specific period of time. Specifically, our dataset consists of 1.9M triples derived from the triples contained in the corresponding DBpedia user queries. We initially distributed data to computational nodes, using k-means, the most widely adopted clustering algorithm. Euclidean distance was used as a distance metric for k-means to assign each triple to one of 16 computational nodes based on the existing features. In this part we do not intend to evaluate the performance of the partitioning algorithm, but the correct categorization of the new incoming triples. Thus our approach can be adapted to any partitioning algorithm.

We implemented our system in Apache Spark. Spark has been set up in a clustered environment of 4 computational nodes, each of them equipped with 230GB of memory and a 38 core processor. For evaluation, we consider accuracy, precision and recall.

Algorithms: To model our problem as a classification task, we used two well-established classifiers, Random Forest and Logistic Regression. Both algorithms can handle the large number of features that we are dealing with. To find the best parameters for our algorithms, we implemented a 5-Fold Cross Validation[3].

Preliminary Results: Our dataset, by its nature, contains large number of duplicates, since large number of triples are instantiated under the same predicate-domain combination. This condition offers an efficient categorization of triples in different machines; correlated triples are placed in the same node. Nevertheless, due to the many duplicates, a classifier may not be able to classify efficiently new data. In our evaluation, we used as test samples data that in their majority has already been seen in the training procedure (Case I), while in a different scenario, we dealt with data unknown for the classifier (Case II). The dataset used in Case I contains the 20% of the original dataset, while in Case II we have a much smaller subset since we need samples that do

[3] The best selected parameters for the final training are, for Random Forest, Max Depth: 10 and Number of Trees: 60, and for Logistic Regression, Regularization: 0.01, Max Iterations: 30 and Elastic Net Parameter: L2.

not overlap with the train part. In Case I (Table 1), clearly both classifiers predict accurately, as they give an accuracy of 0.99. Due to the fact that a model classifies accurately the data instantiated to the majority of predicate-domain pairs, the classifier succeeds in categorizing data commonly queried by users. This is a crucial component of data partitioning. In Case II, we examine triples unknown to the classifier and we observe that the resulting metrics are as good as the first case for both classifiers. Thus, new incoming triples can be partitioned effectively in the distributed environment. Examining further Logistic Regression results, we observe that in Case I there is a small False Positive Rate since we can find samples that do not belong in their actual class. In the second case we do not observe the same result since the smaller sample size, results in statistically less plausible False Positives.

Table 1: Algorithms evaluation.

	Case I		Case II	
	Random Forest	Logistic Regression	Random Forest	Logistic Regression
Precision	0.937	0.999	0.897	1.0
Recall	0.932	0.999	0.90	1.0
Accuracy	0.998	0.999	0.985	1.0

To conclude, in this paper, we propose an approach that combines machine learning algorithms and data partitioning techniques to classify data incrementally, and show the feasibility of our solution. As future work, we plan to deploy our work in a real clustered environment and measure the actual improvement on query execution times, comparing our solution with other competitive approaches.

References

1. G. Agathangelos, G. Troullinou, H. Kondylakis, K. Stefanidis, and D. Plexousakis. RDF query answering using apache Spark: Review and assessment. In *IEEE ICDE*, 2018.
2. V. Christophides, V. Efthymiou, and K. Stefanidis. *Entity Resolution in the Web of Data*. Synthesis Lectures on the Sem. Web: Theory and Technology. Morgan & Claypool, 2015.
3. K. Hose and R. Schenkel. WARP: workload-aware replication and partitioning for RDF. In *IEEE ICDE*, 2013.
4. H. Kondylakis and D. Plexousakis. Ontology evolution in data integration: Query rewriting to the rescue. In *ER*, 2011.
5. H. Kondylakis and D. Plexousakis. Ontology evolution: Assisting query migration. In *ER*, 2012.
6. T. Neumann and G. Weikum. The RDF-3X engine for scalable management of RDF data. *VLDB J.*, 19(1):91–113, 2010.
7. P. Refaeilzadeh, L. Tang, and H. Liu. Cross-validation. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems*, pages 532–538. Springer US, 2009.
8. K. Rohloff and R. E. Schantz. High-performance, massively scalable distributed systems using the mapreduce software framework: the SHARD triple-store. In *SPLASH*, 2010.
9. R. Wang and K. Chiu. A stream partitioning approach to processing large scale distributed graph datasets. In *IEEE Big Data*, 2013.