Horizon 2020 Program (2014-2020)

Big data PPP

Research addressing the main technology challenges of the data economy

Industrial-Driven Big Data as a Self-Service Solution

D3.2: Batch Processing Analytics module implementation as part of I-BiDaaS solution[1†]

Abstract: This deliverable reports on the design and approach of the I-BiDaaS batch-processing module. This report describes the state of the work done in work package 3 from month 12th until month 18th. During these 6 months, we have advanced in the development of both the batch-processing module and the definition and development of the use cases. As part of the batch-processing module we report advances in the Advance Machine Learning submodule, describing our implementation of the Alternating Direction Method of Multipliers optimization, and advances in the technological components of the platform (Hecuba and the Test Data Fabrication tool), that we are leveraging to support the use cases of the project. We also started with the exploration of how our technological component Qbeast can improve the performance of the machine learning algorithms.

Contractual Date of Delivery	30/6/2019
Actual Date of Delivery	30/6/2019
Deliverable Security Class	Public
Editor	*Yolanda Becerra (BSC)*
Contributors	BSC, IBM, UNSPMF, FORTH
Quality Assurance	*Omer Boehm (IBM)* *Dusan Jakovetic (UNSPMF)* *Kostas Lampropoulos (FORTH)*

[1†] *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780787.*

The *I-BiDaaS* Consortium

Foundation for Research and Technology – Hellas (FORTH)	Coordinator	Greece
Barcelona Supercomputing Center (BSC)	Principal Contractor	Spain
IBM Israel – Science and Technology LTD (IBM)	Principal Contractor	Israel
Centro Ricerche FIAT (FCA/CRF)	Principal Contractor	Italy
Software AG (SAG)	Principal Contractor	Germany
Caixabank S.A. (CAIXA)	Principal Contractor	Spain
University of Manchester (UNIMAN)	Principal Contractor	United Kingdom
Ecole Nationale des Ponts et Chaussees (ENPC)	Principal Contractor	France
ATOS Spain S.A. (ATOS)	Principal Contractor	Spain
Aegis IT Research LTD (AEGIS)	Principal Contractor	United Kingdom
Information Technology for Market Leadership (ITML)	Principal Contractor	Greece
University of Novi Sad Faculty of Sciences (UNSPMF)	Principal Contractor	Serbia
Telefonica Investigation y Desarrollo S.A. (TID)	Principal Contractor	Spain

Document Revisions & Quality Assurance

Internal Reviewers

1. *Omer Boehm (IBM)*
2. *Dusan Jakovetic (UNSPMF)*
3. *Kostas Lampropoulos (FORTH)*

Revisions

Version	Date	By	Overview
0.0.1	25/4/2019	Yolanda Becerra	Initial version of the table of contents
0.0.2	30/4/2019	Dusan Jakovetic	Review table of contents
0.0.3	16/5/2019	Yolanda Becerra	Final version of the table of contents
0.0.4	27/5/2019	Yolanda Becerra	Added contributions from BSC
0.0.5	27/5/2019	Dusan Jakovetic	Added contributions from UNSPMF
0.0.6	27/5/2019	Omer Boehm	Added contributions from IBM
0.0.7	28/5/2019	Giorgos Vasiliadis	Added contributions from FORTH
0.0.8	29/5/2019	Yolanda Becerra	Merge contributions from all partners
1.0	14/6/2019	Yolanda Becerra	Update with the internal revisions
2.0	25/6/2019	Yolanda Becerra	Second review

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ADMM: Alternating Directions Method of Multipliers

CC: Call Center

CEP: Complex Event Processing

COMPSs: COMP Superscalar

CQL: Cassandra Query Language

CSI: Customer Satisfaction Index

CSP: Constraint Satisfaction Problem

DB: Database

KPI: Key Performance Index

ML: Machine Learning

MPI: Message Passing Interface

MVP: Minimum Viable Product

PCA: Principal Component Analysis

TDF: Test Data Fabrication

t-SNE: t-Distributed Stochastic Neighbour Embedding

UM: Universal Messaging

Executive summary

This report describes the advances in the work package 3 (WP3) of the I-BiDaaS project: Batch processing innovative technologies for rapidly increasing historical data. The work described in this report summarises the tasks developed from month 12th until month 18th.

This work package focuses on the software stack required for the batch module of the I-BiDaaS platform. We can divide the work done by now into two parts: advances in the machine learning algorithms that the I-BiDaaS platform will provide to users, and advances in the technological components that are driven by the requirements of the use cases of the project. In this report, we also include the description of the use cases that we are considering within this project up to month 18, as well as considerations and initial results on batch analytics of these use cases.

The advances in the machine learning algorithms are oriented to provide the I-BiDaaS platform with the implementation of new machine learning algorithms that the platform will offer to the users. These algorithms are implemented using the COMPSs programming model and in new stages will be adapted to use the Hecuba set of tools.

The advances in the technological components are related to most of the components of the Batch module of the I-BiDaaS platform. Namely, we report advances in Hecuba and the Test Data Fabrication tool. Also, we have performed some experiments that show the benefits of using Qbeast to improve the performance of some machine learning algorithms. We also introduce dislib, a library of machine learning algorithms implemented using COMPSs that will be added to the I-BiDaaS platform. Also, the newly developed COMPSs implementations within I-BiDaaS will be considered to be added to dislib. Finally, we also describe the work done to provide automatic deployment tools that facilitate the installation of all the software components of the I-BiDaaS platform.

Regarding the use cases, in this report, we briefly describe the use cases that we are considering in the project, paying special attention to the use cases that involve the batch module of the I-BiDaaS platform.

1 Introduction

1.1 Overview and contributions to the project goals

The goal of WP3, Batch processing innovative technologies for rapidly increasing historical data, is to develop the I-BiDaaS batch processing module. This module is a software component that is responsible for machine learning and batch analytics within the I-BiDaaS platform. In this report, we describe the work done by the tasks of this work package from the month 12th to the month 18th month of the project. This work package is organized into four tasks:

- Task 3.1: Advancing data analytics via structured (non) convex optimization
- Task 3.2: Innovative Distributed solvers library implementation with COMPSs programming framework
- Task 3.3: Data management: Hecuba
- Task 3.4: Hecuba interaction with TDF

Following, we identify which part of the reported work by this deliverable has been performed by each task and how this work contributes to the final goals of the project:

- Advances in the Machine Learning algorithms that the I-BiDaaS platform will provide to users. This work contributes to the final goals of the project by adding new algorithms, both theoretically (Task 3.1) and in terms of new algorithmic PyCOMPSs [1] implementations (Task 3.2), to the pool provided by the I-BiDaaS platform. In addition, we have analysed the feasibility of including dislib [7], a library that implements a set of machine learning algorithms in COMPSs, as part of the I-BiDaaS platform.
- Advances in the Hecuba tool [8]. This work has been developed as part of task 3.3. This work contributes to the final goals of the project by adding to Hecuba the functionality required by the use cases of the project.
- Advances in the Test Data Fabrication (TDF) [13][14], as well as in its integration with Hecuba, is part of the work developed by task 3.4. The contribution to the final goals of the project is the definition of the data fabrication tools to generate synthetic data based on real data and the integration with the data management component of the I-BiDaaS platform.

1.2 Structure of the report

This report is organized as follows. First, to give context to the work package advances, we start the report with a description of the architecture of the I-BiDaaS platform (see section 2), describing how the batch module positions within the platform.

Following, in section 3, we describe the use cases of the project because the advances of both the machine learning algorithms and the technological components are driven to a large extent by their requirements. Next, we report the advances made within the project from month 12th until month 18th (see section 4). We have considered two different threads: one is advances in the machine learning algorithms (see section 4.1) and the other one is advances in the technological components of the platform (see section 4.2). Finally, in section 5, we present the conclusions of the work done in this period of time and the next steps that we plan to perform.

2 The role of the batch module in I-BiDaaS architecture

As initially reported in the I-BiDaaS deliverable D3.1, the I-BiDaaS batch processing module is a software component that is responsible for machine learning and batch analytics within the I-BiDaaS platform. It contains two sub-modules: 1) COMPSs (COMP Superscalar) programming model [2], and 2) Advanced Machine Learning. The former is a task-based programming model that allows for a simplified development of implementations of algorithms that are to be run over a distributed infrastructure (cluster, gird, or cloud). The latter is an actual pool of machine learning and analytics algorithms implemented within the COMPSs programming model and Python, incorporating external codes like Message Passing Interface (MPI), if and when needed. The utilization of COMPSs allows the users to program their applications through a sequential paradigm with an annotation of code pieces that can be parallelized, while the actual parallelization is done automatically by the COMPSs runtime module. This allows for a short development time with respect to standard solutions. The algorithmic pool will be evolving as the platform lives, and the implementations will be available in the I-BiDaaS knowledge repository for re-use [5]. The pool also includes the dislib library [7] of machine learning algorithms developed in COMPSs. In addition, the newly developed COMPSs algorithms implementations developed within I-BiDaaS will be considered to be contributed to dislib, under appropriate licensing.

The batch processing module will be exposed to the I-BiDaaS user interface so that various modes of operation will be available to end users. More precisely, the users will be able to utilize ready-to-use algorithms from the pool that are activated via one click from the user interface. In addition, while the algorithms' parameters will have built-in default values, they will also provide a user interface for tuning parameters without the need for re-coding. Furthermore, code templates will be available so that new-but-similar-to-existing applications are not developed from scratch. Finally, interactive querying to facilitate exploration and experimentation will be enabled by the Qbeast [3][4].

2.1 Integration of the batch module in the I-BiDaaS platform

The position of the batch processing module with respect to the overall I-BiDaaS platform is described next (Figure 1). The module interacts with the Universal Messaging (UM), Advanced Visualizations, COMPSs runtime, and Hecuba DataBase (DB) system, including Hecuba database and Qbeast indexing system. Another important interaction of the module is with the Test Data Fabrication platform that is accomplished indirectly through the Universal Messaging. In more detail, the interactions are as follows. The batch processing module gets data from the Hecuba DB system, the main database system of the I-BiDaaS platform. The COMPSs runtime module enables that the tasks annotated as parallelizable within the COMPSs programming model are actually executed in parallel. Interactive querying to facilitate exploration and experimentation will be enabled by Qbeast. Through the Universal Messaging, the module delivers to the Test Data Fabrication tool (TDF) certain analytics results and meta-data that are used for semi-automatic and automatic data fabrication. One can see that, in a broader sense, due to tight interactions, the batch processing module encapsulates in its software stack Test Data Fabrication (TDF), COMPSs, Hecuba, Cassandra, and Qbeast.

Integration with the streaming analytics module

We also consider integration of the batch analytics module with the streaming analytics module. In particular, as detailed in D5.2, the batch analytics output therein is passed to the SAG's Apama Complex Event Processing engine (CEP). Besides that, the Apama CEP engine will be

also able to filter incoming streaming data, using appropriate rules that have been generated from the analysis of the historical data. As initially reported in D3.1, this filtering will be performed by the underlying FORTH's GPU-accelerated pattern-matching component. In particular, the filtering rules will be delivered to the GPU, where they will be applied to the incoming streaming data. This will be helpful in cases where the incoming data rate is too high to be processed by the Apama CEP engine; by using appropriate (use-case specific) rules, the GPU-accelerated component will be able to quickly filter out the data that do not need to be processed further, at a quite early stage.

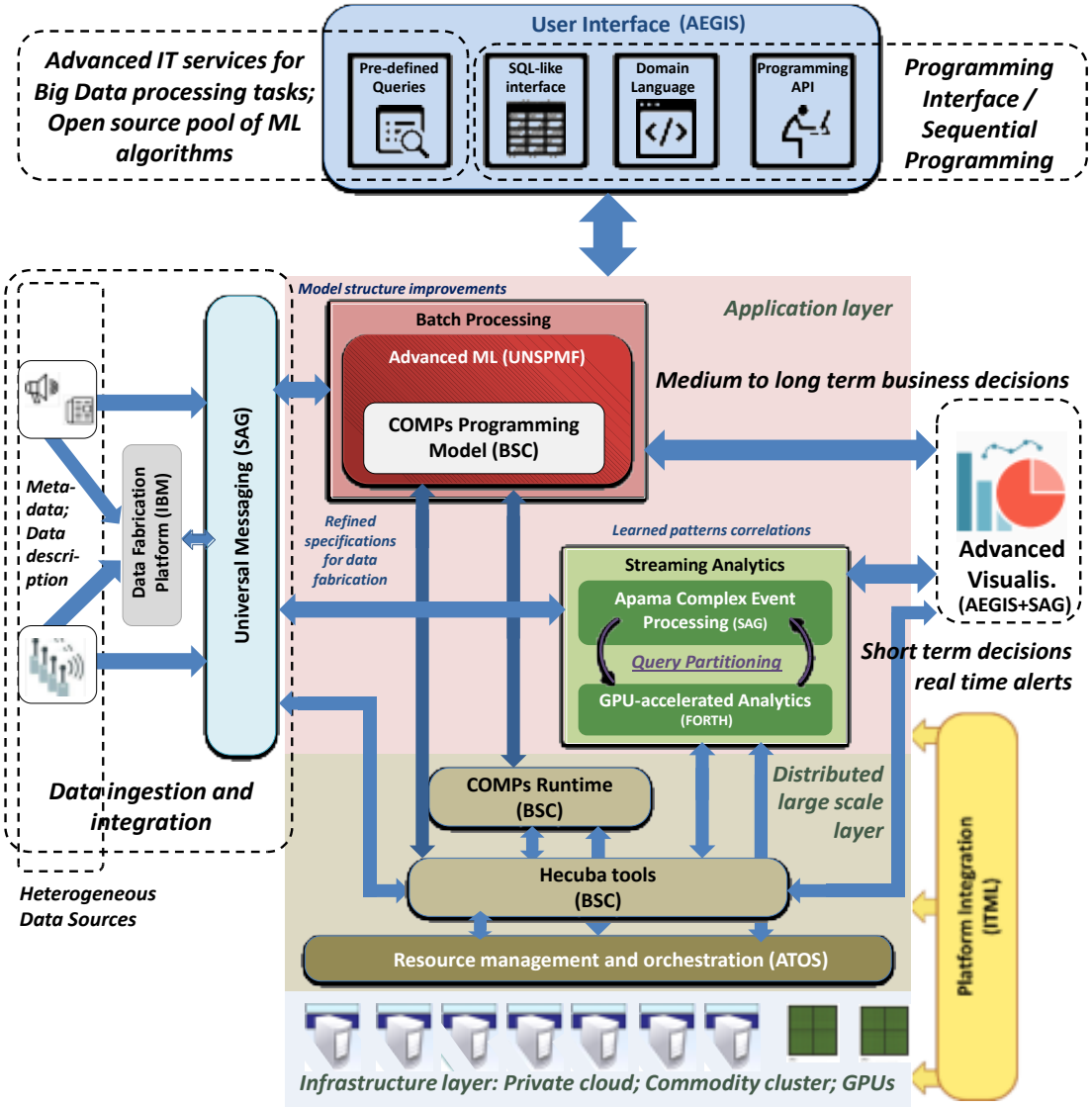

Figure 1: I-BiDaaS platform

Feedback from analytics results to problem modelling

Additionally, we will consider to include in the module a way to provide feedback from analytics results to problem modelling. In more detail, in addition to saving results, we also save all the running parameters of the algorithms, within the central platform database. This means that, in addition to the data analysis task results, we will also save the data about algorithm properties and parameters, platform resources used, execution time, etc. The purpose of this data is to make the end-user experience better in such a way that the modifiable

parameters are suggested to the user, while setting up a new data analysis task. These parameter recommendations will be based on the previous runs of the same or similar models within the I-BiDaaS platform. The parameter recommendations will be visible as placeholders in the web interface of the platform. While one suggested value is probably enough, possibly all the previous runs could be shown upon request to the end-user, so that she/he can decide on which preset parameters to use. This will be especially useful to new or inexperienced users of the platform. In addition, we will consider feasibility of automatic recommendation updates of the parameters for a given model, based on the saved parameter values of previous runs and experiments of the same or similar model.

Preparation and pre-processing

We also consider to include basic data preparation capabilities, such as:

1. Data cleaning
    - Removal of null/NaN (not a number) values from the dataset;
    - Column removal/selection – not all columns are relevant so we can choose to remove certain columns;
2. Data transformation
    - Categorical to numerical data – useful for ML algorithms that can work only with numerical data;
    - Normalization – possible normalization to, e.g., [0,1] values by row or by column in the dataset, again useful for ML tasks;
    - Column type transformations – e.g., Date to Day, Month, Year, and similar.

Implementation details for the current I-BiDaaS integrated solution

In its current implementation (see also deliverable D5.2), the batch module consists of a Python package with several different scripts (modules). These modules represent the code used for running data analysis tasks. When a new algorithm is implemented, it should follow the same pattern. The code for running the data analysis task should be a Python module or if needed a new sub-package with a main entry point where different arguments can be specified. The main entry point is important because these modules are invoked with a bash shell script. This script runs the code with user specified parameters like data location, algorithm parameters, platform parameters (e.g., number of nodes, needed RAM) etc., on the I-BiDaaS architecture. The bash script is the main integration point between actual data analysis code and the user interface. It also serves as a layer of abstraction for the I-BiDaaS architecture, e.g., it can run on a scheduler or in Docker. Currently, as with the I-BiDaaS MVP (see also D5.2), the script uses the "launch-compss" command to run the code on a COMPSs runtime. The script is invoked directly from the Web application, upon user request.

Regarding visualization, the interaction with the batch module depends on the type of the data analysis output. One way to visualize data is to achieve it directly from the python module, saving it as a result of the data analysis task. In alternative, one can use the results from the data analysis task which have been written back to the data storage service in the I-BiDaaS platform directly from the Web application, upon user request. The latter approach is more customizable from the end-user perspective as multiple visualization requests with different parameters can be performed on the same result data.

3 Description of the Use Cases

This section briefly describes the use cases that we are considering as part of I-BiDaaS up to the month 18 horizon of the project, as well as the batch analysis carried out on the corresponding use cases up to month 18.

3.1 Online Bank and IP connection analysis

We describe here a clustering-based analysis of CAIXA's IP address connections synthetic dataset which corresponds to the I-BiDaaS MVP (see D3.1 and D5.2 for details). This represents a follow-up analysis of the use case with respect to the I-BiDaaS MVP (see D3.1 and D5.2). The purpose of the analysis is to provide additional modelling possibilities to this CAIXA's use case.

We briefly describe the data set, the proposed process for data analysis, and outline the results. The obtained results should be understood relative to the fact that the data set utilized is synthetic. Even taking that into consideration, the initial feedback from CAIXA about the usefulness of the developed process is positive, and the approach is promising. The analysis with synthetic data has allowed to extract some insights from the IP connections data with much less effort than using real data, and they should be validated over the real data in order to completely evaluate the benefits of synthetic data analysis approach. Moreover, the next phases of the project assume that a real, tokenized data set for the same use case will be made available. Henceforth, the analogous analysis will be carried out on the real data set as well. While the results reported here correspond to a single-node implementation and moderate-size data set, as provided in the I-BiDaaS MVP, the same analysis can be repeated on the I-BiDaaS platform, as both K-means and DBSCAN algorithms are available in dislib.

The data set contains 72.810 instances, with each instance containing the following attributes:

- *User ID* – representing a unique identification number for each user;
- *IP address* – representing the IP address of the connection of the user;
- *Date* – representing the date and the time of the connection made by the user;
- *Operation* – representing the code of the business operation made by the user;
- *Status* – representing the code of the status of the operation made by the user.

Initially, the dataset is transformed as follows: each user represents a sample, while each IP address represents a feature. In such a data matrix, the value in position (i,j) represents the number of times user i connected via IP address j. Such a dataset turns out to be extremely sparse. In order to tackle this problem and retain only meaningful data, the next pre-processing step is to drop all the IP addresses that were used by only one user (intuitively, such IP addresses represent home network, etc. and thus cannot be used to infer relationships between users). After dropping all such IP addresses, 1.075 distinct IP addresses remain, which represents a significant reduction compared to the initial 22.992 IP addresses contained in the original dataset. Subsequently, we filter out the users which are not connected to any of the remaining IP addresses.

In order to infer relationships between users, we applied clustering algorithms. In particular, we used K-means and DBSCAN algorithms, which are both available in the dislib library. Additionally, we used the t-distributed Stochastic Neighbor Embedding (t-SNE) method to visualize the reduced dataset in 2D. The visualization is presented in Figure 2.

Both K-means and DBSCAN offer some interesting hyperparameters. In particular, K-means allows us the flexibility of setting the desired number of clusters. On the other hand, DBSCAN decides on the number of clusters internally, while providing us with the parameters that represent the minimum number of samples in a neighborhood for a point to be considered a core point, and the maximum distance between two samples for them to be considered as in the same neighborhood. These parameters are to be set by an end user based on experimentation and domain knowledge, and will be tunable through the I-BiDaaS platform user interface.

Clustering was performed on both the full 2.009×1.075 dataset, as well as the t-SNE reduced 2.009×2 dataset. We used silhouette score as the metric of choice for evaluating the clustering. Roughly, the silhouette score evaluates how well each point fits the cluster it was assigned to versus the next closest cluster. The values of the silhouette range from -1 to 1, where 1 represents perfect clustering.

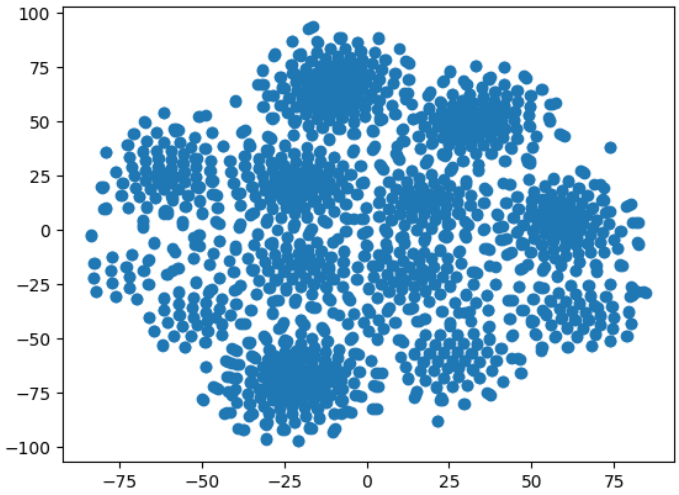

Figure 2: t-SNE 2D visualization.

We evaluate performance of K-means and DBSCAN for different values of parameters. In particular, we experimented with the number of desired clusters (*K)* for K-means and the parameter for defining the maximum distance between two samples for them to be considered as in the same neighborhood (*eps)* for DBSCAN. The parameter representing the minimum number of samples in a neighborhood for a point to be considered a core point (*min_samples*) for DBSCAN was fixed at 2. This is intuitive, since we want to allow the algorithm to find clusters of (at least) 2 users in order to infer relationships. For K-means, we consider 6 different values for *K:* 13, 600, 700, 800, 900, 1000. The choice is motivated as follows: the visual test, based on t-SNE projection suggests roughly 13 clusters. On the other hand, DBSCAN initially found around 600 clusters. Based on this, we decided to try the small value suggested by t-SNE as well as the higher values suggest by DBSCAN. The results are presented in Table 1.

For DBSCAN, we chose 6 different values for *eps*: 0.1, 0.25, 0.5, 0.75, 1, 1.25. Since 0.5 is the default value, we decided to experiment with values slightly lower and slightly higher than the default. Along with the silhouette scores, we report the number of clusters found by the algorithm for different values of the parameter. The results are presented in Table 2.

Table 1: Silhouette scores for K-means clustering.

Desired number of clusters K	*Silhouette score on full data*	*Silhouette score on t-SNE data*
13	-0.015	0.428
600	0.501	0.578
700	0.601	0.671
800	0.696	0.768
900	0.762	0.856
1000	0.753	0.846

Table 2: Silhouette scores for DBscan clustering.

Maximum distance (eps)	*Silhouette score on full data*	*Silhouette score on t-SNE data*	*Num of clusters on full data*	*Num of clusters on t-SNE data*
0.1	0.656	-0.689	665	97
0.25	0.656	0.164	665	580
0.5	0.656	0.394	665	706
0.75	0.656	0.507	665	770
1	0.681	0.808	693	882
1.25	0.681	0.815	693	881

Additionally, we evaluated the sizes of the obtained clusters. A recurring effect is that DBSCAN finds 1 big cluster (e.g., when the number of clusters is 665, DBSCAN finds 1 cluster containing 629 points and when the number of clusters is 693, the big cluster contains 570 points). Interestingly, except for the single big cluster, DBSCAN finds only clusters of sizes 2, 3, 4 and 5, with clusters of size 2 dominating (e.g. 618 and 643 clusters of size 2 when 665 and 693 total clusters found). Regarding K-means, the algorithm also generates a single big cluster and clusters of sizes 2, 3, 4, 5, as well as clusters that contain only 1 point. In general, the following pattern emerges: as K grows, the size of the big cluster decreases, but the number of clusters containing only 1 point grows. Once more, the size 2 clusters dominate.

3.2 Bank Transfers analysis

CAIXA defined a use case focused on an advanced analysis of bank transfers executed from the financial terminal of the employees. When a client goes to a bank office, it correctly identifies himself and orders a bank transfer, the employee can, in the name of the client, use its terminal to execute the bank transfer. The goal is to detect potential fraudulent transfers or

bad practices (e.g. the client was not present in the time of the transfer) using anonymized data, which will allow to perform the analysis out of the premises of the bank.

A dataset has been generated joined different logs. Each row is composed of 87 attributes (some attributes are, for example, employee, time, account, money or client). The data is encrypted without losing value (the Dice Coefficient [9] is used to obtain the similarity of the encrypted data).

BSC will try to detect these anomalies by using clustering analysis based on the K-means algorithm, which is a usual methodology used for anomaly detection [10]. The aim is to find the points that are more distant from the centers of the clusters already computed, in order to detect these anomalies.

Different anomaly detection algorithms are already implemented inside CAIXA: Isolation Forest, One Class SVM, Local Outlier Factor (LOF), Double Median Absolute Deviation (MAD), Anomaly Detection with Supervised Learning (XGB) and Mahalanobis Distance. The new solutions and processes to analyse data outside CaixaBank premises will be compared with the already implemented ones.

The expected benefits and revenues are to establish a testing environment for new Big Data tools outside of CAIXA premises, and to open CAIXA data to a wider community and explore novel data analytics methodologies. Other commercial data analytics tools, such as DataRobot [11], has been recently tested by CAIXA. However, the integration of new technologies inside the CAIXA premises (event for small proof-of-concept deployments) implies an arduous internal process. For that reason, they have been also tested on cloud approaches. The same data tokenization process proposed for I-BiDaaS has been used in those cases. Beyond that, a preliminary comparison of DataRobot and I-BiDaaS has been done. Even being a more mature data analytics product, some limitation has been found in DataRobot as well. DataRobot provides the possibility to test a benchmark of data analytics algorithms, but the tuning of them (e.g. the scoring function for specific columns) is very limited. I-BiDaaS provides an "expert mode", capable of defining your own code and expanding the data analysis possibilities, customization and flexibility.

BSC has started with the data cleaning and with the analysis in order to determine if our already implemented clustering algorithms fit with the anomaly detection requirements of our dataset.

The first step has been to encode the categorical attributes into numerical values and to standardize the data. We have used two different approaches depending on the characteristics of the categories: for high-cardinality (more than 10 values) and exclusive classes, we have used label encoding, and for low-cardinality and not-exclusive classes we have used one-hot encoding [12]. To standardize the data, we have used the method *StandardScaler* from the scikit-learn library [27].

After these changes, the number of attributes increased from the original 87 to 234. This dimensionality complicates the interpretation of the clustering results. For this reason, the second step has been to reduce the attribute vector dimension to 4, applying a Principal Component Analysis (PCA) method from the scikit-learn library [27]. Figure 3 shows the code lines that we have executed.

```
pca = PCA(n_components=4)
pca_result = pca.fit_transform(x)
principal_df = pd.DataFrame(data=pca_result)
```

Figure 3: Code to reduce the data dimensionality

Finally, we have executed two different clustering algorithms: DBSCAN and K-means. As K-means requires the desired number of clusters as an input parameter, we have executed first DBSCAN and we have used the obtained number of clusters as the input parameter of K-means. Figure 4 shows the graphical representation of the clusters that we have obtained with both algorithms (each colour represents a cluster). We can observe that some values are not part of any cluster and thus, they are potential anomalies in the data. The final step is to confirm that these values are actual anomalies, which is part of our ongoing work.

Notice that these experiments have been executed without using the tools that are part of I-BiDaaS, just to evaluate the feasibility of the idea before adapting our tools to the use case. Once we have confirmed that this methodology has enough accuracy for the use case, we will implement a version of the K-means based on the tools of the I-BiDaaS platform, which are designed to show good performance for a big amount of input data.

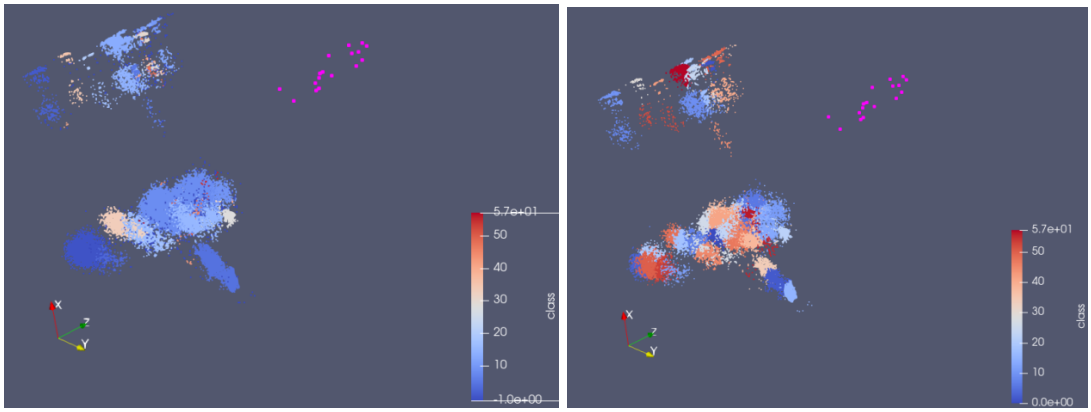

Figure 4: DBSCAN (left) and K-means (right) results

3.3 Phone calls transcription analysis

The use case of the analysis of the phone calls transcriptions has been agreed to be a streaming use case for the month 18 horizon of I-BiDaaS. We include here a brief description of the use case and the corresponding data set, for completeness. One of the main aims of TID is to improve the customer experience in its various Call Centers. As has already been described in D1.1, a Call Center is formed by multiple processes and faces many challenges. Thereby, a very important goal is to derive representative customer satisfaction metrics that can encapsulate the effectiveness of the provided service in a CC scenario. Examples of currently used practices include among others, the duration of the process until an incident has been resolved and the "hold on" time that a customer has experienced if the CC agent requested all the appropriate information from the customer to further proceed. However, more advanced analysis is required, including the employment of advanced machine learning techniques and application of streaming-based models for real-time script adaptation from agent.

Obviously, measuring satisfaction is quite a complex task and sometimes it relies on a subjective evaluation. Many tools are used to identify how TID's customer is satisfied, but the best solution should be to estimate the Customer Satisfaction Index (CSI) based on the real customer-agent interaction along the conversation. As such, this use case aims to detect customer satisfaction and dissatisfaction based on the automatic transcripts of the calls. In addition to possible words and phrases recommended by business units, additional sources of information related to customer satisfaction are taken into consideration, e.g., self-reported satisfaction at the ending of the call.

The training dataset that will be used for this use case consists of meta-data generated over a mixture of structured and unstructured data. In particular, 20 hours of speech that is manually transcribed, and 80 hours of speech that include automatic transcriptions. In both cases, data is anonymized and personal data is removed. The dataset will be used offline for generating appropriate models (i.e., Convolutional Neural network) that will be then used by the Apama CEP engine, to classify incoming streaming transcripts at real-time.

3.4 Aluminium casting use case

We report here how the I-BiDaaS batch processing module will be used for the I-BiDaaS aluminium casting use case based on the FCA foundry use case data set. A real anonymized data set has been made available to the I-BiDaaS consortium during May 2019. We next briefly describe the use case, as well as the proposed modelling approach, and how to apply the I-BiDaaS batch processing module to the use case.

As described in D2.1, the data for this use case is collected from various sources, such as sensors, during the process of aluminium casting. Additionally, the Operator's data is procured, in the form of qualitative evaluation of the process, events, etc. (e.g., manually detected defect). The use case is concerned with the production process of aluminium casting. The goal is to use Big Data for improving the quality of the production process and operational efficiency, in particular, the quality issues on the automotive component. There are many sensors installed on each component. The process of aluminium casting is much more difficult than traditional casting. The main objective is to link all the data during the production process to component defects. Each produced component is completely evaluated and visually inspected. The goal is to correlate defects with the production process characteristics. Potential KPIs for the use case include Scrap percentage, Cost per unit, Quality control and repair costs. The data set contains production, process and control parameters of the production of components bases by die-casting and represents a mixture of structured and unstructured data. The corresponding data set is a real anonymized data set provided to the consortium by CRF.

We initially performed classification using the supervised learning classifiers with the available COMPSs implementations, from the dislib library (https://github.com/bsc-wdc/dislib): cascade support vector classifier and random forest classifier. The results are available to the consortium. In addition, we will consider applying the newly implemented distributed alternating directions method of multipliers (ADMM) algorithm (see Section 4.1.1). The ADMM method is very flexible and can be used here to perform the binary classification problem in several ways, e.g., by applying the logistic loss, hinge loss (support vector machines), or exponential loss, on the given dataset.

In addition, the random forest classifier can be used to assess the feature importance and possibly point to the most important parameters in the process that determine the outcome of the classification. A visualization approach using t-Distributed Stochastic Neighbor Embedding (t-SNE, [15]) is also proposed, to visualize the data in 2D and see whether there is any structure emerging.

Finally, we report that during the I-BiDaaS hackathon in Campus Melfi, organized by CRF, June 18-19, 2019, we performed analysis of an enlarged data set, with additional features, and the results of the analysis are available to the consortium.

4 Advances in the software architecture

This section reports the advances in the development of the components of the Batch module of the I-BiDaaS platform. First, we describe the advances related to the Machine Learning components and, second, we describe the advances in the technological components of the platform.

4.1 Machine learning advances

4.1.1 Algorithms

We summarize significant contributions with respect to algorithmic implementations reported in D3.1. In particular, we report here on an improved and generalized COMPSs implementation of the Alternating Direction Method of Multipliers (ADMM) optimization algorithm [16]. ADMM is a primal-dual optimization algorithm that solves the augmented Lagrangian of a given problem. It is well suited to problems that can be decoupled, either by samples, or by features. Because of this, it fits particularly well for parallel implementations. Furthermore, ADMM provably works irrespective of the choice of the tuning parameters, can solve a very wide class of problems (including many machine learning problems like logistic regression, support vector machines, sparse regression, etc.), and despite generality and flexibility in terms of tuning parameters achieves a close-to-state-of-the-art performance [16].

Our implementation of ADMM utilizes COMPSs and CVXPY, a mature Python library for convex optimization, and it requires minimal programming effort to adapt the code to various optimization problems. In other words, our implementation integrates two mature open source libraries (COMPSs and CVXPY), where CVXPY per-node solvers are orchestrated through COMPSs, and COMPSs is harnessed to achieve good parallelization. As such, this corresponds to a solid technological contribution and innovation. The implementation of ADMM for the Lasso problem is available at the I-BiDaaS knowledge repository [6].

Brief review of ADMM

The ADMM algorithm works as follows. For a given optimization problem:

$$\begin{aligned} &minimize\, f(x) + g(z) \\ &subject\, to\, Ax + Bz = c\,, \end{aligned}$$

ADMM defines its corresponding augmented Lagrangian given by

$$\mathrm{L}_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + \tfrac{\rho}{2}\|Ax + Bz - c\|^2,$$

and solves the problem iteratively, using the following update rules

$$\begin{aligned} x^{k+1} &= arg\min_x L_\rho(x, z^k, y^k) \\ z^{k+1} &= arg\min_z L_\rho(x^{k+1}, z, y^k) \\ y^{k+1} &= y^k + \rho(Ax^{k+1} + Bz^{k+1} - c)\,. \end{aligned}$$

Here, k indexes the iteration counter, x and z are the primal variables, y is the dual variable, and ρ is the Lagrangian penalty parameter. The algorithm is guaranteed to converge to a solution of

the desired optimization problem under mild assumptions, irrespective of the selection of the penalty parameter.

Description of implementation

Our COMPSs implementation is based on the master-worker framework, whereby each worker has a local copy of the primal variable x and the dual variable y, while the master updates the primal variable z. In our context, we utilize CVXPY to calculate the update for the primal variable x. We exploit COMPSs to distribute the local computing of the x update, as well as to distribute the dual update of y. This means that in our implementation, we have two tasks annotations, one for the x update, while the other for the y update.

Our implementation of ADMM preserves the generality of the algorithm. By relying on CVYPY, we offer a generic code, easily adaptable to a multitude of problems. The only modification required by the user is to adapt the x update, by defining the objective function for CVXPY to optimize. In general, a minor change in the code allows ADMM to be adapted to various problems, such as logistic regression, SVM, etc.

Regarding the stopping criterion, we implement the same stopping, with respect to primal and dual residuals, as in equation (3.12) in [16].

We next detail on the COMPSs task annotation process. We focus on the least absolute shrinkage and selection operator (Lasso) problem [7], while a similar process is carried out for other modeling problems. We identify several important parallelization points. First, the process of reading the input data can be annotated as a COMPSs task. The input matrix is divided into N files, where N is the number of workers. The same holds for the input vector. Then, the functions that read the data chunks from files (both the matrix and the vector parts) are annotated with tasks. Thus, each worker reads its own part of data. It is important to also provide a call to *compss_wait_on* from *pycompss.api.api*, in order to ensure synchronization, while the data is collected from all the workers.

We consider three variants of the algorithm. The first one relies on CVXPY. Considering the ADMM algorithm, there are two points of interest to parallelization. The first is the x-update. Here, each worker operates on its own data chunks for the matrix and vector, so the function *x_update* can be annotated as a task. Here, we also provide a *compss_wait_on* call after it in order to collect the results from different workers properly. The same principle holds for the second point for parallelization, updating the value of u, the function *u_update*.

The second variant of the algorithm exploits warm-starting option of CVXPY, while the third variant (currently for the Lasso problem only and under development) exploits the fact that the x-primal updates for Lasso can be solved in closed form.

Initial testing results

We report on the results of testing of the algorithm in terms of accuracy and scalability with respect to the number of nodes and on the Lasso optimization problem. The tests were carried out on UNSPMF local computer cluster, which has 16 nodes and 100 CPU cores. Further cluster specifications are as follows: CPU: 8 x Intel i7 5820k 3.3GHz; 8 x Intel i7 8700 3.2GHz; RAM: 16GB DDR4/node (64GB DDR4 on master node); storage: 24TB dedicated Supermicro storage server; 128GB SSD/node; Network interconnection: 10 Gbps ethernet. The input data matrix is generated in a random fashion, as described in Section 11.1 in [16].

Regarding accuracy, we test in on small-scale problems, where we use as the benchmark the solution to the optimization problem obtained by the corresponding (single-node, sequential) CVXPY solver. Note that CVXPY is a single-node solver, and hence testing accuracy in a

similar fashion over large scale problems may be infeasible. We consider the relative error, i.e., the 2-norm of the difference between the solutions obtained by COMPSs and by CVXPY, divided by the 2-norm solution obtained by CVXPY. Table 3 shows relative error for various (moderate) sizes of the input data matrix, as well for various numbers of cluster nodes. We can see that the achieved accuracy of the algorithm is good.

Table 3: Accuracy testing for the COMPSs implementation of ADMM-Lasso

Input data matrix	Relative error	Number of cluster nodes
36 x 20	4.4406 e-04	2
96 x 50	2.2956 e-03	3
480 x 100	4.6081 e-02	12

We also report on the scalability tests of the ADMM-Lasso implementation in COMPSs. We fix the input data matrix of size 9,984 x 5,000, i.e., the data set has 9,984 samples and 5,000 features. We then run the algorithm for various number of working nodes, and measure the execution time. Figure 5 plots execution time versus number of cores. We can see that the algorithm scales very well with the number of cores.

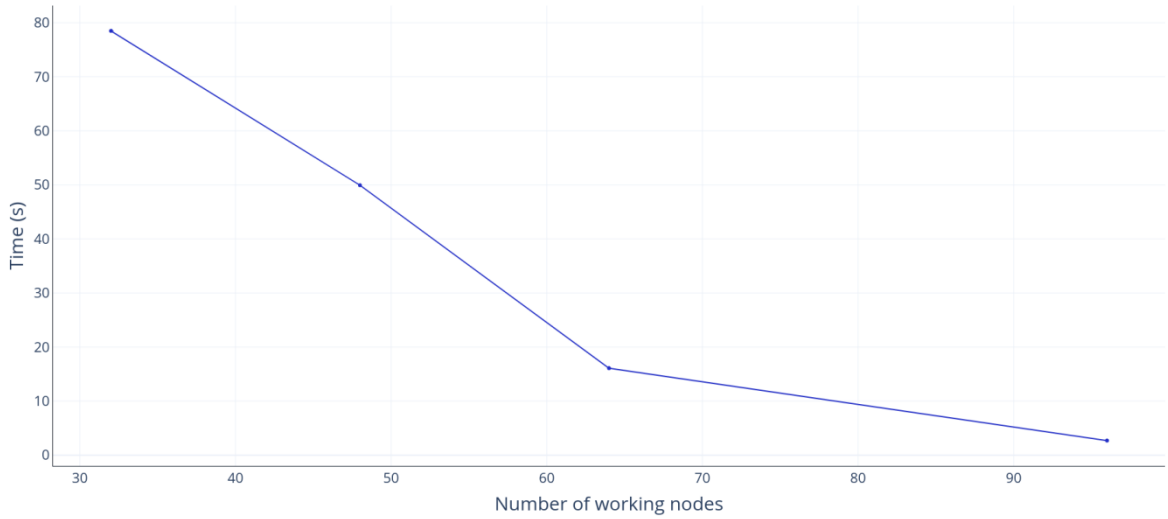

Figure 5: Scalability of the ADMM-Lasso COMPSs algorithm.

We also report in Table 4 how the number of iterations until stopping changes with the number of worker nodes.

Table 4: Accuracy testing for the COMPSs implementation of ADMM-Lasso

Number of worker nodes	Number of iterations
32	50
48	40
64	30
96	5

Ongoing work

As part of ongoing work, we are attempting to implement in COMPSs novel distributed optimization algorithms [19] [20] [21] [22] [24] and [23], as solvers for ML and general purpose problems. Both algorithms introduce certain desirable features. The algorithms in [19][20] [21] [22] are variants of the distributed gradient descent with a sparsified communication model. At each iteration, the number of active agents (workers) is reduced in a probabilistic fashion with respect to a standard schema where all workers are active at all iterations. In this way, the per-iteration computational and communication costs of the algorithm are reduced. The algorithm provably achieves an improved convergence rate with respect to the corresponding standard distributed algorithm where all nodes are active at all times. The algorithm in reference [24] is also based upon the distributed gradient descent algorithm, and it is developed utilizing the methodology of the Trust region optimization methods [24]. The main benefit of this algorithm is the automatic step size selection performed by each node locally, at every iteration. Additionally, while the number of active agents stays the same throughout every iteration, if the computed step does not yield significant decrease of the objective function, the step is rejected. In this case, the agent does not communicate its solution to the master node, and is already stored by the master node at the previous iteration. The implementations of the methods here will be considered for application of the FCA foundry use case of I-BiDaaS.

In addition, based on the implementations reported in Section 5.2 of D3.1, we are developing an implementation of a differentially private decision tree algorithm [25][26][33], where a sequential Python implementation with good accuracy is currently available. Subsequent work will consider a differentially private implementation of the K-means algorithm. Application of the differentially private implementations we develop will be considered for the use cases after the month 18 horizon of the project.

4.1.2 Dislib

The Distributed Computing Library (dislib) provides distributed algorithms ready to use as a library. So far, dislib is highly focused on machine learning algorithms, and it is greatly inspired by scikit-learn [27]. However, other types of numerical algorithms might be added in the future. The library has been implemented on top of the PyCOMPSs programming model, and it is being developed by the Workflows and Distributed Computing group of the Barcelona Supercomputing Center. The library dislib allows easy local development through docker. Once the code is finished, it can be run directly on any distributed platform without any further changes. This includes clusters, supercomputers, clouds, and containerized platforms.

By now, the machine learning algorithms implemented are:

- Random Forests Classification
- Cascade Support Vector Machines Classification
- K-Means Clustering
- DBSCAN
- Gaussian Mixture Model
- K-Nearest Neighbors
- Alternating Least Squares Recommendation

We are going to implement the machine learning algorithms following this interface in addition to benefiting from those that are already part of dislib. In order to fully harness this within I-

BiDaaS, we need to integrate dislib with Hecuba and Qbeast. In addition, the newly developed COMPSs implementations within I-BiDaaS will be considered to be added to dislib.

4.1.3 Integration with data management

In order to fully integrate dislib, and all the machine learning algorithms that we have developed, as part of the I-BiDaaS platform, it is desirable to implement their integration with Hecuba. The algorithms that dislib implements work on numpy ndarrays. Hecuba already provides an implementation of numpy ndarrays thus the only methods that need to be added are the required means to load the data from the database. In addition, we want to extend the Hecuba management of numpy ndarrays with the capability of reading and storing the ndarrays by blocks. We expect that this feature will speed up the performance of the interaction with persistent numpy ndarrays. Thus, we will also modify dislib to delegate to Hecuba the partition into subsets of the ndarray.

Another modification that we need to make to dislib is to add the option of initializing the data for the algorithms. This initialization is necessary to integrate dislib with Qbeast, in order to improve the performance of the algorithms using the sampling capabilities of Qbeast (see section 4.2.2). This integration will benefit not only the users of the I-BiDaaS platform, but also the users of dislib (which is an open source software) that will be able to use this enhanced version of the algorithms.

4.2 Advances in the technological components of the platform

4.2.1 Hecuba advances

We have added to Hecuba a filtering functionality for the persistent data managed by Hecuba, which can be used in the bank transfer analysis use case. The interface with the programmer is the same as with the regular Python filter. Hecuba intercepts the calls to the python filter method, and if it is applied on a regular Python object, then redirects the call to the regular Python filter function. On the contrary, if the filtering is applied on a Hecuba object, then the call is redirected to our new functionality. This new functionality, allows the programmer to query the database and fetch the data in chunks, instead of filtering all the data in memory. The filter parses the lambda passed to it, and then it is translated to Cassandra Query Language, or CQL [28]. Then, the C++ part of Hecuba is in charge of creating the iterator that will iterate over the rows using a specified prefetch size set by the environment variable "PREFETCH_SIZE", with a default value of 10,000. In addition, when the Hecuba object is indexed using Qbeast, we can use this filtering functionality to get just a sample of the data.

We have also modified Hecuba to implement dynamic task granularity. An application may have different tasks executing in parallel, and each task will benefit from a different granularity. The dynamic task granularity tries to choose, in runtime, the most efficient granularity for the PyCOMPSs tasks that are being sent. Previously, programmers had to choose the granularity in their code, or execute the tasks with a default granularity. This new functionality is already tested in MareNostrum4 using the use case provided by CAIXA (Online Bank and IP connection analysis) with randomly generated data, and performance of the application is increased up to 29%. Of the three different parallel tasks of this application, the performance of one of them is increased greatly, and for the other two, the performance is almost the same.

Apart from these new features, several improvements have been made both in performance and bug fixes. For example, the previous C++ implementation of a cache has been replaced by a significantly more efficient new cache. Also, a timestamp has been added to the writing queue

in order to assure last write wins. We have also ported Hecuba to Python 3 because dislib does not work with Python 2.

4.2.2 Utilization of Qbeast to improve K-means

We are studying the utilization of Qbeast to improve the convergence rate of the K-means algorithm, which can be used, for example, in the bank transfer analysis use cases provided by CAIXA.

Qbeast is a multidimensional highly-scalable indexing system that allows us to obtain data from disk much faster than classic management of data. Also, Qbeast reads asynchronously the requested data, which allows us to consider the following idea: each iteration of the clustering process can work only on the sample of the data already read. Thus, while the computation of the iteration is running, Qbeast can provide additional data that can be used in the following iterations. Our expectation is that, as this approach overlaps computation with data loading, it will speed up the convergence of the algorithm.

In order to test the feasibility of the idea before modifying the implementation of dislib, we have adapted an already existing K-means algorithm, included in the Scikit-learn library [27], to test two different approaches for the optimization: Bi-phase K-means and Multi-phase K-means. These optimizations consist on performing K-means executions with fewer data while we wait all the data to be loaded, so that when we have all the data ready to be computed. we have input centers that improve the overall cost of the algorithm significantly. These tests are simulating how Qbeast will query the data.

Bi-phase only performs two executions of the algorithm, the first with a small percentage of the data and the last one with all data. On the other hand, Multi-phase performs multiple K-means executions with an increased percentage of the input data for each phase.

We have executed the different versions of K-means on a data set of 26G (the dataset is geo-tagged market transactions from OpenStreetMap dataset) using two dimensions (coordinates x and y). The multi-phase algorithm executes on several samples of data, which size is increased at each iteration.

Table 5 shows a detailed analysis of the execution of each version using 8 nodes. This table shows the execution time of each phase, considering both the data loading time and the computing time, the size of the sampling (percentage of data used), and the number of iterations required to converge.

Table 6 shows the results of analyzing the scalability of the K-means optimizations. Bi-phase K-means shows best performance among all, being up to 4 times faster than normal K-means, on a system with one node. For these tests, the Multi-phase algorithm has shown slightly slower times than Bi-phase but better than Normal K-means in all cases.

Table 5: Analysis of K-means optimizations

	PHASE 1		PHASE 2		PHASE 3		PHASE 4		
Normal	**load**	**execution**							**total**
percentage of data	100%								
time (s)	14,33	184,75							**199,08**
iterations per phase	173								
Bi-phase	**load**	**execution**	**load**	**execution**					**total**
percentage of data	1%		100%						
time (s)	21,44	37,61	0,1	28,78					**87,93**
iterations per phase	158		59						
Multi-phase	**load**	**execution**	**load**	**execution**	**load**	**execution**	**load**	**execution**	**total**
percentage of data	0,10%		1%		10%		100%		
time (s)	16,16	19,21	4,94	7,31	4,98	5,19	7,31	22,85	**87,95**
iterations per phase	99		45		21		18		

Table 6: Scalability of K-means optimizations

Number of nodes	1	2	4	8
Normal	1688,475	900,569	434,827	199,08
Bi-phase	421,525	213,885	127,155	87,93
Multi-phase	597,846	277,444	148,873	87,95

Right now, we are focusing on understanding how the sampling size impacts the number of iterations in relation to the sampled data. Our objective is to analyze how we can reduce the total number of iterations and run-time in relation to the sampled data.

4.2.3 Advances of Test Data Fabrication Tool

The advances made on the Test Data Fabrication tool (TDF) related to the batch module are on the integration with the database and the exploitation of the data analysis in the TDF data modelling automation. The TDF tool supports generation of data into databases and into various standard file formats. For databases, the tool generates a Constraint Satisfaction Problem (CSP) per each table row (or 'wide row' in case where referential integrity constraints exist), whereas for files, the tool generates a single CSP problem for an entire file. The tool automatically imports the data structure from the database metadata or, in case of files, enables the user to manually define the structure of the data using predefined, C-like, elements. In addition to the structure, modelling of the data and the relationships in it using various types of constraint rules is required. Both the data structure description and the model are translated into an abstract CSP and sent to the integrated CSP solver. The solver, in some cases, will have to query the historical data (previously generated by the solver or an already saved data in the database) in order to provide a valid solution. An example where such query is required would be a constraint requiring the generated values to be unique. An additional example would be a constraint requiring the generated values to have certain probability distribution. In general, constraints that are affected by previously generated data or by inter row dependencies will require the CSP solver to query the data base. DB queries performance can significantly degrade the solution calculation time. There are many considerations required to avoid severe slowdown, e.g., frequency of the querying, the time/space complexity of the query (using selects instead of joins and intersects), etc.

Considering these limitations, the solver process calculates a solution, which contains desired data points satisfying all the constraints. This solution is then translated and written into the desired output database tables or files. The supported databases are the commonly used standard relational databases, e.g., DB2, Oracle, PostgreSQL, etc., which are becoming less frequently used in today's big data algorithms. TDF is extended to support Cassandra database as described in subsection 4.2.3.2.

4.2.3.1 Data fabrication rules

Complete automation of the modelling is a difficult task. It is known to be complex and time consuming and is typically expected to be performed by a human expert. However, automating the creation of a part of the model is possible and is very beneficial. The test data fabrication platform is extended to support generation of new data out of rules which are automatically learned and extracted from real/production data.

The batch processing module can analyse the data and store a set of properties. These properties will be automatically consumed by TDF and translated into data fabrication constraint rules which formulate the data model. The properties that can be used by TDF are:

- Data class, i.e., 'code', 'date', 'identifier', 'indicator', 'quantity', 'text', etc. that can be further refined into more concrete types hierarchically, i.e. types of code, sub categories of texts, etc.
- Statistical summaries of columns containing various numeric data, dates and times.
- Statistical summaries of lengths of columns containing text.
- Number of distinct/unique/empty/null values.
- Formats of text columns.
- Format distributions of text columns.

The fabricated data should be 'similar' to the training data, i.e., exhibit similar statistical and distributional properties, similar relationships, etc.

The automation of the data modelling also uses database (DB) metadata i.e., the schema-table-column names, data types and existing database internal constraint rules applied on it (e.g., primary keys, check constraints). Tables and columns names are used to syntactically find best match an internally pre-defined TDF set of rules.

For example, if the data analysis suggested that a column called 'age' falls into a data class 'quantity' and has a normal distribution with mean equal to 39 and standard deviation equal to 5, with minimum value of 18 and maximum value of 85, the modelling logic will create two constraint rules, one is based on a pre-defined template tailored for this data class, and another requiring that exact same distribution.

i. **'ColumnName'** memberOf {[**18,85]**}

ii. **'ColumnName' = normalDistributionNumber(39,5)**

TDF now includes a generic API for importing data analysis results.

4.2.3.2 Integration with Hecuba

Several changes will be required for TDF to support Cassandra DB in a similar way as relational databases are supported. The goal is that TDF will be able to automatically import the structure and to write the solution back to the database.

An initial approach that was considered to this problem was to continue fabricating data the same way as described above and add a post processing phase which simply exports the fabricated data from the DB that was used (this function is supported by all standard DB vendors) and import the data into Cassandra. However, such a solution requires the data to be fabricated completely prior to applying any algorithm on it.

The support that will be added to TDF includes adding a Cassandra driver that can read and write data and metadata. The driver should enable the automatic import of the structure, the historical data fetching and the writing of the solution data back to the DB.

Cassandra DB is a significantly different database compared to DB2 or Oracle which are relational databases. It can efficiently handle very large amount of data and very large number of entries columns in a table however, it lacks some of the operations that are available in relational databases. JOIN, INTERSECT and others are examples for such operations that TDF is currently using during the solver process. The challenges in adding this support are mostly related to finding ways, that are also efficient, to query Cassandra in a similar way as done in relational DBs.

4.3 Advances in the automatic deployment of the Batch-module of I-BiDaaS

One of the goals of the I-BiDaaS project is to facilitate the utilization of BigData tools to users that are not familiar with BigData technologies and platforms. Following this approach, it is very important to facilitate the automatic deployment and configuration of the Batch-module of I-BiDaaS. For this reason, we have decided to use a virtualization technology based on containers. We have chosen to use docker containers [30] and docker swarm [31]. We have automatized the creation of a docker compose file that defines and runs multi-container docker with all the tools of the Batch-module of I-BiDaaS. We have created a quick-start example of how to setup a swarm cluster and execute applications on it. Section 4.3.1 shows the readme of this quick-start, that is also available at [32].

4.3.1 Quick-Start example

This section shows a step-by-step guide of how to setup a swarm cluster, deploying applications on it.

Create virtual machines and setup swarm cluster

First, we need to install docker. Access to: https://docs.docker.com/v17.12/install/#supported-platforms

And follow the instructions for your OS.

After installing docker, go to https://docs.docker.com/install/linux/linux-postinstall/ and follow the steps in "*Manage Docker as a non-root user"*.

Second, we need to install docker-machine:

```
$ base=https://github.com/docker/machine/releases/download/v0.16.0 &&
  curl -L $base/docker-machine-$(uname -s)-$(uname -m) >/tmp/docker-machine &&
  sudo install /tmp/docker-machine /usr/local/bin/docker-machine
```

To install virtualbox go to https://www.virtualbox.org/wiki/Linux_Downloads and follow the instructions.

Then we create the virtual machines:

```
$ docker-machine create --driver virtualbox myvm1
$ docker-machine create --driver virtualbox myvm2
$ docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL                       SWARM   DOCKER     ERRORS
myvm1  -        virtualbox   Running   tcp://192.168.99.100:2376         v18.09.6
myvm2  -        virtualbox   Running   tcp://192.168.99.101:2376         v18.09.6
```

Let's initialize the *swarm (the parameter --advertise-addr has to be the ip of a vm in the previous list)*:

```
$ docker-machine ssh myvm1 "docker swarm init --advertise-addr 192.168.99.100"
Swarm initialized: current node (h2oqpc6zdrgas4qhoael9ld8v) is now a manager.
To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-4q2biqvtm04f29slk6s6yt154aov1e9z98gu7jol24xte8eeqj-6qcoshxgy2stl7wihhyq81vm8 192.168.99.100:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

Now we will add one worker to this swarm, with the following command *(note that you have to use the previously given token, not the one in this example)*:

```
$ docker-machine ssh myvm2 "docker swarm join --token SWMTKN-1-4q2biqvtm04f29slk6s6yt154aov1e9z98gu7jol24xte8eeqj-6qcoshxgy2stl7wihhyq81vm8 192.168.99.100:2377"
```

List nodes in the swarm:

```
$ docker-machine ssh myvm1 "docker node ls"

ID                            HOSTNAME   STATUS   AVAILABILITY   MANAGER STATUS   ENGINE VERSION
h2oqpc6zdrgas4qhoael9ld8v *   myvm1      Ready    Active         Leader           18.09.6
17nyq6xzi8v7m114i7pkw5our     myvm2      Ready    Active                          18.09.6
$ docker-machine ls
NAME   ACTIVE   DRIVER       STATE     URL                       SWARM   DOCKER     ERRORS
myvm1  *        virtualbox   Running   tcp://192.168.99.100:2376         v18.09.6
myvm2  -        virtualbox   Running   tcp://192.168.99.101:2376         v18.09.6
```

Set docker machine shell environment *(this must always be done before executing runcompss-docker in a new terminal)*:

```
$ eval $(docker-machine env myvm1)
```

Setup Cassandra container

Pull Cassandra container image:

```
$ docker pull cassandra
```

Create network to allow communication between containers:

```
$ docker network create --attachable -d overlay cass-net
```

Run Cassandra container attached to the network:

```
$ docker run --network=cass-net --memory 4g --name my-cass -d Cassandra
```

Figure 6 describes the infrastructure of the cluster.

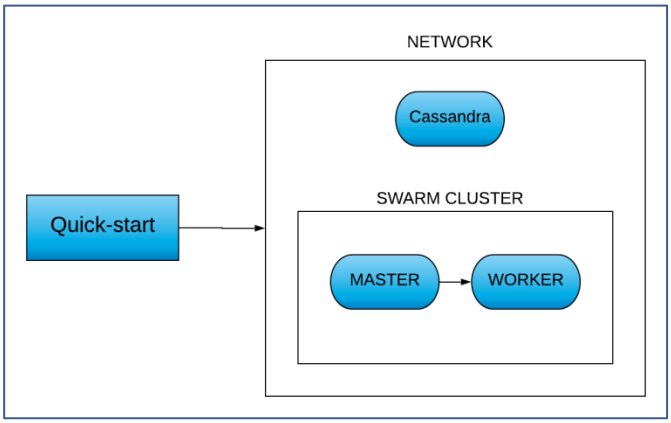

Figure 6: Cluster architecture

Application execution

First, we need to add data to the Cassandra database. We will use datasets/words.csv.

Copy csv to Cassandra container:

```
$ docker cp datasets/words.csv my-cass:/
```

Create Cassandra keyspace and table:

```
$ docker exec my-cass cqlsh -e "CREATE KEYSPACE IF NOT EXISTS my_app WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1};"

$ docker exec my-cass cqlsh -e "CREATE TABLE IF NOT EXISTS my_app.words (position int PRIMARY KEY, words text);"
```

Add data to the table:

```
$ docker exec my-cass cqlsh -e "COPY my_app.words (position, words) FROM '/words.csv' WITH HEADER = TRUE;"
```

To simplify, you can also put these 3 instructions inside a .cql file:

```
$ docker cp datasets/words.csv my-cass:/
$ docker cp datasets/insert_words.cql my-cass:/
$ docker exec my-cass cqlsh -f /insert_words.cql
```

Execute the WordCount app with Hecuba and PyCOMPSs:

```
$ scripts/user/runcompss-docker --w=1 --s='192.168.99.100:2376' --i='bscdatadriven/hecuba-compss-quickstart:latest' --stack=teststack --context-dir='/root/code' --classpath=/root/conf/*.jar --storage_conf=/root/conf/multinode.txt /root/code/WordCountExample.py
```

runcompss-docker allows to choose the parameters of the app deployment. Usage:

```
scripts/user/runcompss-docker
        --worker-containers=N
        --swarm-manager='<ip>:<port>'
        --image-name="DOCKERHUB_USER/IMG-NAME"
        --stack=stack-name
                # [rest of classic runcompss args]
```

Example:

```
scripts/user/runcompss-docker
        --worker-containers=5
        --image-name='bscdatadriven/hecuba-compss-quickstart:latest'
        --stack=teststack
        --context-dir='/root/code'
        --swarm-manager='192.168.99.100:2376'
        --classpath=--classpath=/root/conf/*.jar # Here begin classic runcompss arguments...
        --storage_conf=/root/conf/multinode.txt
        -d
        /root/code/WordCountExample.py


# MANDATORY ARGUMENTS:
   --w, --worker-containers:  # Specify the number of worker containers the app will execute on.
                        # One more container will be created to host the master.
                        # Example: --worker-containers=2
   --i, --image-name:       # Specify the image name of the application image in Dockerhub. Remember you must generate this with runcompss-docker-gen-image.
                        # Remember as well that the format must be: "DOCKERHUB_USERNAME/APP_IMAGE_NAME:TAG" (the :TAG is optional).
                        # Example: --image-name='john123/my-compss-application:1.9'
   --s, --swarm-manager:     # Specify the swarm manager ip and port (format:  <ip>:<port>).
                        # Example: --swarm-manager='129.114.108.8:4000'
```

```
--c, --context-dir:       # Specify the absolute application context directory inside the image.
                          # When using an application image, its provider must give you this information.
                          # Example: --swarm-manager='129.114.108.8:4000'
# OPTIONAL ARGUMENTS:
--stack:                  # Specify the name of the stack that will be deployed.
                          # Example: --stack=teststack
--c-cpu-units:            # Specify the number of cpu units used by each container (default value is 4).
                          # Example: --c-cpu-units=16
--c-memory:               # Specify the physical memory used by each container in GB (default value is 8 GB).
                          # Example: --c-memory=32  # (each container will use 32 GB)
--vm-creation-time:       # Time required to create a docker container on cloud (default: 60 sec)
                          # Example: --vm-creation-time=12
--min-vms:                # Minimum number of docker containers to run on cloud
--max-vms:                # Maximum number of docker containers to run on cloud
```

To finish, we can stop the Cassandra container with:

```
$ docker stop my-cass
```

Already created swarm and Cassandra containers

If a swarm is already created, you can start the virtual machines without creating them:

```
$ docker-machine start myvm1
$ docker-machine start myvm2
$ eval $(docker-machine env myvm1)
```

Then we start the Cassandra container:

```
$ docker start my-cass
```

Now we can execute the app *(using --s='\<ip-swarm-manager>:2376')*:

```
$ scripts/user/runcompss-docker --w=1 --s='192.168.99.100:2376' --i='bscdatadriven/hecuba-compss-quickstart:latest'
--stack=teststack --context-dir='/root/code' --classpath=/root/conf/*.jar --storage_conf=/root/conf/multinode.txt
/root/code/WordCountExample.py
```

Execute your own app

To execute an application that you yourself have implemented, first you have to copy the code to the image. There are several ways to do this, for example running a container in interactive mode, copying the code and then saving the image.

Run the container in interactive mode:

```
$ docker run -it bscdatadriven/hecuba-compss-quickstart:latest bash
```

Then, in another terminal, first copy the container id of the running container:

```
$ docker ps
CONTAINER ID        IMAGE                                          COMMAND                  CREATED             STATUS
PORTS                                      NAMES
695459a6e169        bscdatadriven/hecuba-compss-quickstart:latest  "bash"                   4 seconds ago       Up 4 seconds
22/tcp, 80/tcp                             vibrant_greider
cf23231bc034        cassandra                                      "docker-entrypoint.s…"   5 hours ago         Up 5 hours          7000-
7001/tcp, 7199/tcp, 9042/tcp, 9160/tcp     my-cass
```

Now we have to copy the files to the container *using the container id*:

```
$ docker cp myfile.py 695459a6e169:/path/to/code/.
```

Then we save the image in a new dockerhub repository:

```
$ docker commit 695459a6e169 mydockerhubuser/my-image:1.0
$ docker push mydockerhubuser/my-image:1.0
```

To finish, we can execute the app as follows (*using --s='\<ip-swarm-manager>:2376')*:

```
$ scripts/user/runcompss-docker --w=1 --s='192.168.99.100:2376' --i='mydockerhubuser/my-image:1.0' --
stack=teststack --context-dir='/path/to/code/' --classpath=/root/conf/*.jar --storage_conf=/root/conf/multinode.txt
/path/to/code/myfile.py
```

5 Conclusions and future steps

This report summarizes the advances that we have made in WP3 from month 12th to month 18th, related to both machine learning algorithms and technological components of the I-BiDaaS platform. We have added and evaluated new machine learning algorithms to the pool of the algorithms provided by the I-BiDaaS platform. We have also analyzed the convenience of including dislib as part of the platform, detecting the changes required to integrate this library as part of the software stack.

As part of the future work of this work package we include the following tasks:

- Continue adding new machine learning algorithms to the pool of the I-BiDaaS platform.
- Complete the integration of dislib with Hecuba and Qbeast.
- Complete the implementation of the I-BiDaaS Batch use cases, using the implementation of the machine learning algorithms of the pool.
- Complete the integration of the TDF tool with Cassandra.
- Complete the integration with the streaming module of the I-BiDaaS platform
- Complete the integration with the user interface and with the visualization tools of I-BiDaaS.

References

[1] PyCOMPSs: Parallel computational workflows in Python, Enric Tejedor, Yolanda Becerra, Guillem Alomar, Anna Queralt, Rosa M. Badia, Jordi Torres, Toni Cortes, Jesús Labarta, IJHPCA 31(1): 66-82 (2017), DOI: 10.1177/1094342015594678

[2] COMP Superscalar, an interoperable programming framework, Badia, R. M., J. Conejero, C. Diaz, J. Ejarque, D. Lezzi, F. Lordan, C. Ramon-Cortes, and R. Sirvent, SoftwareX, Volumes 3–4, December 2015, Pages 32–36, DOI: 10.1016/j.softx.2015.10.004

[3] ParaView + Alya + D8tree: Integrating High Performance Computing and High Performance Data Analytics. Antoni ARtigas, Cesare Cugnasco, Yolanda Becerra, Fernando M. Cucchietti, Guillaume Houzeaux, Mariano Vázquez, Jordi Torres, Eduard Ayguadé, Jesús Labarta. ICCS 2017: 465-474

[4] Cesare Cugnasco, Yolanda Becerra, Jordi Torres, Eduard Ayguadé: D8-tree: a de-normalized approach for multidimensional data analysis on key-value databases. ICDCN 2016: 18:1-18:10

[5] Knowledge repository of I-BIDaaS project, https://github.com/ibidaas/knowledge_repository

[6] ADDM implementation by UNSPMF. https://github.com/ibidaas/knowledge_repository/tree/master/tools_technologies/sources/batch_processing/unspmf

[7] Dislib, https://github.com/bsc-wdc/dislib

[8] Hecuba, https://github.com/bsc-dd/hecuba

[9] A Variation Coefficient Similarity Measure and Its Application in Emergency Group Decision-making. Xuanhua Xu, Liyuan Zhang and Qifeng Wan. Systems Engineering Procedia (5); 119 – 124. 2012

[10] Survey on Anomaly Detection using Data Mining Techniques. Shikha Agrawal, Jitendra Agrawal. Procedia Computer Science (60): 708-713. 2015.

[11] DataRobot. https://www.datarobot.com/

[12] Choosing the right encoding method-label vs. OneHot Encoder. Towards Data Science. https://towardsdatascience.com/choosing-the-right-encoding-method-label-vs-onehot-encoder-a4434493149b. 2018

[13] Create high-quality test data while minimizing the risks of using sensitive production data. IBM InfoSphere Optim Test Data Fabrication, IBM, 2017, https://www.ibm.com/il-en/marketplace/infosphere-optim-test-data-fabrication.

[14] Test Data Fabrication. Security and Data Fabrication, IBM Research, 2011, https://www.research.ibm.com/haifa/dept/vst/eqt_tdf.shtml.

[15] L.J.P. van der Maaten and G.E. Hinton. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research* 9(Nov):2579-2605, 2008

[16] Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers, S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, Foundations and Trends in Machine Learning, 3(1):1–122, 2011. (Original draft posted November 2010.)

[17] Task-based programming in COMPSs to converge from HPC to big data . Javier Conejero, Sandra Corella, Rosa M Badia, and Jesus Labarta, , The International Journal of High Performance Computing Applications 1–16, 2017

[18] CVXPY: A Python-Embedded Modeling Language for Convex Optimization, S. Diamond and S. Boyd, Journal of Machine Learning Research, 17(83):1-5, 2016.

[19] Sahu, A. & Jakovetić, D. & Bajović, D & Kar, S. (2018). Communication-Efficient Distributed Strongly Convex Stochastic Optimization: Non-Asymptotic Rate

[20] Sahu, A. & Jakovetić, D. & Bajović, D & Kar, S. Communication efficient distributed weighted non-linear least squares estimation, EURASIP Journal on Advances in Signal Processing

[21] Convergence rates for distributed stochastic optimization over random networks, A. Kumar Sahu, D. Jakovetic, D. Bajovic, S. Kar, to appear in proc. IEEE International Conference on Decision and Control, CDC 2018, invited paper, Dec 17-19, Miami, FL, USA

[22] Non-asymptotic rates for communication efficient distributed zeroth order strongly convex optimization, A. Kumar Sahu, D. Jakovetic, D. Bajovic, S. Kar, to appear in proc. GlobalSIP 2018, IEEE Global conference on signal and information processing, Nov 26-28, Anaheim, CA, USA

[23] Sahu, A. & Jakovetić, D. & Bajović, D & Kar, S. Communication efficient distributed weighted non-linear least squares estimation, EURASIP Journal on Advances in Signal Processing

[24] Armacki, A. & Jakovetić, D. & Krejić, N. & Krklec Jerinkić, N. (2019). Distributed Trust-Region Method With First Order Models

[25] Arik Friedman and Assaf Schuster. Data mining with differential privacy. In International Conference on Knowledge Discovery and Data Mining, pages 493–502, 2010.

[26] Xueyang Hu et al. Differential Privacy in Telco Big Data Platform. Proceedings of the VLDB Endowment, Vol. 8, No. 12 Copyright 2015 VLDB Endowment 2150-8097/15/08.

[27] Scikit-learn: machine learning in python. https://scikit-learn.org/stable/

[28] Cassandra Query Language (CQL). https://docs.datastax.com/en/dse/6.7/cql/index.html

[29] OpenStreetMap dataset. https://wiki.openstreetmap.org/wiki/Downloading_data

[30] Docker documentation. https://docs.docker.com/

[31] Docker swarm. https://docs.docker.com/get-started/part4/

[32] Quick-start for the batch-module of I-BiDaaS. https://github.com/bsc-dd/knowledge_repository/tree/quick-start/quick-start

[33] Sam Fletcher and Md Zahidul Islam. Decision Tree Classification with Differential Privacy: A Survey. CoRR. 2016. http://arxiv.org/abs/1611.01919