Online Network Traffic Characterization

Deliverable
Available algorithms identification

ONTIC Project
(GA number 619633)

Deliverable D3.1
Dissemination Level: PUBLIC

Authors

Daniele Apiletti, Elena Baralis, Fabio Pulvirenti, Silvia Chiusano, Tania Cerquitelli, Paolo Garza, Luigi Grimaudo, Luca Venturini
POLITO

Bo Zhu, Alberto Mozo, Antonio Hernando
UPM

Version
ONTIC_D3.1.2015.06.04.2.50

Version History

Previous Version	Modification date	Modified by	Summary
D3.1.2014.11.27	26 Nov 2014	POLITO	First draft
D3.1.2014.12.16	16 DEC 2014	UPM	Second draft
D3.1.2015.01.15	15 JAN 2015	UPM	Final draft
D3.1.2015.01.22	22 JAN 2015	POLITO	Draft revision
D3.1.2015.01.30	30 JAN 2015	POLITO	Final review
D3.1.2015.05.30	26 MAY 2015	POLITO	Revised draft
D3.1.2015.06.04	04 JUN 2015	POLITO	Final revised

Quality Assurance

Quality Assurance Manager	**Elena Baralis, Polito**
Reviewer #1	**Bruno Ordozgoiti, UPM**
Reviewer #2	**Fernando Arias, EMC**

Table of Contents

List of figures

List of tables

1. Acronyms and Definitions

Acronyms

Acronym	Defined as
AP	Affinity Propagation
BoF	Bag of Flows
CPD	Conditional Probability Distributions
EM	Expectation Maximization
HAP	Hierarchical Affinity Propagation
HDFS	Hadoop Distributed File System
HMM	Hidden Markov Models
ML	Machine Learning
MPI	Message Passing Interface
MPP	Massively Parallel Computer
MRC	Map Reduce Class
NBD	Naïve Bayes Discretization
NBK	Naïve Bayes Kernel
NP-hard	Non-Deterministic Polynomial-time hard
PCA	Principal Component Analysis
PFP	Parallel FP-growth
RDD	Resilient Distributed Datasets
SQM	Statistical Query Model
SVD	Single Value Decomposition
SVM	Support Vector Machine
YAFIM	Yet Another Frequent Itemset Mining

2. Purpose of the Document

The purpose of this document is to analyze the available offline algorithms identified as state of the art in the context of the ONTIC project. This document contains and defines the state of the art and its limitations in terms of scalability and performance of the available algorithms. A comprehensive overview of off-the-shelf scalable techniques and current state-of-the-art distributed algorithms is provided, addressing the main tasks of WP3: clustering, classification and correlation analysis. Furthermore, a state-of-the-practice section describes the most popular frameworks and software solutions relevant in the ONTIC context.

3. Intended Audience

The intended audience of this document includes every partner within ONTIC project, especially those participating in work packages WP2, WP3, WP4, and WP5. It also includes any reader interested in knowing the technological and algorithmic foundations of ONTIC.

The readers of this document will receive information about both the state of the art and the state of the practice related to scalable clustering, classification and correlation algorithms.

4. Executive Summary

In the last decades, Internet network traffic has increased exponentially thanks to a worldwide adoption of digital services, making network traffic analysis a hot issue.

Data mining and machine learning techniques provide great potential when applied to network traffic characterization, such as QoE management, anomaly detection and congestion control. Furthermore, network traffic data analysis has to deal with massive datasets, typical examples of big data. Much effort has been spent in this direction, as witnessed by the vast scientific literature.

The framework proposed by the ONTIC project heavily relies on offline analytics in order to extract interesting data from huge collections of network traffic. Hence, the development of new and improved techniques in this context is one of the core tasks of the Project.

This document provides an overview of current, promising techniques that fall into the three main families of algorithms: frequent itemset mining and correlation discovery, unsupervised clustering and supervised classification.

The goal of clustering and, in general, unsupervised learning algorithms is to discover hidden structures in unlabeled data. Supervised learning, instead, aims at building a predictive model from an input dataset. For this kind of algorithms, a set of labelled data is required, used to build the predictive model, which is then used to classify new unlabeled samples. Correlation analysis, finally, is a data mining method used to discover relevant correlations among data. It consists of frequent itemset extraction and association rules generation.

Since the ONTIC framework is designed to cope with large amounts of data, the WP3WP3's state of the art survey is focused on the application of machine learning and data mining Techniques in scalable environments, by including promising works that rely on distributed or parallel frameworks suitable for big data. The current state-of-the-practice in distributed frameworks is based on Apache Hadoop and Apache Spark. The former is based on the MapReduce paradigm: a Map procedure performs processing and filtering on input data, and a Reduce procedure applies an aggregation function. The Hadoop MapReduce approach has been widely adopted in recent years;. However, the latest trend is rather exploiting Spark, which is more suitable for iterative tasks and algorithms that access data more than once.

The document also highlights current boundaries of the state of the art, in terms of volume and features. To this end, a thorough comparison based on common performance metrics and capabilities has been performed and is presented in the document.

The main content of the document consists of three sections.

- Tools and software solutions in the state of the practice for big data analysis (Section 5).
- Scientific review of the state of the art in big data techniques: unsupervised clustering, supervised classification, itemset extraction and association rule mining (Section 6).
- Algorithms specifically applied in the network traffic analysis domain (Section 7).

The Apache Hadoop platform, together with its extensions, such as Apache Spark, being the current de-facto standard in the big data environment, is currently considered the most promising foundation to build an offline characterization framework for the ONTIC project.

5. Tools and software solutions in the state of the practice for big data analysis

Currently, the use of big data is widespread in both the public and private sectors, as reported and detailed in Deliverable D2.2. Being able not just to collect but to analyze big data will be of huge value from both an economic and a social point of view. The importance of this skill is revealed by the number of companies involved in the field: big data has gone from being an expertise area surrounded by considerable hype and confusion, to a foundation for business and even industries.

In the data mining domain, most of the technologies involved in big data analytics have to be redesigned: often, it is not enough to simply adapt these tools to the new magnitude of the problems. Even data storage had to be adapted because relational database management systems proved to be not effective anymore: hence, in the last years we have witnessed the introduction of distributed file systems like Google File System (1) and its open source derivative HDFS (2).

For its schema-free feature, but also for the easy replication, eventual consistency, and large amount of data support, NoSQL databases have become among the most adopted in big data environments. In particular, among the column oriented databases, the ones storing data by column instead of by rows, we should mention Bigtable (3) by Google (and its open source derivative HBase (4)), and Cassandra (5) by Facebook. Document databases, instead, are able to support more complex structures: the most popular are MongoDB (6) and SimpleDB by Amazon (7).

On the other hand, MPI (8) or OpenMP (9), because of their lack of resiliency and fault tolerance, are not suitable to implement parallel programs in big data environment. Thus, many programming models have been introduced to solve specific environment applications.

MapReduce is probably the most popular example of a generic, batch-based, processing model. As already detailed in D2.2, it enables automatic parallelization and distribution of work-intensive applications over clusters of machines. Hadoop, built over Google MapReduce, is a tool which ideally fits the vast majority of big data problems, but it fails dramatically when dealing with iterative processes that must read data more than once. To address this limitation, other technologies have been developed. Apache Spark, a new distributed framework, is gaining popularity: like Hadoop MapReduce, Spark can be used to analyze data that are too large to fit into the memory of a single machine.

The success and spread of these platforms has been strongly supported by several specific machine learning libraries integrated in the frameworks. **Mahout** (10) is a scalable machine learning library with a relatively long history, containing implementations of algorithms in areas such as classification, clustering, recommendation systems, etc. Despite the fact that the majority of existing implementations are based on the MapReduce paradigm, as witnessed by Table 1, it is encouraging the utilization of modern scalable distributed systems that provide more data processing operations such as Spark. In the last version (Mahout Samsara 0.10.0), it includes all the features which have made it very popular in the last years, i.e. a solid baseline of Hadoop MapReduce algorithms, with the addiction of Spark support. Furthermore:

- It now supports R-like linear algebra operations and it is becoming synergic with H2O (11), a suite of math and predictive data analysis leveraging Hadoop and Spark, characterized by an easy to use web interface and supporting R, Python and JSON.

- It is maintained by a team of more than 30 core developers, however, a lot of contributions have been made by independent researchers, e.g. (12) and (13).
- More than 30 big commercial players adopted Mahout library (14) such as:
    - AOL (for shopping recommendations)
    - Foursquare (recommendation engine)
    - LinkedIn (model training)
    - NewsCred (clustering new articles and generates the day top stories)
    - Radoop (providing a drag-n-drop interface for big data analytics, leveraging Mahout algorithms)
    - SpeedDate.com (collaborative filtering engine to recommend member profiles)
    - Twitter (user interest modelling)
    - Yahoo! (antispam through frequent pattern mining)
- Also a lot of academies leverage Mahout for research and data processing, some of them are:
    - Carnegie Mellon University
    - ROBUST project
    - TU Berlin
    - Nagoya Institute of Technology
    - Digital Enterprise Research Institute NUI Galway

Nevertheless, Mahout has lost part of its fascinating power towards the community of developers, who are now attracted by Apache Spark and its machine learning library **MLlib** (15). Indeed, the contributions to the respective repositories witness the superiority of Spark as for developing pace, number of contributors and issues resolved (16), (17). For example, the number of open pull requests to the repositories is more than 6 times larger for MLlib today.

MLlib tends to offer more efficient implementations compared with other scalable ML libraries, especially for iterative algorithms (Table 2). Depending on whether all the necessary information can be kept in memory, users can select from different storage levels, including using memory only, using both memory and disk, and store RDD as either de-serialized or serialized objects. Furthermore, MLlib:

- Supports Java, Scala and Python, being fully compatible with HDFS data source
- Runs off the shelf on YARN, a resource manager already set up on existing Hadoop clusters, but it is also ready to run standalone or on a single node
- It is adopted by many big companies (18), (19), which have also contributed to the development of the algorithms, such as:
    - AliBaba (graph inspection / PageRank)
    - Concur (travel expenses analytics and personalization)
    - Flytxt (telecommunications profiling)
    - Pearson (stream analysis of student performance)
    - Technicolor (batch and analysis of IoT)

- Twitter (pair similarities)

Mahout 0.10.0 - Available data mining algorithms			
Type	**Single machine implementation availability**	**MapReduce**	**Spark**
Classification			
Logistic Regression - trained via SGD	Yes	No	No
Naive Bayes / Complementary Naive Bayes	No	Yes	Yes
Random Forest	No	Yes	No
Hidden Markov Models	Yes	No	No
Multilayer Perceptron	Yes	No	No
Clustering			
K-means Clustering	Yes	Yes	No
Fuzzy K-means	Yes	Yes	No
Streaming k-means	Yes	Yes	No
Spectral Clustering	No	Yes	No
Dimensionality Reduction			
Singular Value Decomposition	Yes	Yes	Yes
PCA (via Stochastic SVD)	Yes	Yes	Yes
QR Decomposition	Yes	Yes	Yes

Table 1: Data mining algorithms provided in Mahout

Weka (20) is a traditional data mining tool that contains various implementations of many ML algorithms. It used to deal with modest-sized dataset and got good results. However, Weka's latest versions provide several wrapper packages based on both online and offline frameworks like MOA (21), Hadoop, etc. endowing it with the capability to conduct big data analysis.

Finally, **MADlib** (22) provides a SQL toolkit for large scale dataset analysis that runs on top of Hadoop. Data processing is performed locally while parallelism and scalability are achieved by utilizing MPP shared-nothing techniques. MADlib consists of many implementations of ML algorithms in the fields of classification, clustering, regression, topic modelling, association rule mining etc.

MLlib - Spark version 1.3.1 – Available data mining algorithms	
Type	**Algorithms**
Classification and Regression	SVMs, logistic regression, linear regression
	Naive Bayes
	Decision trees
	Random Forests and Gradient-Boosted trees
Clustering	K-means
	Gaussian mixture
	Power iteration clustering
	Latent Dirichlet allocation
	Streaming k-means
Dimensionality reduction	Singular value decomposition (SVD)
	Principal component analysis (PCA)
Frequent Pattern Mining	FP-growth

Table 2: Data mining algorithms provided in MLlib

6. Scientific review of the state of the art in big data techniques

Data mining and machine learning algorithms aim at extracting previously unknown, interesting information, such as dependencies (frequent itemset mining and association rules), groups of data objects (clustering algorithms) and categories of new observations (classification and regression).

In recent years, applying machine learning techniques in the network context has often entailed working on a huge amount of data like network traffic datasets (e.g. the data extracted with network analysis tools Tstat (23), NetFlow (24), etc.). These types of databases are often so large that they are a typical example of big data. In these cases, traditional approaches are starting to show their limitations.

Furthermore, the shift towards horizontal scaling in hardware has highlighted the need of parallelization in data mining and machine learning techniques. Hence, recently an increasing number of researchers have adopted the MapReduce (2) approach.

The main idea of MapReduce is to split the processing of data into independent parallel tasks; as presented in Section 5, Apache Hadoop (25) is one of the most popular MapReduce frameworks. We can find an increasing number of distributed MapReduce implementations of many different algorithms in data mining, such as (26), (27) and (13).

Apache Spark (28) (briefly presented in Section 5, see Deliverable D2.2 for deeper details), with its Resilient Distributed Datasets and its smart APIs, outperforms Hadoop MapReduce in terms of performance (execution time) and overcomes its limitations, with particular focus on iterative in-memory computation, which is a common characteristic of many data mining algorithms.

This section presents a scientific review of the state of the art that is intentionally focused on distributed algorithms exploiting the above-mentioned parallel frameworks.

6.1 Scalable algorithms for unsupervised clustering

The target of unsupervised learning is to discover hidden structures in unlabeled data. In particular, clustering techniques aim at grouping sets of objects in such a way that objects in the same groups (clusters) are more similar to each other than to those in other groups (clusters). Cluster analysis divides data into groups, attempting to capture the natural structure of the data. The greater the similarity or the homogeneity within a group, and the greater the difference between the groups, the better the clustering result is.

In this analysis we define a set of features to describe, evaluate and compare each algorithm. The aim is to deliver an evaluation that follows criteria inherited from data miningthe data mining and big data domains.

The first set of criteria is related to specific characteristics of the clustering algorithms. These criteria are the following.

- **Family**: in literature, clustering algorithms are grouped into families, depending on the basic intuition that drives the cluster generation. For instance, for the prototype-based clustering algorithms, a cluster is a set of objects in which each object is closer (more similar) to the cluster prototype, a specially-selected object among the others. For density-

based algorithms, instead, a cluster is a dense region of objects that is surrounded by a lower density region. Among all the clustering families, we focus our analysis on prototype-based, density-based, hierarchical, spectral, and sampling-based approaches.

- **Underlying centralized algorithm**: each distributed clustering approach is typically based on an underlying centralized algorithm, which is extended and adapted to be suitable for the distributed context.
- **Similarity measure**: to identify the points that are similar and group them into the same cluster, a crucial choice for clustering algorithms is distance computation, the closer the distance between points, the more similar they are. The similarity measure can strongly affect clustering results. The Euclidean distance (29) is the most intuitive and popular choice, as reflected by Table 3, Table 4 and Table 5, but many other types of distance exist. Given two points p and q, the Euclidean distance between p and q is the length of the line segment connecting them: $d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$. Another common distance measure is the cosine distance (29), computed between two vectors p and q, and defined as: $d(p,q) = \frac{p\ q}{||p||\ ||q||}$.
- **Input parameters**: some popular clustering techniques (e.g., K-means) require the user to specify the desired number of clusters, whereas this is often a result that the user would like the algorithm itself to determine; many other input parameters are very challenging to tune and may strongly affect the final result. This problem is worsened in the big data environment because it is very costly to execute the algorithm many times and also the simplest pre-processing phase is very data and resource intensive. For each technique, our analysis will identify the most important input parameters, in order to highlight the ease of the tuning up.

The second set of evaluation criteria is related to the distributed nature of the processing. They are often underestimated in the data mining context but represent very critical issues (30), (31):

- **Load balancing**: an unbalanced load undermines the advantages of a parallel environment as the overall execution time is that of the slowest, most loaded node; in a fully unbalanced environment, the worst case scenario leads to no benefits from parallelization while still incurring all the overheads of coordinating a rather complex distributed system.
- **Communication costs**: even if they are often underestimated, they can overwhelm computational costs and represent the most likely bottleneck of a distributed system, (32).

Finally, we present a global performance comparison set of tables (Table 8, Table 9 and Table 10) based on the experimental evaluation section of each reference, identifying the algorithms with the best performance in big data environments.

We note that up to now the research community has focused on the implementation of distributed versions of previously-known centralized techniques. Hence, the innovation and the scientific contribution of the works under evaluation in this survey predominantly consist in the adaptation to distributed and parallel environments.

The following subsections are separated according to clustering algorithm families: prototype-based clustering, density-based, hierarchical, and so on. Finally, a performance evaluation is presented and open issues are discussed.

6.1.1 Prototype-based clustering

Prototype-based clustering is based on the basic idea of partitioning the dataset into a set of objects in which each object is closer (more similar) to the prototype that defines the cluster than to the one of any other clusters, in order to minimize a criterion (e.g. sum of squared distance). For many types of data, the prototype can be regarded as the most central point (in such a case, clusters tend to be globular). K-means represents the most popular approach of this clustering family: each cluster is identified by a centre (centroid). The algorithm follows an iterative procedure, creating K partitions and assigning each input point to the cluster with the “closest” centre. After that, for each cluster, the centroid is recomputed from the new cluster distribution. The algorithm stops when the maximum number of iterations is reached or no further changes are required.

The most popular distributed implementation of K-means is provided in the Apache Mahout library, which is a very popular machine learning library (10) and is further discussed in Section 5. The implementation is very straightforward: the key idea is to exploit more than one node to compute the new set of centroids, thus distributing both computation and data.

- The points are assigned to the closest centroid.
- Each node receives only the data related to a single cluster.
- Each node independently computes its new centroid.

As reported in Table 3, the algorithm supports many types of distance measure, such as Euclidean, Manhattan (and more generally Minkovski), and Mahalanobis, thus making the algorithm flexible enough to be exploited in different contexts. Besides the type of distance measure, the main input parameter needed by the distributed K-means is the same as with its centralized version: the desired number of clusters. The Mahout-provided implementation addresses this issue with a Canopy Clustering (33) option, which is a very fast but approximated technique able to guess a suitable number of clusters, provided that the user defines two distance thresholds.

The iterative nature of the algorithm makes its MapReduce implementation sub-optimal: at each iteration all the data must be read (again) from the disk. Furthermore, both communication costs and load balancing are difficult to address: most of the dataset must be transmitted through the network to the different nodes, thus increasing the communication costs, while load balancing is strictly related to the dataset distribution: larger clusters are inherently more resource intensive than smaller clusters.

Another distributed K-means implementation is provided by Apache Spark in its machine learning library MLlib (15) (detailed in Section 5). It follows the same overall structure as the Mahout implementation. Differences include the support for a single distance measure (Euclidean), and an option to optimize the initial centroid choice: the basic idea is to smartly sample the initial dataset and run a local K-means on the samples to find the best centroids for the initial iteration (34). The usage of Spark RDDs (Resilient Distributed Dataset, detailed in Section 5 and also in D2.2) partially solves the challenges related to the iterative nature of the algorithm. The key improvement over the Mahout implementation is the read-once approach (if the dataset fits in the main memory of the overall cluster). Load balancing and communication costs are the same as the Mahout implementation: all the points belonging to the same cluster are processed in the same node, even if within a unique RDD.

Further implementations of distributed K-means are available, such as G-means (35), which introduces a K-means implementation based on the Mahout version without the need for the number of clusters as input parameter. As shown in Figure 1, G-means exploits a normality test

on the dataset distribution to decide whether a cluster should be split. The basic idea is that K-means clusters should follow a Gaussian distribution. The distributed implementation spreads the computation costs related to the normality test and the new centroid processing (the latter is the same as the Mahout base version), and also takes into account communication costs.

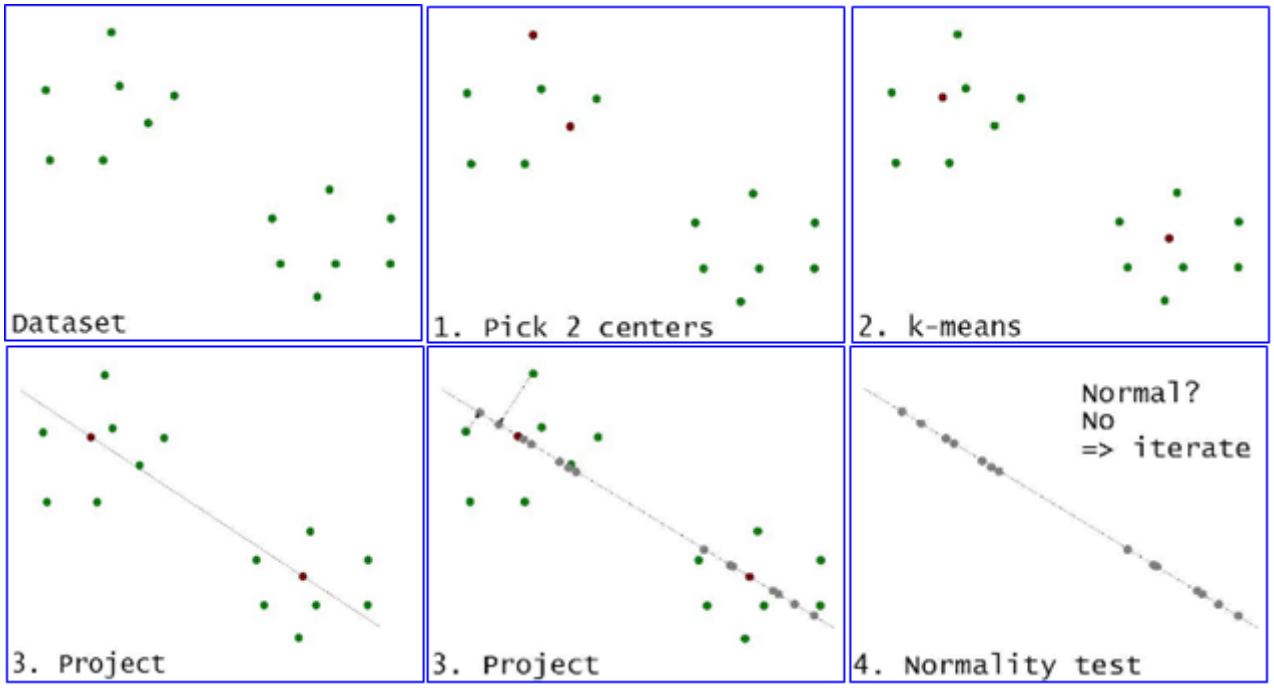

Figure 1: G-means strategy to determine the number of clusters (35).

A different approach is presented in (36), which exploits a K-median and K-centre Hadoop implementation. While the K-means utility function minimizes the within-cluster sum of squares, the K-median clustering minimizes the total distance from the centers. The K-center utility function, instead, aims to minimize the maximum radius of the clusters. The work consists of two basic ideas. The first is to sample the dataset, and ignore all the data points close to the samples, assuming they are already represented by the samples themselves. Hence, computational costs are reduced. The second idea is to partition the dataset and extract k centers for each partition, then re-cluster, thus improving the scalability: in this way, a node can process only a shard of the whole dataset, achieving a complete independency from the other nodes and communicating only the results of the clustering. The available distance measures are not clearly stated and the algorithm still needs the number of clusters as input parameter. Unfortunately, load balancing and communication costs are not addressed (the reported evaluations are performed on a single machine).

K-means and prototype-based algorithms						
Reference	**Framework**	**Underlying Algorithm**	**Distance Measure**	**Comm. cost handling**	**Load balancing handling**	**Input Parameters**
K-means Mahout implementation (10)	Hadoop	K-means, iterative	Euclidean, Manhattan, Mahalobis and Minkovski	NA	NA	Number of clusters
K-means MLlib implementation (15)	Spark	K-means	Euclidean, Manhattan	NA	NA	Number of clusters
Determining the k in K-means with MapReduce (35)	Hadoop	K-means	Euclidean	Yes	No	None
Fast Clustering Using MapReduce (36)	Hadoop	K-median, K-centroid	NA	No	No	Number of clusters

Table 3: K-means and other prototype-based distributed algorithm comparison.

6.1.2 Density based clustering

Density based techniques have been largely adopted in literature: the basic idea is that clusters are defined as areas of higher density than the remainder of the data set. Objects in these sparse areas, that are required to separate clusters, are usually considered to be noise or border points. This kind of definition of a cluster is often employed when the clusters are irregular (e.g., non-globular) or intertwined, and when noise and outliers are present. DBSCAN (37) is the most popular density-based algorithm. It is a center-based approach, in which density is estimated for each point counting the neighbors within a specified radius. The points are hence partitioned into core points (if the number of neighbors are over a certain threshold, called min-points), border points (if the point is not a core point but it belongs to the neighborhood of a core point) and outliers (neither a core point nor a border point, they are also called noise points). The algorithm merges in the same cluster the core and border points closer than the input maximum radius, while noise points are discarded.

As shown in Table 4, (26) introduces a Hadoop-based DBSCAN implementation: the basic idea is to distribute subsets of the original dataset across the nodes, run a local clustering phase, and finally merge the resulting local clusters to obtain the overall result. Hence, the key feature of the algorithm is the possibility to compute local clusters in parallel. To this aim, each node runs a local DBSCAN only on a subset of the original points: the partitioning is done in a preliminary MapReduce job exploiting a strategy based on feature discretization and buckets, trying to minimize the border points and to spread the points as evenly as possible. Centralized DBSCAN has two input parameters: the radius and the min-points, the same as the distributed version. Unfortunately, their choice is very critical, representing a challenge for the user, and the distributed version does not address or improve this limit. However, it provides a pre-processing step aimed at enhancing load balancing and communication costs.

A Spark-based implementation of DBSCAN is available at (38). It provides both the Euclidean and the Manhattan distances. Unfortunately, no publication describes this implementation.

DBCURE, presented in (39), follows the same paradigm as the Hadoop-based DBSCAN implementation (local clustering on each node and final merge of results). It exploits ellipsoidal τ-neighborhoods of points to be less sensitive to density changes (example in Figure 2 where it is compared with a DBSCAN implementation). The computation load of merging points belonging to dense clusters is spread across different nodes. Finally, the merging is performed at the cluster level, without considering the single points one by one, thus further reducing computation costs. The distance measure adopted is the Euclidean distance, and the input parameters are the same as DBSCAN. Load balancing and communication costs are not directly addressed.

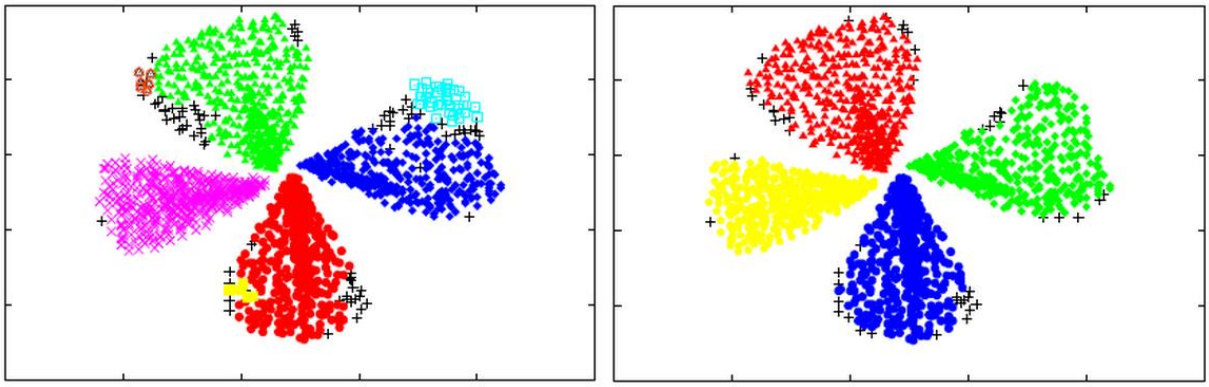

Figure 2: **Effectiveness of** DBCURE and DBCURE-MR **(39)** against DBSCAN (on the left).

Density-based algorithms						
Reference	**Framework**	**Underlying Algorithm**	**Distance Measure**	**Comm. cost handling**	**Load balance handling**	**Input parameters**
MR-DBSCAN (26)	Hadoop	DBSCAN	Euclidean	Yes	Yes	Density radius and number of neighbors
Spark DBSCAN (38)	Spark	DBSCAN	Euclidean, Manhattan	NA	NA	Density radius and number of neighbors
DBCURE-MR (39)	Hadoop	CURE	Euclidean	No	No	Ellipsoidal Density radii and number of neighbors

Table 4: Density-based distributed algorithm comparison

6.1.3 Hierarchical approaches, spectral clustering and sampling

Other popular categories of clustering are hierarchical, spectral and sample-based algorithms.

Unlike other categories of clustering algorithms, which only provide a flat partitioning of the input data set, **hierarchical clustering** algorithms build a “dendrogram” that reflects a hierarchy of

clusters. Clusters are usually merged with each other at a certain level of similarity (distance), but divisive implementations (29) also exist. Traditional hierarchical algorithms are not intuitive choices for MapReduce implementation: the iterative processes do not fit the paradigm and the construction process of the dendrogram causes high communication cost.

In (40) the authors introduce a distributed feature selection and hierarchical clustering algorithm on top of Hadoop. The algorithm is basically composed of iterations which begin with calculating the similarity of every pair of points and finding the most similar ones; then they are merged and the similarity matrix – a matrix of scores that represent the similarity among data points – is updated. The Pearson correlation is used as distance measure, while no input parameters are specified (only the parameters related to the feature selection phase are clearly stated). Load balancing is not taken into account while I/O and communication costs are reduced by the so called batch update, which is considered the main innovation of the work: the idea is to combine several iterations into one, so that the calculation and the processing of the similarity matrix are accelerated in batch mode, consequently reducing both the I/O and communication costs.

Affinity propagation is a novel clustering technique and consists in finding a set of "exemplars" that can represent the rest of data points. Iterative updates are conducted by passing messages between data points, taking into consideration all data points as potential exemplars. (41) introduces the most representative MapReduce implementation of a Hierarchical Affinity Propagation technique. The basic idea is to split the updates into parallel jobs where each job receives a subset of the input data. After a given number of iterations, the cluster memberships are extracted. The approach exploits the Euclidean distance as dissimilarity measure. The approach, based on Affinity propagation (a member of the prototype based family of clustering algorithms), differs from the other prototype based techniques because it does not require the number of clusters as input parameter. However, it is a hierarchical technique and requires the number of hierarchies as input parameter, which is a less critical feature because it does not affect the results of the clustering within the same level. Communication costs are not addressed, whereas load balancing is discussed.

Spectral clustering is based on graph theory. It is able to identify clusters of arbitrary shapes and aims at global optimum convergence. Its main idea is to perform the clustering using the eigenvectors obtained by conducting Eigen decomposition of the similarity matrix of a sample set: (42) introduces a parallel spectral clustering algorithm that uses both MapReduce and Message Passing Interface (MPI). The Euclidean and Cosine distance measures are exploited. The input parameters are the number of clusters and the number of samples from the matrix. As reported in Table 5, communication cost is an evaluation parameter, while load balancing is not discussed.

A completely different approach is introduced by BoW (43). In this work, communication and disk I/O costs are taken into account upfront: the proposed framework is able to infer the best strategy between a **sampling** approach or a full dataset processing, by means of a deep analysis of the dataset features and cluster node parameters. For this reason, a detailed description of the cluster hardware is required as input parameter. The distance measure and the underlying algorithm used for clustering are not clearly specified. However, its contribution is provided by the proposed sampling strategy: the basic idea is to sample the dataset and to cluster only the samples on a single machine, in order to extract the most general high-level clusters. Then, only the points which do not belong to any of the already identified clusters are transmitted through the network to the rest of the nodes in the cluster, in order to reduce the communication costs. The balance of the work among the nodes is also taken into account.

Other families of clustering algorithms						
Reference	**Framework**	**Underlying Algorithm**	**Distance Measure**	**Comm. cost handling**	**Load balance handling**	**Input parameters**
An Efficient Hierarchical Clustering Method for Large Datasets with MapReduce (40)	Hadoop	Hierarchical clustering / feature selection	Pearson Correlation	Yes	No	NA
Parallel Hierarchical Affinity propagation (41)	Hadoop	Affinity propagation	Euclidean	No	Yes	Number of hierarchies - No need of number of clusters
Parallel Spectral Clustering in Distributed Systems Pattern Analysis and Machine Intelligence (42)	Hadoop	Spectral Clustering	Euclidean, Cosine	Yes	No	Number of clusters / Number of samples
Bow (43)	Hadoop	Sampling	NA	Yes	Yes	Hardware configuration

Table 5: Comparison for other families of distributed clustering algorithms.

6.1.4 Per framework summary

In Table 6 and Table 7 the available references are reported separately for each framework (i.e., Hadoop and Spark).

Hadoop-based clustering algorithm implementations						
Reference	**Family**	**Underlying Algorithm**	**Distance Measure**	**Comm. cost handling**	**Load balance handling**	**Input parameters**
K-means Mahout implementation (10)	Prototype-based	K-means, iterative	Euclidean, Manhattan, Mahalobis and Minkovski	NA	NA	Number of clusters
Determining the k in k-means with MapReduce (35)		K-means	Euclidean	Yes	No	None
Fast Clustering Using MapReduce (36)		K-median, K-centroid	NA	No	No	Number of clusters
MR-DBSCAN (26)	Density-based	DBSCAN	Euclidean	Yes	Yes	Density radius and number of neighbors
DBCURE-MR (39)		CURE	Euclidean	No	No	Ellipsoidal Density radius and number of neighbors
An Efficient Hierarchical Clustering Method for Large Datasets with MapReduce (40)	Hierarchical	Feature selection	Pearson Correlation	Yes	No	NA
Parallel Hierarchical Affinity propagation (41)		Affinity propagation	Euclidean	No	Yes	Number of hierarchies - No need of number of clusters
Parallel Spectral Clustering in Distributed Systems Pattern Analysis and Machine Intelligence (42)	Spectral		Euclidean, cosine	Yes	No	Number of clusters / Number of samples
Bow (43)	Sampling-based		NA	Yes	Yes	Hardware configuration

Table 6: Hadoop-based implementations of distributed clustering algorithms

Spark-based clustering algorithm implementations						
Reference	**Family**	**Underlying Algorithm**	**Distance Measure**	**Comm. cost handling**	**Load balance handling**	**Input parameters**
K-means MLlib implementation (15)	Prototype-based	K-means	Euclidean, Manhattan	NA	NA	Number of clusters
Spark DBSCAN (38)	Density-based	DBSCAN	Euclidean, Manhattan	NA	NA	Density radius and number of neighbors

Table 7: Spark-based implementations of distributed clustering algorithms

6.1.5 Performance evaluation

In this section we compare the experimental section of each paper. As reported in Table 8, Table 9, and Table 10, the performance evaluation will be driven by the analysis of the datasets used for the evaluation, the code implementation, the dataset availability, and the cluster dimensions and speedup. All the reported information is extracted from the papers. The dataset features and its open availability are useful to foster open comparisons with other techniques, especially if the algorithm implementation is not freely available. The cluster dimension and the horizontal scalability are important issues in parallel environments: the advantage of having a large amount of machines can be mitigated by the overhead and communication costs related to distributed frameworks. Ideally, given a fixed-size dataset, we should be able to distinguish two phases in the scalability performance evaluation: a first phase in which the execution time decreases with the addition of machines, and a second phase where the limit of scalability is reached and the addition of new machines is not effective. In the best case, the first phase is linear and the second is steady. This said, it is clear that an analysis of the horizontal scalability should investigate whether the first phase is linear or sublinear, the slope of the speedup (if linear), the number of machines at which the speedup stops improving, and how this affects the performance (steady or decaying). Experiments designed to analyze horizontal scalability should therefore be replicated over a growing number of machines, enough to interpolate a line and until the performances stop improving. Unfortunately, it is often the case that, in the environments used in the evaluations, it is challenging to reach the limit for horizontal scalability and identify two distinct phases or detect a clear trend.

We do not include cluster quality measures because all approaches claim to achieve very high quality performance on their own use cases, and a fair comparison would be extremely difficult due to the heterogeneity of conditions.

Among the prototype-based clustering techniques, the K-means implementation of (35) has shown to be reliable with benchmarks on synthetic datasets of up to 10 million points and ten dimensions. The algorithm implementation is freely available. It was evaluated on a commodity cluster of 12 nodes reaching a linear speedup. The base MLlib and Mahout K-means implementations do not publicly provide any official performance evaluation.

In (41) the algorithms were tested against pictures and, in the scalability evaluation, a dataset consisting of 788 bidimensional points was used. The code implementation is available, and it has

been tested on up to 80 nodes, showing a linear speedup only in the initial phase (with less than 40 nodes).

K-means and prototype-based algorithms - Performance evaluation				
Reference	**Evaluation Dataset Size**	**Dataset availability**	**Implementation availability**	**Scalability evaluation and speedup**
K-means Mahout implementation (10)	NA	NA	Yes	NA
K-means MLlib implementation (15)	NA	NA	Yes	NA
Determining the k in K-means with MapReduce (35)	10 million points with 10 features	No	Yes	12 nodes with a linear speedup
Fast Clustering Using MapReduce (36)	10 million points with 3 features	No	No	1

Table 8: Prototype-based distributed algorithm performance evaluation

Among the DBSCAN implementations, only the Hadoop-based one is presented in a scientific paper together with its performance evaluations, but the implementation is not freely available. The dataset used for the scalability evaluation is not available. It consists of 1.92 billion GPS coordinates. The implementation scalability has been evaluated on up to 12 nodes, achieving a linear speedup.

The Spark implementation code is freely available, but there are no official performance evaluations. They only claim to have handled datasets of 3 million points.

The work presented in (39) is the latest density-based clustering implementation and does not provide the code. The dataset is publicly available and consists of less than 200.000 entries and 3 dimensions, while the synthetic datasets used for the scalability benchmarks consist of up to 10 million points and 20 dimensions. Its performance was evaluated in a network on up to 20 nodes, but the speedup is less than linear.

Density-based algorithms - Performance evaluation				
Reference	**Evaluation Dataset Size**	**Dataset availability**	**Implementation availability**	**Scalability evaluation and speedup**
MR-DBSCAN: An Efficient Parallel Density-Based Clustering Algorithm Using MapReduce. (26)	1.92 billion transactions with 2 features	No	No	12 nodes with a linear speedup
Spark DBSCAN (38)	3 million points	No	Yes	NA
DBCURE-MR: An efficient density-based clustering algorithm for large data using MapReduce. (39)	10 million transactions with 8 features	No	No	20 nodes, without a linear speedup

Table 9: Density-based distributed algorithm performance evaluation

The hierarchical clustering algorithm introduced in (40) was evaluated on 1.2 TB web logs, but no information on the number of dimensions is provided, and the dataset is not freely available, as

well as the code. It was tested on a cluster of only two nodes, without taking into account speedup evaluations.

The distributed spectral clustering algorithm introduced in (42) has shown to be able to deal with very large datasets: the first is a collection of Reuters' articles (193844 records) and the second consists of more than 2 million pictures. The experimental test cluster consists of 256 machines, where it achieves a linear speedup on the dataset of pictures, and a less-than-linear speedup on the article repository.

Finally, the clustering algorithm presented in (43) was evaluated on public datasets with up to 1.4 billion points of 6 features. The algorithm is not available and it has been tested on a very large cluster (up to 400 machines), without achieving a linear speedup.

Other clustering algorithms - Performance evaluation				
Reference	**Evaluation Dataset Size**	**Dataset availability**	**Implementation availability**	**Scalability evaluation and speedup**
An Efficient Hierarchical Clustering Method for Large Datasets with Map-Reduce. (40)	1.2 TB of weblogs, not clearly specified	No	No	2 nodes
Parallel Hierarchical Affinity propagation (41)	788 transactions with 2 features	Yes	Yes	80 nodes with a linear speedup with the first 40 nodes
Parallel Spectral Clustering in Distributed Systems (42)	2 121 863 pictures	Yes	Yes	256 nodes without a linear speedup
Clustering Very Large Multi-dimensional Datasets with MapReduce (43)	1,4 billion points with 6 features	No	No	400 nodes without a linear speedup

Table 10: Performance evaluation of other families of clustering algorithms

6.1.6 Final considerations and open issues

The large number of available techniques are an evidence of the complexity of the clustering problem: each technique, depending on its nature, is suitable for a certain type of input data or context.

In general, algorithms that are very complex like Spectral Clustering and Affinity Propagation, or very communication-intensive like Hierarchical ones, are very difficult to efficiently parallelize. In addition, the MapReduce paradigm is not suitable to execute iterative processes. The result is that the convenience of a distributed implementation of such clustering families is questionable because of the overhead costs related to the distributed frameworks.

The K-means implementations are probably the most mature, especially the Spark MLLlib one. However, being a very simple approach, it is not suitable for every type of problem (e.g., K-means identifies globular-shaped clusters and its basic implementation does not gracefully handle noisy data). Density-based approaches are well-suited for use cases with a non-flat geometry of the clusters or uneven cluster sizes (29), but represent a more challenging problem in big data environments because of the quadratic space and time complexity.

Finally, the approach presented in (43) is interesting for communication and I/O cost evaluations, which are often ignored in other works. Furthermore, the exploitation of a sampling approach to deal with large amounts of data is promising. This strategy represents a valid alternative for problems like clustering, taking into account the possibility to exploit distributed frameworks for pre-processing and sampling, which better fit the data locality paradigm.

Relating the current state of the art with ONTIC activities, some gaps need to be filled:

- Few algorithm implementations are publicly available and most of them have only proved to scale to millions of records, whereas clustering techniques in ONTIC aims at processing hundreds of millions of records.
- Most datasets in the state-of-the-art consist of very few attributes (i.e., less than 10), whereas clustering techniques in ONTIC it may be required to handle tens of attributes. The dimensionality issue is probably the most challenging to address due to the complexity explosion and to the difficulty to manage high-dimensional datasets also for the non-distributed algorithms, which usually solve the problem by devising highly-specialized solutions.

6.1.7 Other approaches

Besides the most popular clustering families, which have been presented in the previous sections, others have been extended on distributed frameworks: co-clustering, distribution-based, and correlation-based parallel approaches are present in recent literature and represent a promising line of research.

Co-clustering allows to simultaneously group the rows and the columns of a matrix dataset; a distributed approach is presented in (44). Typical clustering can only discover whether groups of objects have similar features; co-clustering can also target rows that have similar properties for a limited subset of features, by automatically identifying the feature subsets.

Distribution-based clustering algorithms, instead, assume that objects belonging to the same cluster should follow the same distribution. In (45), a MapReduce implementation of EM (Expectation Maximization) was used for estimating the parameters that represent conditional probability distributions (CPDs) for a co-occurrence analysis model. Other distribution-based parallel approaches are available as Spark implementations in MLlib, such as Gaussian Mixture Model or Latent Dirichlet Allocation.

Finally, **correlation clustering** is a technique that operates in a scenario where the relationships between the objects are known instead of the actual representation of the objects. For instance, given a graph with signed edges, it tries to partition it into clusters that minimize the sum of disagreement with respect to signs and edges. In (46), the authors introduce an iterative MapReduce implementation of this technique, showing promising results on datasets with tens of millions of nodes.

6.2 Scalable algorithms for supervised classification

Supervised classification is the learning task of modelling a function that best approximates the distribution of the input dataset with respect to a specific outcome, i.e., the class label. Classification algorithms work on labelled input data (i.e., a training set with a specific attribute called class label) and produce an inferred function which can be used for assigning the label to new unlabeled data. These techniques have been widely adopted in network traffic classification: recent literature presents many applications based on distributed frameworks able to manage very large datasets.

In this section, we introduce a variety of classification algorithms that are relevant for the ONTIC context. To this aim, they have been selected upon two criteria:

- Presenting a distributed or parallelizable approach, with special focus on the Hadoop and Spark distributed frameworks;
- Being tested on a "big" corpus, where big means reaching the million over at least a dimension (e.g. number of records or attributes) or many Gigabytes in size, with the corpus being publicly available or at least described in its characteristics.

Most distributed algorithms currently available in the state of the art are based on well-known centralized techniques; hence their innovative contribution typically consists in the parallel distribution of the centralized technique. The evaluation and comparison of the algorithms focuses primarily on the performance and the scalability, as determined by published experiments. Due to the lack of a standard benchmark, experiments addressing these two crucial characteristics are very heterogeneous among different papers. Additional features present in most of the works, which are also relevant in a distributed environment are:

- The number of MapReduce iterations, critical in the Hadoop context.
- The communication costs among nodes, which are often a bottleneck in distributed systems.

For a more quantitative evaluation, we rely on the following performance and scalability measures, selected among the most common across the experimental sections of the papers:

- Accuracy, i.e. the ratio of correctly classified samples with respect to the total samples.
- The size of the largest dataset on which the model training has been completed successfully.

When reported experiments in the papers were repeated on different numbers of machines, we also infer the horizontal scalability, i.e. how the system speeds up when the load is spread over more nodes.

The distribution of a learning algorithm on parallel frameworks like MapReduce generally involves common methodologies, or patterns, often mixed together by the developers to reach the goal of an efficient parallelization. The most popular patterns are the following:

- **Fully-replicated** approaches require the entire dataset to be present on each node.
- **Partitional** approaches initially divide the dataset into fixed subsets (partitions) and do not further shuffle the dataset among nodes during the processing. We can distinguish two subtypes:

  - **disjoint-partitional**: when the partitions are completely disjoint. The algorithms that directly exploit the partition strategy of HDFS, for example, belong to this group;
  - **overlapping-partitional**: when a single row of the dataset can belong to multiple partitions. The algorithms that sample the dataset with replacement, for example, belong to this group.

- **SQM-like** approaches, whose working principle reflects the Statistical Query Model presented in (47): algorithms are written in a summation form of several statistics, which are sufficient to generate the model. According to (47), when this approach can be applied it can reach almost linear speedup. Usually, due to the nature of such models, these algorithms are also disjoint-partitional.
- **Iterative** approaches, where the number of MapReduce iterations over the data depends on reaching a specific condition.

The following subsections group together classification algorithms sharing the same centralized technique: ensemble, support vector machine, and rule-based. Table 11 summarizes the main features of each algorithm. Finally, a performance evaluation is presented and open issues are discussed.

6.2.1 Ensemble algorithms

Ensemble algorithms are a common technique used to generate a single global model from many different simpler sub-models, improving the overall quality by properly combining them.

In PLANET (27), Bayardo et al. from Google present a MapReduce implementation of a regression tree ensemble in the context of prediction of ads behavior. In particular they focus on “bounces”, clicks on ads that are immediately followed by the user returning back to the search engine. The environment is characterized by massive datasets since a wide variety of features are considered for each click.

PLANET exploits regression trees, which are learning tools that differ from classification ones for the labels that they want to predict, which are continuous. Tree models divide the data space into non-overlapping regions. Each non-leaf node defines region boundaries in the data space and is characterized by a predicate on an attribute from the input dataset. A region is defined by the path from the root to a leaf node in the tree: the final leaf value is the prediction over the input record. In the construction phase, splits are evaluated on their impurity reduction. Loosely defined, the impurity at a node is a measure of the dissimilarity in the labels of the training records that are input to the node.

In PLANET, many such regression trees are merged in an ensemble. The algorithm is fully-replicated and highly iterative, and the number of MapReduce iterations is variable; however, it cannot be less than the depth of the trees, and this lower bound can be respected only when the number of machines is greater than the width of the trees. The communication costs are neglected; indeed, the whole dataset is shipped to all the mappers completely.

Another work exploiting ensembles is COMET (48), where the base models are instead Random Forests; being an ensemble technique for trees themselves, the overall Random Forests model is an ensemble of ensembles. The ensemble generator is unique and similar to AdaBoost, a recursive technique that generates each distinct classifier with initial weights tuned to strengthen the predictions for the values misclassified by the previous steps, and finally vote or average to obtain a unique prediction from them all. The algorithm is disjoint-partitional; the resulting models are

then merged with a final Reduce phase in a unique ensemble. The number of MapReduce iterations is thus only one. The communication costs are also really limited; if the dataset is pre-partitioned, the only data exchanged are the sub-models generated at each node.

6.2.2 SVM-based and logistic regression algorithms

Another widely used method that has been successfully parallelized is Support Vector Machines (SVMs), exploited by (49), (50) and (51). In brief, SVMs are a binary classification tool aiming at maximizing the distance, or gap, between the values of the two classes.

(49) classifies emails in categories by applying a cascade of SVMs, which in Hadoop is equivalent to split the dataset in partitions and generate an SVM on each mapper, with the following iterations working on the output of the previous ones. The approach is disjoint-partitional. Being hierarchical in its structure, the number of MapReduce iterations is $\log_2(M)$, where M is the number of mappers. The paper neglects the communication costs, but they are limited to the exchange of the SVM models at each iteration.

(50) ports SVMs to the Hadoop framework by training a separate linear SVM for each mapper and then joining the trained classifiers at the reduce phase, exploiting intrinsic properties of the SVMs. The algorithm is disjoint-partitional. The number of MapReduce iterations is only one. The communication costs are neglected in the paper, but the only data exchanged are the linear SVMs, which are usually very small and manageable.

In (51) both SVM and logistic regression are implemented on Spark by solving their corresponding optimization problem, through a distributed approach potentially suitable for any optimization problem. The algorithm is SQM-like, iterative; the master is in charge of the global synchronization of the current solution at each iteration. The iterative nature of the algorithm is not a big issue in Spark, especially in the case when the dataset fits in memory. The stopping criterion, on which the number of iterations depends, is the amount of improvement of the current solution with respect to the previous one, which has to be above a threshold for the algorithm to continue. The communication costs are a major issue with a high number of dimensions.

6.2.3 Rule-based algorithms

Other attempts to design distributed classification algorithms involve rules, in different fashions: rule-based classifiers (52) generate an ordered set of rules in the form “if...then...”; linguistic fuzzy rule-based classifiers (53) generate rules where the antecedent can be fuzzy and have linguistic values, such as “x is small”. Both approaches are disjoint-partitional and neglect the communication costs. Notably, (53) needs only one MapReduce iteration, while (52) is iterative, with its stopping criterion being the coverage of a percentage of the dataset (specified by the user) with the extracted rules.

	Framework	Baseline Algorithm	Approach type	Implementation availability	Number of iterations	Communication cost impact
PLANET (27)	Hadoop	Regression tree ensembles	Fully-replicated, iterative	no	[depth, 2^{depth}-1]	huge
COMET (48)	Hadoop	Ensemble of Random Forests	Disjoint-partitional	no	1	limited
MapReduce based Parallel SVM for Email Classification (49)	Hadoop	SVMs in cascade	Disjoint-partitional	no	$\log_2(M)$	limited
Distributed confidence-weighted classification on mapreduce (50)	Hadoop	Linear SVM	Disjoint-partitional	no	1	limited
Large-scale Logistic Regression and Linear SVM using Spark (51)	Spark	Logistic regression and SVM	SQM-like, iterative.	yes	while gradient > epsilon	huge on high dimensionality
RuleMR (52)	Hadoop	rule based classifier	iterative	no	while coverage < min	ignored
MapReduce to build linguistic fuzzy rule based classification systems (53)	Hadoop	fuzzy rule based linguistic classifier	Partitional	no	1	ignored

Table 11: Classification algorithm comparison

6.2.4 Performance evaluation

Almost all the distributed versions of the above-mentioned algorithms reach, within a reasonable confidence interval, the accuracy of the centralized version (single-core or single-machine). The only exception to this is (51), which, despite having been evaluated on a collection of datasets heterogeneous in sizes, densities and dimensions, presents only experiments and results addressing its scalability. As for the others, since their experimental sections consider completely disjoint collections of datasets (and, in the case of (49), very different tasks), it is unfortunately impossible to rank them for their performance.

The experiments present training datasets very different in size and shape. (51), as already mentioned, executes the proposed (and, at the time of writing, released to the community) algorithm on a good range of datasets of different characteristics. Also, it is the only work to reach a considerably high dimensionality, with a peak of more than 3 million attributes in the largest dataset, way more than the second in this ranking, COMET (48), which counts 1143 features. All

the other works stay well below the hundred attributes, as can be seen in Table 12, which shows, for each work, the kind and size of the largest dataset of the trials. The largest volume of data was reached by (50), with more than a billion records, and (27), with some three hundred million records, both with a limited number of features (21 and 10, respectively). (53) and (49) rely, for their largest-scale test, on publicly available datasets, the biggest of which counts 5 million records.

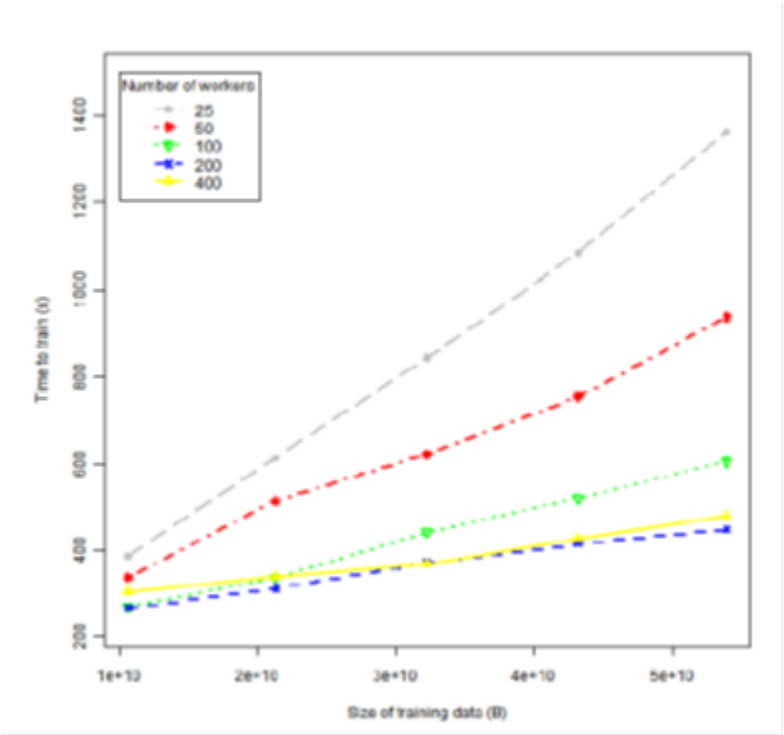

Figure 3: Running time with growing number of machines in PLANET (27)

The evaluation of horizontal scalability is tricky. As already stated in Section 6.1.5 we have to point out that the ideal behavior is linear, until a limit of scalability after which the overhead of the framework outweighs the improvement brought by parallelism. The experimental section should therefore aim to inspect how the real behavior deviates from the ideal one. Unfortunately, the machines at disposal of the researchers often limit such analyses. It is not the case of (27), however, which varies the number of machines from 25 up to 400. Here the speedup seems linear with a slope less than unitary, and the largest-scale experiments reach the point of no improvement, with an almost steady behavior. (50) scales up the mappers from 100 to 10000. Also here the initial phase is linear; but, since the complexity of the reducer grows with the number of mappers, it soon reaches a point after which the performance degrades. In such a scenario, care must be taken to tune the number of mappers, avoiding oversizing the cluster. In (51) the authors vary the number of workers from 2 to 16, and has the merit of performing experiments on datasets of different shapes, showing that it actually weighs on the speedup. On a very dense dataset, indeed, the scale up is almost linear; on the contrary, on a very sparse dataset the performances showed a degradation with the addition of more machines. (53) performs the experiments on 16, 32 and 64 mappers. Even though the speedup is evident and promising (from 16 to 32 mappers it is triple), this datum is not sufficient to infer a general trend or speak about linearity. In (49), experiments do not show the horizontal scalability, but we can infer from the theoretical complexity that it is ideally almost linear, with a slope between ½ and ¼. For the other works studied, no information is available.

6.2.5 Final considerations and open issues

PLANET can be considered a milestone, as it has set the bar for all the works that have followed.

In its approach, all the dataset is delivered to each node: mappers, in fact, are able to understand if a record has to be processed thanks to the model (that is spread too). This is considered a bottleneck not to underestimate, that can be partially mitigated by the point that all the nodes at the same level are processed in a single MapReduce Job, in a breadth-first fashion. Nevertheless, memory limitations and the number of machines prevent the controller from scheduling MapReduce jobs for all the nodes on a queue at once. Evaluations have shown that adding more machines significantly decreases running time until the overhead of distributing the processing (and data) outweights its benefits. We plan to take the lessons learned from PLANET into account while devising the algorithmic solutions for ONTIC, however, both in volume and dimensionality (i.e., PLANET has been experimentally evaluated on a 10-feature dataset), some gaps need to be filled.

Works (50) and (51) have demonstrated to be able to take up the challenge of PLANET both in volume and in dimensionality, but not without any space for criticism. In particular, (51) completely neglects the evaluation of the accuracy. Even though the code is now part of MLlib, thus publicly available, reproducing the experiments is impossible because of the unavailability of the main corpus, owned by Yahoo. (50), instead, scales up the problem only in volume, being their work strictly focused on ad latencies.

The other works mentioned here do not reach the same benchmark size, but it must be recalled that such sizes correspond to datasets not available to the public, owned by Yahoo and Google. A theoretical pitfall of their approaches, however, is its reliance onat they iterative methods, shown especially by (52). MapReduce, indeed, does best when the number of map/reduce iterations is limited, and the dataset has to be scanned few times. If a few iterations are not sufficient to reduce the dataset to a manageable volume (i.e. to fit in memory and, for instance, be processed by a single node), MapReduce is not a viable strategy on any platform. (52) overcomes this problem through user parameters that indirectly stop the iterative process even if the optimum (e.g. a full coverage of the dataset) is not reached.

Besides the volume and dimensionality challenges, which are exquisite scientific issues to be addressed, and the limited scalability due to the iterative nature of most solutions, a practical limit in the current state of the art is the unavailability of the algorithms: only the MLlib-provided classification algorithm is publicly available (Table 11). This opens the way for ONTIC to learn the lessons presented in the state of the art and push the limits of currently public implementations to improve in terms of accuracy and interpretability: the MLlib provides classifiers are currently based on SVM, which is almost a black-box approach, whereas most companies are deeply interested in understanding the classification model created by the algorithms to foster their competitiveness. A promising solution in ONTIC is expected to come from a distributed rule-based algorithm based on the Spark MLlib framework, so that a scientific contribution can be delivered by improving upon existing software solutions.

	Input	Volume (#records)	Dimensions (#attributes)	training:test split
PLANET (27)	numerical	314M	10	90:10
COMET (48)	numerical and categorical	1,4M	1143	70:30
MapReduce based Parallel SVM for Email Classification (49)	emails	200K	X	50:50
Distributed confidence-weighted classification on mapreduce (50)	numerical and categorical	1.3G	21	77:23
Large-scale Logistic Regression and Linear SVM using Spark (51)	numerical and categorical	460554	3052939	NA
RuleMR (52)	categorical	100M	10	NA
MapReduce to build linguistic fuzzy rule based classification systems (53)	numerical and categorical	5749132	12	90:10

Table 12: Largest-size datasets in classification state of the art

6.2.6 Other approaches

Other works in literature deserve attention: one of these is surely Nimble (54). This is not an algorithm but a framework used to address deficiencies of existing parallel programming paradigms, which are considered too low-level and ill-suited for implementing machine learning and data mining algorithms. Nimble is a portable infrastructure that has been specifically designed to enable rapid implementation of parallel algorithms through reusable building blocks that can be efficiently executed using MapReduce.

(55) instead, introduces a MapReduce implementation of three popular classification algorithms: K-Nearest Neighbors, Decision Tree and Naïve Bayesian classifier; finally, (56) implements an optimized decision tree for noisy big data: in some contexts, big data repositories can be very sparse and noisy; in this work, the authors control the growth of the Decision Tree through specific parameters and a new pruning mechanism.

6.3 Scalable algorithms for correlation analysis

Association rule mining is a data analysis method able to discover interesting and hidden correlations among data. It consists of two steps: (i) frequent itemset extraction, in which all the frequent patterns are mined from the input dataset and (ii) association rule generation.

Association rule mining is performed on transactional datasets; where each transaction consists of a set of items (attribute-value pairs). The itemset support is the number of transactions in which it is present. An itemset is defined as frequent if its support is over an input parameter, called minimum support threshold. An itemset is closed if none of its immediate supersets has the same support (29). Association rules are generated from frequent itemsets and highlight correlations among items.

The Frequent Itemset Mining problem in big data is a very challenging problem, since the complete itemset generation is a complex mining task requiring knowledge of the whole dataset.

We focus on distributed approaches based on Hadoop and Spark frameworks, and evaluate both data mining performance and distributed scalability, as referenced in Table 13. The first family of criteria is related to the characteristics of the frequent itemset mining algorithm:

- The **underlying centralized algorithm**: the main contribution of state-of-the-art papers is focused on the parallel implementation and combination of known centralized techniques.
- **Search space exploration strategy** and **data distribution**: this couple of attributes, directly inherited from the underlying approaches, characterizes the candidate itemset generation (57). The performance of the algorithms that adopt a breadth-first exploration is negatively affected by a large average transaction width, because more candidate itemsets must be examined (29). Since average transaction width is strongly related to the input data distribution, there exists a relationship between the exploration strategy and the input dataset distribution. For example, Apriori-based algorithm (58), detailed in Section 6.3.2 , with their breadth-first exploration approach, better fits datasets characterized by sparse distributions, i.e. low correlation among patterns and high item cardinality.

The second set of dimensions is more related to the parallelization of the algorithms, following the same criteria of the previous sections:

- Load balancing of the processing among the nodes.
- Communication costs to exchange intermediate data across the cluster.

Finally, we present a performance evaluation based on the experimental section of each paper, as summarized in Table 14. The following subsections are structured according to the underlying frequent itemset algorithm: **FP-growth based algorithms** and **Apriori** or **Eclat**-based algorithms.

6.3.1 FP-growth based algorithms

FP-growth (59) is among the most popular approaches for frequent pattern mining (FP stands for ‘frequent pattern’). It is based on an FP-tree transposition of the transaction dataset and a recursive “divide and conquer” visit of the FP-tree. **Parallel FP-growth** is a distributed FP-growth implementation which exploits the MapReduce paradigm to parallelize and mine independent FP-trees that are distributed to the cluster nodes. As referenced in Table 13, PFP exploits a deep first exploration of the FP-tree, which is more efficient for dense datasets. The parallel algorithm extracts the k most frequent closed itemsets. Parallel FP-growth (12) is based on the generation

of independent FP-trees that allow achieving work independence among the nodes. However, the independent FP-trees can have different characteristics and this factor has a significant impact on the execution time of the mining tasks. When the FP-Trees have different sizes, the tasks are unbalanced and hence the whole mining process is unbalanced. This problem could be potentially solved by splitting complex trees in sub-trees: however, defining a metric to split a tree is not easy. Finally, the algorithm takes into account communication costs even if the shards of the dataset that are sent to the nodes overlap significantly.

Spark PFP (15) represents a pure transposition of FP-growth to Spark; it is also included in MLlib, the Spark machine learning library. The algorithm implementation in Spark is very close to the Hadoop sibling, i.e., it first builds independent FP-trees and then invokes the mining step on each tree (one independent task for each FP-tree). It is characterized by dynamic and smooth handling of the different stages of the algorithm, without a strict division in phases. Its main advantage over the Hadoop sibling is the low I/O cost, potentially leading to a single read of the dataset from disk, by loading the transactions in an RDD and processing the data in main memory, whereas the Hadoop-based implementation of PFP performs many more I/O operations.

6.3.2 Apriori and Eclat based algorithms

BigFIM and Dist-Eclat (13) are based, instead, on Apriori and Eclat algorithms respectively.

Apriori is a very popular technique: it uses a bottom up approach in which we can distinguish a candidate generation phase (candidates are extended one item at a time) and a counting phase (candidates are tested against the dataset). Eclat is based on equivalence classes (groups of itemsets sharing a common prefix), which are smartly merged to obtain all the candidates. The strategy is close to FP-growth with a depth-first exploration manner. Apriori, instead, expands the candidates in a breadth-first fashion, and proceeds iteratively until no candidates reach the minimum support threshold.

Dist-Eclat exploits the Eclat algorithm to extract a superset of closed itemsets. The algorithm is divided in two phases: the first aims at finding k-sized prefixes on which, in the second phase, the algorithm builds independent subtrees. Dist-Eclat is very fast but, with some prefixes configuration, it assumes that the whole initial dataset (transposed in vertical format) should be stored in the nodes main memory.

As reported in Table 13, from an analytical point of view, Dist-Eclat and BigFIM are the only algorithms for which an evaluation of the communication costs and load balancing is presented in their respective experimental sections. In particular, the choice of the length of the prefixes generated during the first step affects both communication costs and load balancing. The former would benefit from shorter prefixes while the latter would improve with a deeper level of the mining phase before the redistribution of the seeds. Hence, depending on the data distribution and the characteristics of the Hadoop cluster, Dist-Eclat can be tuned to optimize communication costs or load balancing. Dist-Eclat better fits dense datasets: the depth-first search strategy may require more infrequent itemsets generated and tested than, for instance, Apriori does. As a result, Eclat efficiency reduces for sparse data with short patterns where most itemsets are infrequent (60).

The **BigFIM** implementation is very similar to Dist-Eclat. The only difference lies in the prefixes extraction phase, where BigFIM exploits the Apriori algorithm: BigFIM's structure makes it more scalable when dealing with very large datasets (i.e., with more than 6 million transactions and more than 45 million items). Even if it is slower than Dist-Eclat, which is focused on speed, the former is developed to run on really large datasets, where the latter runs out of memory. The

reason is related to the first phase in which, exploiting Apriori strategy, the k-prefixes are extracted in a breadth-first fashion. Consequently, the nodes do not have to keep large transaction lists in memory but only itemsets to be counted. One of the most critical issues of the application of Apriori to large datasets is that at a certain level, the set of candidates cannot be stored in main memory: this is not the case because, for the first few levels, this is still possible. Dist-Eclat, instead, as already said, in the worst case is limited by the need of communicating and storing the whole dataset in each node. Finally, because of the differences in the extraction technique used in each phase, in the first one BigFIM achieves the best performance with sparse datasets, while in the second phase it better fits dense ones: overall it does not show a data-distribution preference.

YAFIM (61) is an Apriori distributed implementation developed in Spark. Apriori works best with sparse datasets and it is characterized by a different behavior than Eclat and FP-growth: the iterative nature of the algorithm has always represented a challenge for its application in MapReduce-based big data frameworks. The reasons are the overhead caused by the launch of new Map Reduce jobs and the requirement to read the input dataset from the disk at each iteration. YAFIM exploits Spark RDDs to cope with these issues. Precisely, it assumes that all the dataset can be loaded into RDDs in order to speed up the counting operations. Hence, after the first phase in which all the transactions are loaded, the algorithm starts the iterative merging and pruning, organizing the candidates in a hash tree to speed up the search. YAFIM exploits the Spark feature "broadcast variables abstraction", which allows programmers to send a piece of shared data to each slave only once, rather than with every job that uses the data. This implementation mitigates communication costs (reducing the inter job communication), while load balancing is not addressed.

Algorithm	Framework	Underlying algorithm	Data distribution	Search Strategy	Communication cost handling	Load balance handling
PFP (12)	Hadoop	FP-growth	dense	Depth-first	Yes	No
Spark PFP (15)	Spark	FP-growth	dense	Depth-first	Yes	No
Dist-Eclat (13)	Hadoop	Eclat	dense	Depth-first	Yes (best effort with load balance)	Yes
BigFIM (13)	Hadoop	Apriori and Eclat	sparse and dense	Breadth-first & Depth-first	Yes (best effort with load balance)	Yes
YAFIM (61)	Spark	Apriori	sparse	Breadth-first	Yes	No

Table **13**: Correlation analysis algorithm comparison

6.3.3 Performance evaluation

The performance evaluation follows the structure reported in Table 14: the size of the datasets and their public availability, the availability of the algorithm code, the number of parameters needed by the algorithm, the cluster dimension, and the speedup are compared among the different approaches. Additionally, we highlight the type of itemsets they are able to extract: closed itemsets (29) are very common because they provide a compact representation of frequent itemsets. Indeed, most algorithms extract the closed itemsets: YAFIM is the only one which extracts all the frequent itemsets.

The Spark PFP implementation provided in MLlib lacks any performance evaluation, however all datasets are publicly available. BigFIM represents the most scalable solution when dealing with

very large datasets (more than 6 million transactions and 45 million items). Dist-Eclat, evaluated on the same datasets, cannot be considered as scalable because of the issue evidenced in section 6.3.2 .

The Hadoop PFP is evaluated on an even larger dataset, but it does not achieve the same performance as Dist-Eclat or BigFIM, as shown in Figure 4. Finally, YAFIM experimental section uses the smallest datasets, with less than one million transactions.

All the Hadoop implementations have been evaluated with a quite high number of mappers and/or reducers. As shown in Table 14, PFP has been evaluated with a number of machines up to 2500 (the authors work at Google), and showed the best behavior, achieving an almost linear speedup. BigFIM and Dist-Eclat exploited up to 80 nodes, and reached a performance limit due to the size of some subtrees blowing up. YAFIM's behavior has been analyzed with a benchmark of over 12 machines that resulted in a near-linear speedup.

Almost all the approaches provide a public implementation: only YAFIM code is not published.

All techniques require the MinSup (Minimum Support) input parameter. BigFIM and Dist-Eclat require additional input, since they work with a customizable length of first-phase prefixes, and this feature could require some experiments to find the best value, depending on the kind of datasets and the cluster configuration. Tuning the prefix length allows BigFIM and Dist-Eclat to handle both communication costs and load balancing.

Parameter-free itemset mining is a quite hot topic and the most interesting algorithm is introduced in (62). The paper presents a parameter-free distributed (not MapReduce) approach: the main idea is to efficiently explore the search space of the candidates in a different manner, exploiting several interestingness measures, collecting data into highly similar subsets or partitions. This partition strategy allows the nodes to work independently from each other on small shards of the dataset.

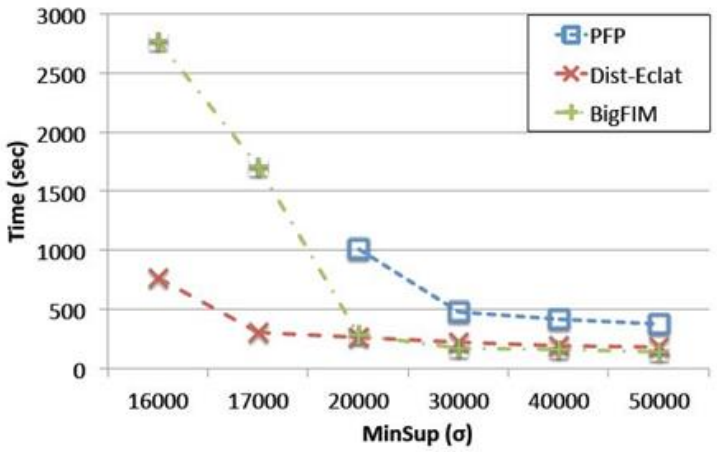

Figure 4: Dist-Eclat and BigFIM performance compared to PFP (13).

6.3.4 Final considerations and open issues

Even if PFP has been considered the reference technique for a long time, the more recent Spark implementations seem to be very promising.

Unfortunately, some of the frameworks work with the assumption that the dataset can fit in memory. This assumption is not always realistic in a big data environment. This is the case for YAFIM: however, we consider it very interesting because of its direct comparison with the Hadoop version to evidence the key advantage delivered by RDD utilization. Dist-Eclat outperforms PFP, but in the worst case assumes that the whole dataset can be sent to all the nodes, especially when trying to achieve a good load balancing: this can be critical with very large datasets. BigFIM proved to be slower than Dist-Eclat but able to process larger datasets when Dist-Eclat runs out of memory.

For future perspective and developments, we consider Spark implementations more promising than Hadoop ones, even if the former currently appears as less mature. Spark algorithms are very recent in literature and we expect to find more complete implementations very soon.

Algorithm	Evaluation Dataset Size	Dataset availability	Implementation availability	SpeedUp	MinSup	Itemsets
PFP (12)	15,898,949 rows and total items 84,925,908*	Yes	Yes	Almost linear speedup - up to 2500 nodes	Yes	Closed
Spark PFP (15)	NA	Yes	Yes	NA	Yes	Closed
Dist-Eclat (13)	6,201,207 rows and 45,446,863	Yes	Yes	Almost Linear speedup until a threshold – up to 80 nodes	Yes	Closed
BigFIM (13)	6,201,207 rows and 45,446,863 different items (lower minimum support than Dist-Eclat)	Yes	Yes	Almost Linear speedup until a threshold – up to 80 nodes	Yes	Closed
YAFIM (61)	100,000 rows and 870 different items or 49,046 rows and 2,113 different items	Yes	NO	Almost linear speedup - up to 12 nodes	Yes	Frequent

Table 14: Performance evaluation of correlation analysis algorithms

With respect to the ONTIC activities, we are glad to see many publicly available implementations (Table 14), but the currently fastest solutions (Dist-Eclat and BigFIM) have proved to scale only to 6 million transactions and up to 80 nodes. Within ONTIC we aim at pushing this limit in both directions: the ONTIC dataset size is expected to be much larger, and the availability of a huge enterprise-level cluster in EMC, an ONTIC partner, will allow us to scale much further. Furthermore, ONTIC may require to address also the dimensionality issue in terms of rule/itemset length: this opens the way for innovative scientific contributions as well as positive impacts on the industry, since long itemsets are very difficult to extract but they model complex and potentially interesting interactions (such as longer network patterns and multi-dimensional behaviors).

6.3.5 Other approaches

The state-of-the-art literature is rich in sampling methodologies for correlation discovery and frequent itemset mining. Frequent itemset mining and association rules discovery are processes which rely on finite results, strictly related to a minimum support threshold. However, in some contexts, high quality approximated results are useful anyway, especially if they represent a resource easy and fast to obtain. In (63), (64), approximated frequent itemsets are extracted from a subset of the whole dataset. The authors guarantee the quality of the results through the usage of analytical and probabilistic models. (65), instead, proposes an approximate frequent itemset mining method that uses k-medoids to cluster transactions. Clusters are then analyzed and their representative transactions are used as candidate itemsets whose real support is then obtained from the input dataset. Finally, (66) introduced a distributed implementation of the Partition algorithm (67). Partition is a frequent itemset mining algorithm that consists of splitting the database in shards and finding local frequent itemsets, sequentially in each of them. Then, local frequent itemsets are merged in order to obtain global frequent candidates which will be tested, finally, against the whole dataset. The distributed implementation of Partition is very straightforward, with the candidate generation and merging done in parallel in a single MapReduce job.

7. Data mining for Network Traffic Characterization

Network traffic characterization serves as the foundation of different network activities since it makes fundamental contribution to advanced network management and security control, including security monitoring, intrusion detection, Quality of Service or Experience and lawful interception. Among the many research laboratories addressing network traffic characterization, NSCLab is an Australian laboratory in Deakin University whose scientific publications can provide us useful insights into this research field (68). Classic port-based and payload-based traffic characterization approaches are facing greater restrictions since more constraints such as unreliable port number and stricter privacy policies limit their performance. Alternative approaches that are based on statistical characteristics applying machine learning techniques were proposed in recent research works (69), (70). These approaches can be divided into two categories according to the specific ML techniques:

- supervised methods aim at finding a mapping from network data features to output classes using labelled training data;
- unsupervised methods, instead, aim to discover clusters in unlabeled data within which network traffic flows or packets that have similar characteristics are grouped into the same cluster without any prior knowledge of the input data.

In the domain of network traffic characterization, supervised methods often achieve better performances, but since the labelling of input data is time consuming and unrealistic in a big data scenario, unsupervised methods also have their own advantages.

The state of the art presented in this section helps in connecting the general data mining approaches described in Section 6 with the ONTIC applicative domain. Furthermore, they represent the baseline on which new solutions can be devised, by improving scalability, performance and efficiency, and by adapting them to the specific ONTIC use cases (see Deliverable D5.1).

7.1 Supervised methods

In (71) a thorough and detailed survey is presented introducing and comparing different machine learning techniques for Internet traffic classification. Eighteen state-of-the-art research works are categorized according to different ML strategies. These representative researches are analyzed on the basis of a few key requirements they proposed as vital in the employment of ML-based traffic classifiers in operational IP networks, such as:

- timely and continuous classification (using only a few packets of each flow)
- directional neutrality (recognize an interest without relying on directionality)
- efficient use of memory and processors
- portability and robustness.

The work presented in (72) provides a performance comparison of five machine learning techniques, namely Bayesian Network, C4.5 Decision Tree, Naïve Bayes Discretization (NBD) density estimation, Naïve Bayes Kernel (NBK) density estimation, and Naïve Bayes Tree for IP traffic classification, using both the original feature set and the smaller subsets generated by

Consistency-based and Correlation-based feature selection methods. The experimental results showed the impact of feature selection on the performance of different ML techniques. They also encouraged the comparison of computational performance as an evaluation metric besides classification accuracy: even though the classification accuracy for different techniques might be similar, their computational performance may be remarkably distinguishable.

In (73) the authors propose a model named "bag of flows" (BoF) to reflect the correlation information widely existing in network traffic flows. By their definition, BoF is a combination that consists of some correlated traffic flows which are generated by the same application. Consequently, the BoF model-based traffic classification approach aggregates the individual classification results for each instance inside the BoF produced by the Nearest Neighbor classifier to generate the final label for the whole BoF. Three different strategies were proposed: AVG-NN calculates the average distance of multiple flows to make the unique decision for a BoF; MIN-NN uses the minimum flow distance to decide the class label; MVT-NN combines multiple decisions and follows the majority vote rule to generate final result for a BoF. Their classification results of BoF model-based approach outperformed that of an original k-NN algorithm. In (74) a similar approach was proposed by aggregating the prediction results of naïve Bayes method for a BoF.

7.2 Unsupervised methods

Supervised methods aim to classify input data into given classes, which is very difficult when unknown applications that do not belong to any of the predefined classes are introduced; for unsupervised methods, instead, class labels are not necessary to perform the analysis.

In (75) the authors propose a novel clustering method able to obtain application-level clusters making use of both flow statistical characteristics and IP packet payload in the training stage, while avoiding the use of payload information in the test stage considering the privacy policy. First, many clusters of high purity are generated by clustering based on flow statistical features; then similar clusters are aggregated by inspecting their payload content; and a new classifier is established on the basis of the flow statistical features of the aggregation results for the testing stage. They also proposed to describe the content of traffic clusters utilizing a bag-of words (BoW) model and comparing the similarity among clusters using latent semantic analysis (LSA).

In (76) a new approach performing clustering using Random Forest proximities instead of Euclidean distance was proposed. First, a pairwise proximity matrix was derived by means of Random Forest classification on the original data and a set of synthetic data. Then, a K-Medoids clustering procedure was performed to generate the final K clusters using the proximity matrix.

7.3 Semi-Supervised methods

In the domain of network traffic characterization, a lot of background information can assist to obtain better performance in the clustering process.

In (77) the authors make the assumption that in certain time intervals, when flows share the same destination IP address, destination port and protocol, these flows are believed to use the same application layer protocol. If this kind of correlation information exists, they propose two pairwise constraints called Must-link (two instances must be in the same cluster) and Cannot-link (two instances cannot be in the same cluster). Three variants of the classic K-means clustering algorithm are then adopted in order to incorporate the defined constraints: (i) COPK-MEANS intends to directly satisfy all established constraints; (ii) MPCK-MEANS sets up a penalty cost for violating constraints, and seeks for the minimum of a modified objective function considering the

distances between data instances and cluster centers, and the penalty cost; (iii) LCVQE modifies the objective function to penalize the violation of any constraint by directly calculating distances.

Their experimental results showed that the performance of internet traffic clustering, measured by the ratio of true positives and the total number of flows, can be significantly improved: they reach a 10% higher accuracy than K-means.

In (78) the authors propose a Set-Based Constrained K-means algorithm (SBCK-Means) that is similar to COPK-Means, and experimental results comparing SBCK-Means with classic K-means are presented: the novel algorithm achieves a 20% better accuracy.

In (79) a different semi-supervised method is proposed: flow correlation information is introduced in both training and testing stages. In the training stage, flow correlation is used to automatically assign labels to previously unlabeled instances based on their correlation to a (small) set of pre-labeled instances. In the testing stage, strongly correlated flows, which are modelled by a bag-of-flows (BoF), are classified as a combination considering their individual classification results. The final compound classification result for the whole BoF follows the majority vote rule.

8. Conclusions

The deliverable contribution is three-fold and consists of:

- A state-of-the-practice overview of big data platforms for data mining (Section 5).
- A scientific review of the state of the art in big data analysis techniques (Section 6).Existing applications of analysis algorithms for network traffic characterization (Section 7).

The deliverable reports the available offline algorithms identified as state of the art in the context of the ONTIC project. A broad and thorough comparison is presented, taking into account specific features of each family of algorithms and limitations in terms of dataset availability, code availability, implementation scalability, and performance. The comprehensive overview of scalable techniques addresses the main tasks of WP3: clustering, classification and correlation analysis. The document highlights current boundaries of the state of the art, in terms of volume and features.

Relating the current state of the art with ONTIC activities, some gaps need to be filled:

- Most of the current public algorithm implementations have only proved to scale to millions of records, whereas ONTIC aims at processing hundreds of millions of records. Addressing this issue not only requires a natively scalable approach, but also entails the evaluation of smart sampling techniques that can cut the processing time and resource requirements, so that experiments are accessible also to distributed computing environments available in universities, and results are produced in a reasonable time frame to allow the iterative design and development process of academic scientific research.
- Most datasets in the state-of-the-art consists of very few attributes (i.e., less than 10), whereas ONTIC may require to handle tens to hundreds of attributes, to model complex and potentially interesting interactions (such as longer network patterns and multi-dimensional behaviors). The dimensionality issue is probably the most challenging to address due to the complexity explosion and to the difficulty to manage high-dimensionality datasets also for the non-distributed algorithms. This issue is currently solved by non-distributed algorithms with highly-specialized approaches.
- Apart from the volume and variety (dimensionality) challenges, which are exquisitely scientific issues to be addressed, and the limited scalability due to the iterativeness of most solutions, a practical limit in the current state of the art is the unavailability of the algorithms. This opens the way for ONTIC to learn the lessons presented in the state of the art and push the limits of currently public implementations to improve in terms of performance (e.g., accuracy) and model interpretability: most companies are deeply interested in understanding the analysis models created by the algorithms to foster their competitiveness.

The state of the art presented in Section 7 helps in connecting the general data mining approaches described in Section 6 with the ONTIC applicative domain. Furthermore, they represent the baseline on which new solutions can be devised, by improvement in scalability, performance and efficiency, and by adaptation to the specific ONTIC use cases (see Deliverable D5.1).

Finally, a state-of-the-practice section describes the most popular frameworks and software solutions relevant in the ONTIC context. The current state-of-the-practice in distributed frameworks is based on Apache Hadoop and Apache Spark. The former is based on the MapReduce paradigm: a Map procedure performs processing and filtering on input data, and a Reduce

procedure applies an aggregating function. Hadoop MapReduce approach has been widely adopted in recent years; however the latest trend is rather exploiting Spark, which is more suitable for iterative tasks and algorithms accessing data more than once.

The Apache Hadoop platform, together with its extensions, such as Apache Spark, being the current de-facto standard in the big data environment, is currently considered the most promising foundation to build an offline characterization framework for the ONTIC project.

9. References

1. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. *19th ACM Symposium on Operating Systems Principles 2003*. 2003.
2. Ghemawat, Jeffrey Dean and Sanjay. MapReduce: simplified data processing on large clusters. *Commun. ACM 51, 1 (January 2008)*. 2008.
3. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *OSDI'06: Seventh Symposium on Operating System Design and Implementation*. 2006.
4. Apache HBASE. *http://hbase.apache.org/*.
5. Apache Cassandra. *http://cassandra.apache.org/*.
6. MongoDB. *https://www.mongodb.org/*.
7. Amazon SimpleDB. *http://aws.amazon.com/it/simpledb/*.
8. MPI - Message Passing Interface. *http://it.wikipedia.org/wiki/Message_Passing_Interface*.
9. Open MPI: Open Source High Performance Computing. *http://www.open-mpi.org/*.
10. The Apache Mahout Project, . Mahout Machine Learning Library. *http://mahout.apache.org/*.
11. H2O. *http://0xdata.com/product/*.
12. Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. Pfp: parallel fp-growth for query recommendation. *Proceedings of the 2008 ACM conference on Recommender systems*. 2008.
13. Moens, S., Aksehirli, E. and Goethals, B. Frequent Itemset Mining for Big Data. *Big Data, 2013 IEEE International Conference on*. 2013.
14. Powered by Mahout. *http://mahout.apache.org/general/powered-by-mahout.html*.
15. The Apache Spark Project. MLLIB Machine Learning Library. *https://spark.apache.org/mllib/*.
16. Apache Mahout, aggregated statistics on the git repository. *https://github.com/apache/mahout/pulse/monthly*.
17. Apache Spark, aggregated statistics on the git repository. *https://github.com/apache/spark/pulse/monthly*.
18. Powered by Spark MLlib. *https://cwiki.apache.org/confluence/display/SPARK/Powered+By+Spark*.
19. Databricks blog - MLlib . *https://databricks.com/blog/category/coding/mllib*.
20. Weka 3: Data Mining Software in Java. *http://www.cs.waikato.ac.nz/ml/weka/*.
21. MOA (MASSIVE ONLINE ANALYSIS). *http://moa.cms.waikato.ac.nz/*.
22. MADlib: Big Data Machine Learning in SQL. *http://madlib.net/*.
23. Tstat Homepage. *http://tstat.tlc.polito.it*.
24. Cisco Netflow. *http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html*.
25. Apache Hadoop. *http://hadoop.apache.org*.
26. He, Yaobin, et al. MR-DBSCAN: An Efficient Parallel Density-Based Clustering Algorithm Using MapReduce. *Parallel and Distributed Systems (ICPADS), 2011 IEEE*. 2011.
27. *Planet: massively parallel learning of tree ensembles with mapreduce*. Panda, Biswanath, et al. 2, s.l. : VLDB Endowment, 2009, Proceedings of the VLDB Endowment, Vol. 2, pp. 1426-1437.
28. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12)*.
29. Pang-Ning Tan, Michael Steinbach, Vipin Kumar. *Introduction to Data Mining*. s.l. : Addison-Wesley Companion Book Site, 2006.

30. Foto N. Afrati, Magdalena Balazinska, Anish Das Sarma, Bill Howe, Semih Salihoglu, and Jeffrey D. Ullman. Designing good algorithms for MapReduce and beyond. *In Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12)*. 2012.
31. Jure Leskovec, Anand Rajaraman, Jeff Ullman. *Mining of Massive Datasets*. s.l. : Cambridge University Press.
32. Anish Das Sarma, Foto N. Afrati, Semih Salihoglu, and Jeffrey D. Ullman. Upper and lower bounds on the cost of a map-reduce computation. *In Proceedings of the 39th international conference on Very Large Data Bases (PVLDB'13)*. 2013.
33. Andrew McCallum, Kamal Nigam, Lyle H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. *KDD 2000*.
34. Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. Scalable k-means++. *Proc. VLDB Endow. 5, 7 (March 2012)*. 2012.
35. Thibault Debatty, Pietro Michiardi, Olivier Thonnard, Wim Mees. Determining the k in k-means with MapReduce. *In Proc. of BeyondMR 2014*. 2014.
36. Alina Ene, Sungjin Im, and Benjamin Moseley. Fast clustering using MapReduce. *KDD '11*. 2011.
37. Xu, Martin Ester and Hans-peter Kriegel and Jörg Sander and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *KDD'96*. 1996.
38. Spark DBScan. *https://github.com/alitouka/spark_dbscan*.
39. Younghoon Kim, Kyuseok Shim, Min-Soeng Kim, and June Sup Lee. DBCURE-MR: An efficient density-based clustering algorithm for large data using MapReduce. *Inf. Syst. 42 (June 2014), 15-35*. 2014.
40. Sun, Tianyang, et al. An Efficient Hierarchical Clustering Method for Large Datasets with Map-Reduce. *Parallel and Distributed Computing, Applications and Technologies, 2009 International Conference, IEEE*.
41. Dillon Mark Rose, Jean Michel Rouly, Rana Haber, Nenad Mijatovic, and Adrian M. Peter. Parallel Hierarchical Affinity Propagation with MapReduce. *Proceedings of the 2014 IEEE International Conference on Cloud Engineering (IC2E '14)*.
42. Chen, Wen-Yen, et al. Parallel Spectral Clustering in Distributed Systems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.33, no.3, pp.568,586*. 2011.
43. Robson Leonardo Ferreira Cordeiro, Caetano Traina, Junior, Agma Juci Machado Traina, Julio López, U. Kang, and Christos Faloutsos. Clustering very large multi-dimensional datasets with MapReduce. *KDD'11*. 2011.
44. Sun, Spiros Papadimitriou and Jimeng. DisCo: Distributed Co-clustering with Map-Reduce: A Case Study towards Petabyte-Scale End-to-End Mining. *ICDM '08*. 2008.
45. Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. *Proceedings of the 16th international conference on World Wide Web (WWW '07)*. 2007.
46. Flavio Chierichetti, Nilesh Dalvi, and Ravi Kumar. Correlation clustering in MapReduce. *KDD '14*. 2014.
47. *Map-reduce for machine learning on multicore*. Chu, Cheng and Kim, Sang Kyun and Lin, Yi-An, et al. s.l. : MIT; 1998, 2007, Advances in neural information processing systems, Vol. 19, p. 281.
48. *COMET: A recipe for learning and using large ensembles on massive data*. Basilico, Justin D, et al. 2011. Data mining (ICDM), 2011 IEEE 11th international conference on. pp. 41-50.
49. *A MapReduce based Parallel SVM for Email Classification*. Xu, Ke, et al. 6, 2014, Journal of Networks, Vol. 9, pp. 1640-1647.
50. *Distributed confidence-weighted classification on mapreduce*. Djuric, Nemanja, et al. 2013. Big Data, 2013 IEEE International Conference on. pp. 458-466.
51. *Large-scale logistic regression and linear support vector machines using Spark*. Lin, Chieh-Yen, et al. 2014. Big Data (Big Data), 2014 IEEE International Conference on. pp. 519-528.

52. *RuleMR: Classification rule discovery with MapReduce*. Kolias, Vasilis, et al. 2014. Big Data (Big Data), 2014 IEEE International Conference on. pp. 20-28.
53. *On the use of MapReduce to build linguistic fuzzy rule based classification systems for big data*. López, Victoria, et al. 2014. Fuzzy Systems (FUZZ-IEEE), 2014 IEEE International Conference on. pp. 1905-1912.
54. *NIMBLE: a toolkit for the implementation of parallel data mining and machine learning algorithms on mapreduce*. Ghoting, Amol, et al. 2011. Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 334-342.
55. He, Qing, et al. Parallel implementation of classification algorithms based on MapReduce. s.l. : Springer, 2010, pp. 655-662.
56. *Incrementally optimized decision tree for noisy big data*. Yang, Hang, Fong, Simon and ACM. 2012. Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications. pp. 36-44.
57. Goethals, Bart. Survey on frequent pattern mining. *Univ. of Helsinki*. 2003.
58. Srikant, Rakesh Agrawal and Ramakrishnan. Fast Algorithms for Mining Association Rules in Large Databases. *VLDB'94*. 1994.
59. Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. *SIGMOD '00*. 2000.
60. Lan Vu, Gita Alaghband. Mining Frequent Patterns Based on Data Characteristics. *Proc. of the 2012 Int. Conf. on Information and Knowledge Engineering. 2012*. 2012.
61. Hongjian Qiu, Rong Gu, Chunfeng Yuan, and Yihua Huang. YAFIM: A Parallel Frequent Itemset Mining Algorithm with Spark. *Proceedings of the 2014 IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPSW '14)*. 2014.
62. Gregory Buehrer, Roberto L. de Oliveira Jr., David Fuhry, Srinivasan Parthasarathy. Towards a Parameter-Free and Parallel Itemset Mining Algorithm in Linearithmic Time. *ICDE 2015*. 2015.
63. Matteo Riondato, Justin A. DeBrabant, Rodrigo Fonseca, and Eli Upfal. PARMA: a parallel randomized algorithm for approximate association rules mining in MapReduce. *roceedings of the 21st ACM international conference on Information and knowledge management (CIKM '12)*. 2012.
64. Upfal, Matteo Riondato and Eli. Efficient Discovery of Association Rules and Frequent Itemsets through Sampling with Tight Performance Guarantees. *ACM Trans. Knowl. Discov. Data 8, 4, Article 20 (August 2014)*. 2014.
65. Malek, Maria and Kadima, Hubert. Searching Frequent Itemsets by Clustering Data: Towards a Parallel Approach Using Mapreduce. *WISE 2011*. 2011.
66. J. Da-wei, S. Jie, L. Bin, Z. Cheng. An Efficient Algorithm Based on MapReduce for Computing Frequent Itemsets. *ICCSEE 2013*. 2013.
67. Ashoka Savasere, Edward Omiecinski, and Shamkant B. Navathe. An Efficient Algorithm for Mining Association Rules in Large Databases. *VLDB '95*. 1995.
68. NSCLab at Deakin University. *http://anss.org.au/nsclab/publication.html*.
69. Paxson, V. Empirically derived analytic models of wide-area TCP connections. *IEEE/ACM Trans. Networking, vol. 2, no. 4, pp. 316-336, 1994*. 1994.
70. C. Dewes, A. Wichmann, and A. Feldmann,. An analysis of Internet chat systems. *ACM/SIGCOMM Internet Measurement Conference 2003*. 2003.
71. T. T. Nguyen, G. Armitage. A Survey of Techniques for Internet Traffic Classification using Machine Learning. *IEEE COMMUNICATIONS SURVEYS and TUTORIALS, 2008*. 2008.
72. N. Williams, S. Zander, G. Armitage. A Preliminary Performance Comparison of Five Machine Learning Algorithms for Practical IP Traffic Flow Classification. *ACM SIGCOMM 2006*. 2006.
73. J. Zhang, Y. xiang, Y. Wang, W. Zhou, Y. Xiang, Y. Guan. Network Traffic Classification Using Correlation Information. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 2013*. 2013.
74. J. Zhang, C. Chen, Y. Xiang, W. Zhou, Y. Xiang. Internet Traffic Classification by Aggregating Correlated Naïve Bayes Predictions. *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY*. 2013.

75. J. Zhang, Y. Xiang, W. Zhou, Y. Wang. Unsupervised traffic classification using flow statistical properties and IP packet payload. *Journal of Computer and System Sciences, 2013*. 2013.
76. Y. Wang, Y. Xiang, J. Zhang. Network Traffic Clustering Using Random Forest Proximities. . *IEEE ICC 2013*. 2013.
77. Y. Wang, Y. Xiang, J. Zhang, S. Yu. A Novel Semi-Supervised Approach for Network Traffic Clustering . *NSS 2011*.
78. Wang, Xiang, Zhang. Internet Traffic Clustering with Constraints. *IWCMC 2012*. Y. Wang, Y. Xiang, J. Zhang, S. Yu.
79. J. Zhang, C. Chen, Y. Xiang, W. Zhou. Semi-Supervised and Compound Classification of Network Traffic. *International Journal of Security and Networks, 2013*. 2013.