Big Data to Enable Global Disruption of the Grapevine-powered Industries

D7.2 - Experimental Report on Current Datasets

DELIVERABLE NUMBER	D7.2
DELIVERABLE TITLE	Experimental report on Current Datasets
RESPONSIBLE AUTHOR	Giannis Stoitsis (Agroknow)

Co-funded by the Horizon 2020

Framework Programme of the European Union

GRANT AGREEMENT N.	780751
PROJECT ACRONYM	BigDataGrapes
PROJECT FULL NAME	Big Data to Enable Global Disruption of the Grapevine-powered industries
STARTING DATE (DUR.)	01/01/2018 (36 months)
ENDING DATE	31/12/2020
PROJECT WEBSITE	http://www.bigdatagrapes.eu/
COORDINATOR	Nikos Manouselis
ADDRESS	110 Pentelis Str., Marousi, GR15126, Greece
REPLY TO	nikosm@agkroknow.com
PHONE	+30 210 6897 905
EU PROJECT OFFICER	Mrs. Annamaria Nagy
WORKPACKAGE N. | TITLE	WP7 | Cross-sector Rigorous Experimental Testing
WORKPACKAGE LEADER	CNR
DELIVERABLE N. | TITLE	D7.2 | Experimental report on Current Datasets
RESPONSIBLE AUTHOR	Giannis Stoitsis (Agroknow)
REPLY TO	stoitsis @agroknow.com
DOCUMENT URL	http://www.bigdatagrapes.eu/
DATE OF DELIVERY (CONTRACTUAL)	30 June 2019 (M18), 30 June 2020 (M30)
DATE OF DELIVERY (SUBMITTED)	28 June 2019 (M18), 31 July (M31), 31 Dec 2020 (M36)
VERSION | STATUS	2.1 | Final
NATURE	RE (REPORT)
DISSEMINATION LEVEL	PU (Public)
AUTHORS (PARTNER)	Ioanna Polychronou (Agroknow), Mihalis Papakonstadinou (Agroknow), Giannis Stoitsis (Agroknow), Panagiotis Rousis (Agroknow), Timotheos Lanitis (Agroknow)
CONTRIBUTORS	Salvatore Trani (CNR), Vinicius Monteiro de Lira (CNR)
REVIEWER	Franco Maria Nardini (CNR)

VERSION	MODIFICATION(S)	DATE	AUTHOR(S)
0.1	Table of Contents	02/06/2019	Panagiotis Zervas (Agroknow), Mihalis Papakonstadinou (Agroknow)
0.4	Initial version	04/06/2019	Panagiotis Zervas (Agroknow), Mihalis Papakonstadinou (Agroknow)
0.6	Input from partners	06/06/2019	Franco Maria Nardini (CNR), Todor Primov (CNR), Nikola Rusinov (ONTOTEXT)
0.7	Internal review	17/06/2019	Franco Maria Nardini (CNR)
0.9	Partners review, final comments and edits	24/6/2019	Franco Maria Nardini (CNR), Todor Primov (CNR), Nikola Rusinov (ONTOTEXT)
1.0	Final version	18/06/2018	Panagiotis Zervas (Agroknow), Mihalis Papakonstadinou (Agroknow)
1.1	Initial second version	15/05/2020	Ioanna Polychronou (Agroknow), Mihalis Papakonstadinou (Agroknow), Panagiotis Rousis (Agroknow), Timotheos Lanitis (Agroknow)
1.4	Input from Partners	25/06/2020	Vinicius Monteiro de Lira (CNR), Salvatore Trani (CNR)
1.6	Internal review, final comments	28/07/2020	Ioanna Polychronou (Agroknow), Giannis Stoitsis (Agroknow), Franco Maria Nardini (CNR)
2.0	Final second version	31/07/2020	Ioanna Polychronou (Agroknow), Panagiotis Rousis (Agroknow), Timotheos Lanitis (Agroknow)
2.1	Final version	15/01/2020	Giannis Stoitsis (Agroknow), Mihalis Papakonstadinou (Agroknow)

PARTICIPANTS		CONTACT
Agroknow IKE (Agroknow, Greece)		Nikos Manouselis Email: nikosm@agroknow.com
SIRMA AI (SAI, Bulgaria)		Todor Primov Email: todor.primov@ontotext.com
Consiglio Nazionale DelleRicherche (CNR, Italy)		Raffaele Perego Email: raffaele.perego@isti.cnr.it
Katholieke Universiteit Leuven (KULeuven, Belgium)		Katrien Verbert Email: katrien.verbert@cs.kuleuven.be
Geocledian GmbH (GEOCLEDIAN, Germany)		Stefan Scherer Email: stefan.scherer@geocledian.com
Institut National de la Recherché Agronomique (INRA, France)		Pascal Neveu Email: pascal.neveu@inra.fr
Agricultural University of Athens (AUA, Greece)		Katerina Biniari Email: kbiniari@aua.gr
Abaco SpA (ABACO, Italy)		Simone Parisi Email: s.parisi@abacogroup.eu
SYMBEEOSIS EY ZHN S.A. (Symbeeosis, Greece)		Konstantinos Rodopoulos Email: rodopoulos-k@symbeeosis.com

ACRONYMS LIST

BDG	BigDataGrapes
DSPS	Distributed Stream Processing Systems
IoT	Internet of Things
LDBC	Linked Data Benchmark Council
MIPS	Million instructions per second
MFLOPS	Million floating-point operations per second
MUDD	Multi-dimensional data generator
OWL	Ontology Web Language
RDF	Resource Description Framework
SPARQL	Symantec Protocol and RDF Query Language
SNB	Social Network Benchmark
SPB	Semantic Publishing Benchmark
SDPS	Streaming Data Processing Systems
TPC	Transaction Processing Performance Council

EXECUTIVE SUMMARY

The deliverable D7.2 “Experimental Report on Current Datasets” consists of a report describing the outcomes of the experimentation performed on the datasets provided by each pilot, utilizing the Big Data Grapes stack.

This report starts by outlining the methodology and metrics we employ for our experimentation. We focus our experimentation on a pilot level, by initially analysing the provided datasets and evaluating them against the Big Data Vs, as suggested and described in detail in D7.1.

We move on to identify and document the data flows each pilot adheres to throughout the Big Data Grapes stack. For each of them, we highlight the frameworks and the components involved in the process.

We experiment the identified data flow steps by describing, for each of them, three usage scenarios. During the execution of each scenario, for each step we monitor the chosen performance metrics, by showing the respective diagrams and by analysing the outcomes.

This deliverable also presents an end to end report for each of the pilot’s data flows, identifying bottlenecks and making suggestions to improve the performance of the BigDataGrapes architecture where needed.

TABLE OF CONTENTS

1 INTRODUCTION 10

2 METHODOLOGY 11

2.1 STEPS 11

2.2 METRICS 11

3 RIGOROUS TESTING EXPERIMENTS 12

3.1 FARM MANAGEMENT PILOT 12

3.1.1 Dataset Analysis & Evaluation 12

3.1.2 Data Flows 14

3.1.3 Experimentation 15

3.1.4 End to End Report 21

3.2 NATURAL COSMETICS PILOT 22

3.2.1 Dataset Analysis & Evaluation 22

3.2.2 Data Flows 23

3.2.3 Experimentation 24

3.2.4 End to End Report 30

3.3 TABLE AND WINE GRAPES PILOT 31

3.3.1 Dataset Analysis & Evaluation 31

3.3.2 Data Flows 37

3.3.3 Experimentation 38

3.3.4 End to End Report 44

3.4 WINE MAKING PILOT 45

3.4.1 Dataset Analysis & Evaluation 45

3.4.2 Data Flows 49

3.4.3 Experimentation 49

3.4.4 End to End Report 54

3.5 FOOD PROTECTION PILOT 55

3.5.1 Dataset Analysis & Evaluation 55

3.5.2 Data Flows 57

3.5.3 Experimentation 57

3.4.4 End to End Report 64

4 Experimentation workflow and panel for machine learning solutions 65

4.1 EXPERIMENTATION STEPS OF THE INTELLIGENT PANEL 65

4.2 CUSTOM DATASET UPLOAD 65

4.3 MACHINE LEARNING MODELS 66

4.4 MONITORING AND MODEL SAVING 66

4.5 DEPLOYING MODELS ON THE BIG DATA PLATFORM 67

5 CONCLUSIONS 68

LIST OF FIGURES

Figure 1 Abstract overview of sensor data ingestion 15
Figure 2 Completion time of sensor data ingestion 16
Figure 3 CPU usage during sensor data ingestion 16
Figure 4 Network usage during sensor data ingestion 16
Figure 5 Memory usage during sensor data ingestion 16
Figure 6 Completion time of satellite image processing data ingestion 17
Figure 7 CPU usage during satellite image processing data ingestion 18
Figure 8 Network usage during satellite image processing data ingestion 18
Figure 9 Memory usage during satellite image processing data ingestion 18
Figure 10 Completion time of semantically enriched data ingestion 19
Figure 11 CPU usage during ingestion of semantically enriched data 19
Figure 12 Network usage during ingestion of semantically enriched data 19
Figure 13 Memory usage during ingestion of semantically enriched data 19
Figure 14 CPU usage during water stress prediction 20
Figure 15 Memory usage during water stress prediction 20
Figure 16 Completion time of dataset upload step 24
Figure 17 CPU usage during dataset upload 24
Figure 18 Network usage during dataset upload 25
Figure 19 Memory usage during dataset upload 25
Figure 20 Completion time of enriched data ingestion 26
Figure 21 CPU usage during ingestion of enriched data 26
Figure 22 Network usage during enriched data ingestion 26
Figure 23 : Memory usage during enriched data ingestion 26
Figure 24 Completion time of satellite image processing data ingestion 27
Figure 25 CPU usage during satellite image processing data ingestion 27
Figure 26 Network usage during satellite image processing data ingestion 28
Figure 27 Memory usage during ingestion of satellite image processing data 28
Figure 28 Completion time of data correlation step 29
Figure 29 CPU usage during correlation of data 29
Figure 30 Network usage during correlation of data 29
Figure 31 Memory usage during correlation of data 29
Figure 32 Completion time for dataset upload 38
Figure 33 CPU usage during dataset upload 38
Figure 34 Network usage during dataset upload 39
Figure 35 Memory usage during dataset upload 39
Figure 36 Completion time for data streams pipeline 40
Figure 37 CPU usage during data streams pipeline 40
Figure 38 Network usage during data streams pipeline 40
Figure 39 Memory usage during data streams pipeline 40
Figure 40 Completion time for ingestion of enriched data 42
Figure 41 CPU usage during ingestion of enriched data 42
Figure 42 Network usage during ingestion of enriched data 42
Figure 43 Memory usage during ingestion of enriched data 42
Figure 44 CPU usage during correlation API call 43
Figure 45 Memory usage during correlation API call 44
Figure 46 Completion time for dataset upload 50
Figure 47 CPU usage during dataset upload 50
Figure 48 Network usage during dataset upload 50
Figure 49 : Memory usage during dataset upload 50
Figure 50 Completion time for the ingestion of semantically enriched data 51
Figure 51 CPU usage during the ingestion of semantically enriched data 52

Figure 52 Network usage during the ingestion of semantically enriched data 52
Figure 53 Memory usage during ingestion of semantically enriched data 52
Figure 54 Completion time of leaves prediction 53
Figure 55 CPU usage during prediction of leaves 53
Figure 56 Memory usage during prediction of leaves 53
Figure 57 GPU usage during prediction of leaves 54
Figure 58 Completion time for dataset upload 58
Figure 59 CPU usage during dataset upload 58
Figure 60 Memory usage during dataset upload 59
Figure 61 Network usage during dataset upload 59
Figure 62 CPU usage during price prediction 60
Figure 63 Memory usage during price prediction 60
Figure 64 CPU usage during using recall prediction API 61
Figure 65 Memory usage during recall prediction API 62
Figure 66 Completion time for risk assessment 63
Figure 67 CPU usage during using risk assessment 63
Figure 68 Memory usage during risk assessment 63
Figure 69 Network usage during risk assessment 64
Figure 70: Intelligent panel of the BigDataPlatform 65
Figure 71: Comparing different models based on their performance 67
Figure 72: Save and register the prediction model to the Big Data Platform 67

LIST OF TABLES

Table 1: Sentinel-2 satellite images dataset 12
Table 2 : Landsat-8 satellite images dataset 12
Table 3 Satellite image processing dataset 13
Table 4 Environmental & Field indicators dataset 14
Table 5 Laboratory tests dataset 22
Table 6 Field metadata & Geographical information dataset 23
Table 7 Geographical & Field metadata dataset 31
Table 8 Soil quality dataset 31
Table 9 Vegetation indices estimation dataset 32
Table 10 Vegentation indices dataset 33
Table 11 Radioactive transfer & biophysical characteristics dataset 33
Table 12 Laboratory tests dataset 34
Table 13 IoT data 35
Table 14 Yield data dataset 35
Table 15 Qualitative data dataset 36
Table 16 Photosynthesis data dataset 36
Table 17 Experimentation results of rdfization & semantic enrichment 41
Table 18 Genetic data dataset 45
Table 19 Soil characteristics dataset 45
Table 20 Plot management dataset 46
Table 21 Climatic data dataset 47
Table 22 Laboratory analysis dataset 47
Table 23 Winemaking activities dataset 48
Table 24 Sensory analysis dataset 48
Table 25 Laboratory tests dataset 55
Table 26 Food price dataset 56
Table 27 Food incidents dataset 57
Table 28: Machine learning models available in the Intelligent panel 66
Table 28 Experimentation outcomes 68

1 INTRODUCTION

The BigDataGrapes (BDG) platform aims at targeting the needs of the grapevine industry using Big Data technologies, frameworks and components. In this report we perform an experimentation over the currently provided datasets by focusing on the needs of each use case as described in D2.1.

This report is structured as follows: first, we describe the methodology we use and the metrics we employ for the experimentation. Both of them are focused around the 4Vs of Big Data, as described in D7.1. We then present the description of each pilot's datasets and their respective data flows through the BDG stack. We perform this analysis by performing step by step and end to end tests for each dataset while monitoring the chosen metrics. Finally, we conclude this report by reporting the outcomes of this process, the identification of bottlenecks and heavy-duty tasks. We also provide suggestions on how to improve the performance of the whole BDG stack.

It should be noted all the experiments are performed on the final version of the BDG stack by using the deployed and usable components and tools. It is important to mention that the current version of the components is the final one. All of the Application Programming Interface (API) calls done throughout the experimentation support user authentication via API keys. The user authentication information is stored in a MySQL instance. All of the data flows are performed using Apache Nifi [1]and the data uploads are performed either through API endpoints or using Filebeat [2]from the Elastic stack, in case of internal data transfers.
The main outcome of this deliverable is that the current and final version of the BDG stack behaves efficiently in all experiments performed. Current data for the platform stress test were used for the experiments.

[1] https://nifi.apache.org/
[2] https://www.elastic.co/beats/filebeat

2 METHODOLOGY

In this section, we describe the methodology we use for the experimentation over the provided datasets, along with the steps we perform and the respective metrics we use for each of them.

As described in D7.1, we focus our experimentation around the 4Vs of Big Data: volume, velocity, variety and veracity. Based on the characteristics of each dataset, we choose to employ techniques and methods described in the bibliography (Deliverable D7.1, see BDGS and BigDataBench) to ensure the presence of every V for each of the datasets provided by the pilots. We focus our experimentation on the performance of the BDG stack by tracking several system indicators as described in section 2.2. Finally, we employ MetricBeat, i.e., the proposed tool for monitoring the scalability of BD platform reviewed in D7.1, over our Elastic stack to monitor and report our chosen metrics.

Since all of the BDG stack has been deployed in a microservice architecture, our step by step and end to end experimentation is done over the API endpoints provided by the platform. To make the experimentation easier we developed a python script that simulate these endpoint calls.

To showcase the potential of the deployed stack in terms of scalability, we perform the experimentation using three usage scenarios for each step of the process by gradually increasing the requests made towards the platform.

2.1 STEPS

We choose to perform the experimentation on a pilot level. We identified a set of steps we follow for each of them. First, we identify and abstractly describe the provided datasets for each pilot. Then we move on with evaluating them against the Vs of Big Data, generating where applicable synthetic data series to cover any of the 4 Vs that are not covered by the provided datasets. We then identify the data flow each dataset will follow in the BDG stack, denoting the steps of this flow. Finally, using a python script that will simulate bursts of this data flows throughout the BDG stack, we will monitor and report our chosen metrics for this benchmark in real time.

It should be noted that all of the experimentation is done using the actual provided data by each pilot and involve the integrated components of the BDG stack.

2.2 METRICS

As mentioned in the previous sections, we will perform step by step and end to end experimentation for each pilot. To track and report the performance of the stack we chose to monitor the following indicators:

- Completion time, both on a step by step level, as well as on the whole end to end data flow,
- CPU (central processing unit) usage, we will track the CPU usage by each of the components as they are triggered by the flow,
- Memory usage of each of the components and technologies,
- Network usage in terms of bytes, we employ this metric, since the whole stack is based on a microservice architecture.

3 Rigorous testing experiments

3.1 Farm management Pilot

The farm management pilot aims at developing a system that can support the farmer in his/her data collection, as well as in his/her day to day work. To that end, this specific pilot has granted access to datasets varying from satellite images and their respective processing to environmental and field indicators tracking by sensors deployed in the fields. In the following sections we analyze the provided datasets, following the methodologies described in D7.1. We identify the data flows of this specific use case inside the stack and experiment on each of them, using the metrics and the methodology described in the previous section.

3.1.1 Dataset Analysis & Evaluation

This pilot is characterized by datasets concerning satellite images from the external services of Sentinel-2 and Landsat-8 and the respective processing. All of this data is accessible through API endpoints provided by GEOCLIDEAN. It is also characterized by datasets concerning the environmental indicators and sensor data coming from deployed installations on the respective fields. In the tables below, we further describe these datasets and evaluate them against the 4Vs of Big Data.

Table 1: Sentinel-2 satellite images dataset

Dataset	Sentinel-2 Satellite Images	
Metadata/Description	This dataset contains the satellite images collected from Sentinel-2 for the fields provided by AUA, INRA, ABACO and Symbeeosis	
Provider	GEOCLIDEAN	
Sample	Sample of this data can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume this dataset has ~2K scenes with a total size of ~1.5TB of data which is an acceptable value
	Velocity	The provided dataset is accessed through a REST service and shows the desired velocity since it is generated and ingested in real-time
	Variety	In terms of variety the provided dataset shows one type of data
	Veracity	There is no need for synthetic data generation

Table 2 : Landsat-8 satellite images dataset

Dataset	Landsat-8 Satellite Images

Metadata/Description	This dataset contains the satellite images collected from Landsat-8 for the fields provided by AUA, INRA, ABACO and Symbeeosis	
Provider	GEOCLIDEAN	
Sample	Sample of this data can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume this dataset has ~1.2K scenes with a total size of ~1.2TB of data which is an acceptable value
	Velocity	The provided dataset is accessed through a REST service and shows the desired velocity since it is generated and ingested in real-time
	Variety	In terms of variety the provided dataset shows one type of data
	Veracity	There is no need for synthetic data generation

Table 3 Satellite image processing dataset

Dataset	Satellite Image Processing Dataset	
Metadata/Description	This dataset contains the satellite image processing outcomes for the provided fields	
Provider	GEOCLIDEAN	
Sample	Sample of this data can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume this dataset has ~100K of data points
	Velocity	The provided dataset is accessed through a REST service and shows the desired velocity since it is generated and ingested in real-time
	Variety	In terms of variety the provided dataset has 19 different data types along with metadata information & images, that are spread across 83 fields

	Veracity	Given the combination of the previous Vs, we have a moderate volume of data that shows high velocity, being produced and available in real-time and with an acceptable variety. Thus, we believe there is no need for synthetic data generation

Table 4 Environmental & Field indicators dataset

Dataset	Environmental & Field Indicators	
Metadata/Description	This dataset contains environmental & field indicators by sensors deployed in ABACO's field	
Provider	ABACO	
Sample	Sample of the data can be accessed here	
Evaluation	Big Data Vs	
	Volume	In terms of volume this dataset has ~230K data points
	Velocity	The provided dataset is accessed through a REST service and shows the desired velocity since it is generated and ingested in real-time
	Variety	In terms of variety the provided dataset has 35 different data types
	Veracity	There is no need for synthetic data generation

Concluding the dataset analysis for this specific pilot, we observe that we have high volume datasets, having ~300K records that require a total amount of storage of ~2.7TB that is constantly growing. The provided datasets are all available through REST services generated and ingested in real-time and cover 56 different data types. Based on these observations we do not believe that there is a need for synthetic data generation and will move on with the experimentation with the provided datasets.

3.1.2 Data Flows

In this section, we describe the steps that the provided datasets follow inside the BDG stack. As with all the use cases we will experiment on, we will focus on the ones that are already completed and integrated into the stack that serve this specific use case.

Initially, the data collected from the sensors deployed in the field are ingested into the platform. The next step involves the ingestion of the image processing data for each of the fields to be ingested. After this step, the data are converted into Resource Description Framework (RDF) and stored into the knowledge graph of the platform for their semantic enrichment. Finally, the data are extracted from the knowledge graph, and sent to the predictive analytics service for the analysis to happen. In its current state, this final step, utilizes several BDG components. More specifically it uses Apache Hadoop[3], Hue [4]and Spark[5]. However, the datasets it currently uses are not the ones provided by this pilot, leading us to exclude this step from our experimentation.

3.1.3 Experimentation

In this section we perform the experimentation for each of the identified steps, using the datasets provided for this use case. To that end we use the ~230K sensor data coming from the field and the ~100K of satellite image processing outcomes coming from the GEOCLIDEAN service. As described in the methodology section, we perform our experimentation using three scenarios for each step, to better showcase the scalability potential of the deployed stack.

3.1.3.1 Ingestion of Sensor Data

This step involves the ingestion of the sensor data, through the data stream component, Apache Kafka, of the BDG stack through a web service and stored into BDG's MongoDB for the metadata on the field, or initially into Apache Cassandra to take advantage of the high write throughput and then into Elasticsearch for the actual numerical values.

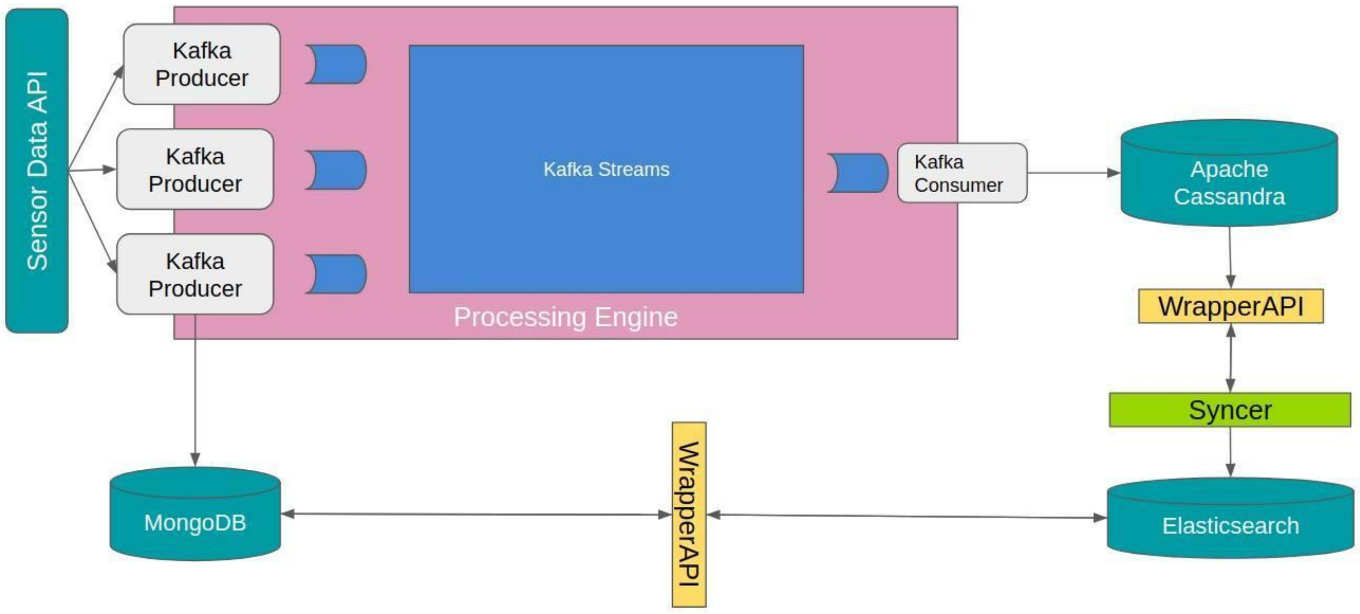


Figure 1 Abstract overview of sensor data ingestion

Figure 1 shows an overview of this specific process. As shown in the figure, the data is initially collected by Kafka producers and split between the metadata information, which is sent to MongoDB and actual data points that are inserted into Kafka topics to be processed by Kafka streams and collected by a Kafka consumer that sends the processed data to the Apache Cassandra. The last step of this step is the syncing between the data that reside in the Apache Cassandra instance and Elasticsearch.

To experiment on this step we identify the following usage scenarios, one with a Kafka setting of 1 producer/broker/consumer for all the data that reside into this dataset, one in which we will use a setting of 3

[3] https://hadoop.apache.org/
[4] https://gethue.com/
[5] https://spark.apache.org/

producers/brokers/consumers and a final in which we have a setting of 35, creating an 1-1 scenario for each data type to test the concurrency capabilities of the respective component.

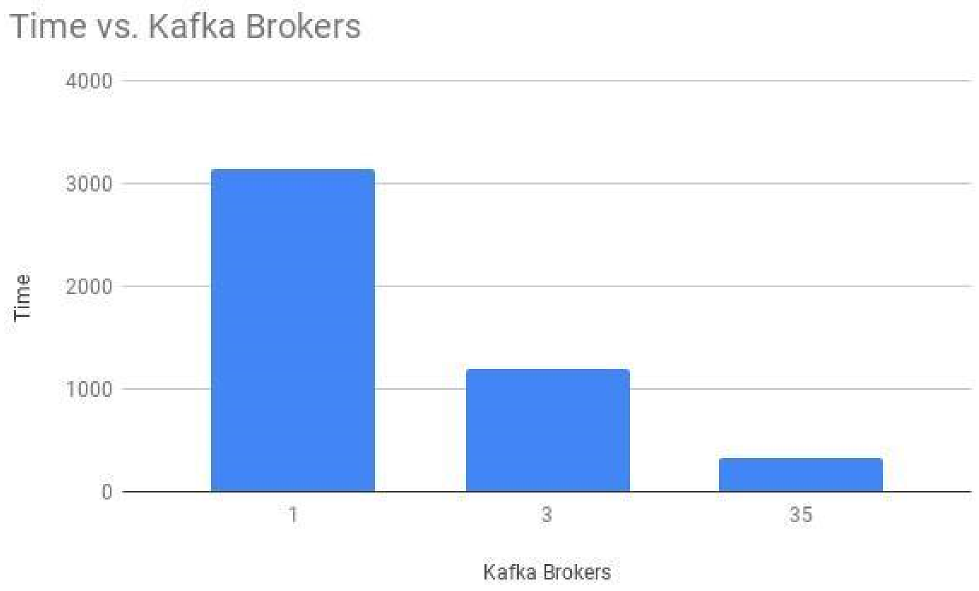


Figure 2 Completion time of sensor data ingestion

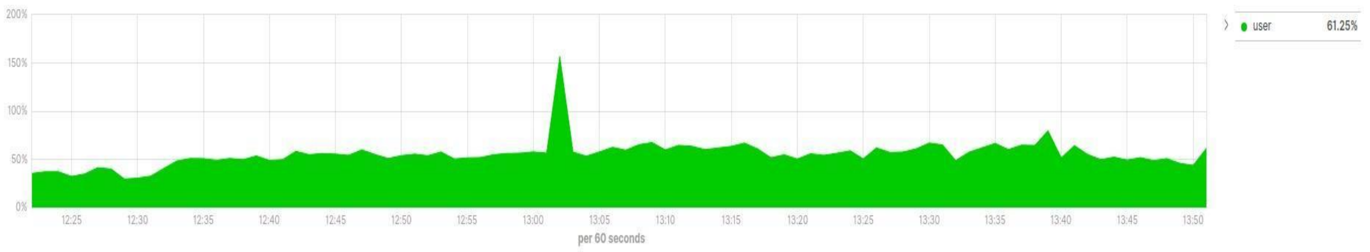


Figure 3 CPU usage during sensor data ingestion

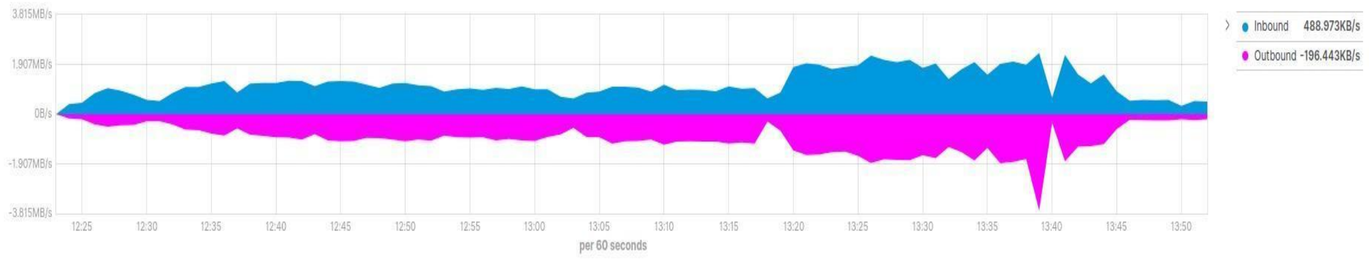


Figure 4 Network usage during sensor data ingestion

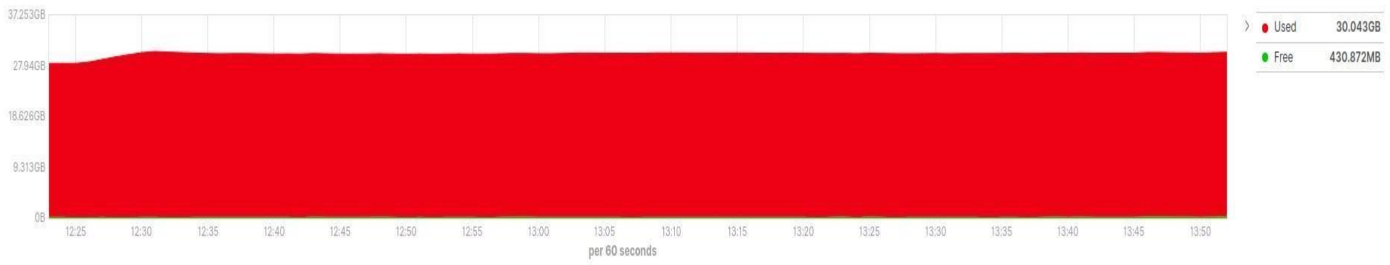


Figure 5 Memory usage during sensor data ingestion

As shown in the above figures, we observe the highest completion time (figure 2) for the ingestion scenario involving the setting of one producer/broker/consumer for Apache Kafka, taking 3,138 seconds to complete. During this scenario, we have an average of 50% CPU usage (figure 3) for the whole stack, with a peak of 150%. The network traffic during this scenario shows an average of 500KB/s of inbound traffic. In the next setting of 2 producer/broker/consumers we observe on average the same CPU usage, without a spike as with the previous one. Figure 4 shows a much higher network traffic, on average 1.9MB/s and the whole scenario takes 1,192 seconds to complete. In our final experiment, in which we have one dedicated producer, broker and consumer for each of the observed indicators, we have a much lower completion time, 328 seconds, and on average the same CPU usage as in the previous scenario, 50%. In terms of network traffic, we have a small increase in comparison to the previous one, leading to an inbound traffic of 2MB/s. Throughout all scenarios, we observe the same memory usage of the whole system with very little increases, easily explained by the fact that all of the involved components are Java based, set to reserve the required memory upon startup.

Following our experimentation, we conclude that the setting of 1 producer/broker/consumer for all of the data, is ineffective for this amount of data, since it takes a lot of time to be completed. We consider the final usage scenario as the best one with dedicated Apache Kafka pipelines for each indicator, since it takes little time to be completed without showing higher values in network traffic and CPU, as compared with the second one.

3.1.3.2 Ingestion of Satellite Image Processing Data

This step concerns the harvesting and storage of the satellite image processing dataset for each field through GEOCLIDEAN's service. In the following use cases, we experiment on this specifically for each pilot's field, so we chose for this step to experiment on the overall ingestion process of this service, involving all of the 100K data points. As with all the steps, we perform our experimentation with three different usage scenarios: in the first we will ingest 50% of the data sequentially, in the second all of the data are ingested in a sequential way, and finally we employ a scenario in which all of the data is ingested concurrently using 20 workers, each one responsible for the ingestion of the data for the two fields.

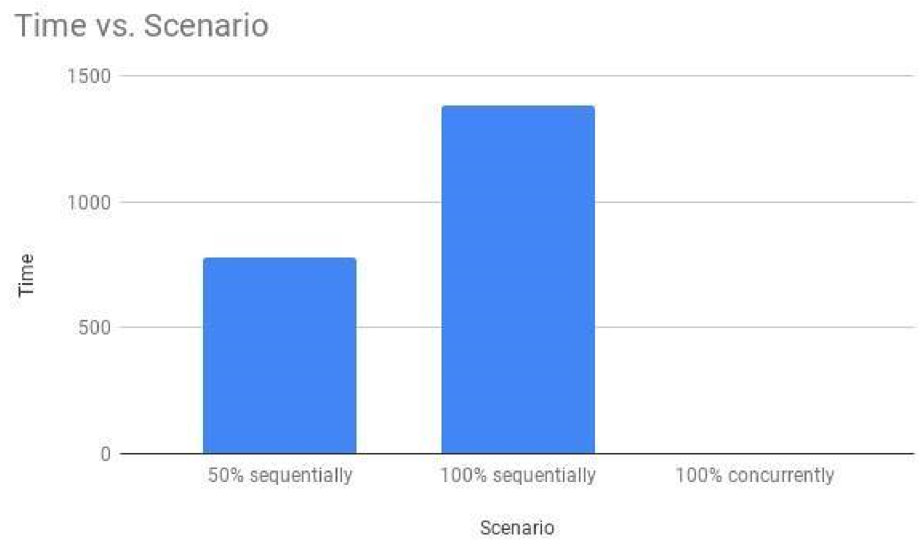


Figure 6 Completion time of satellite image processing data ingestion

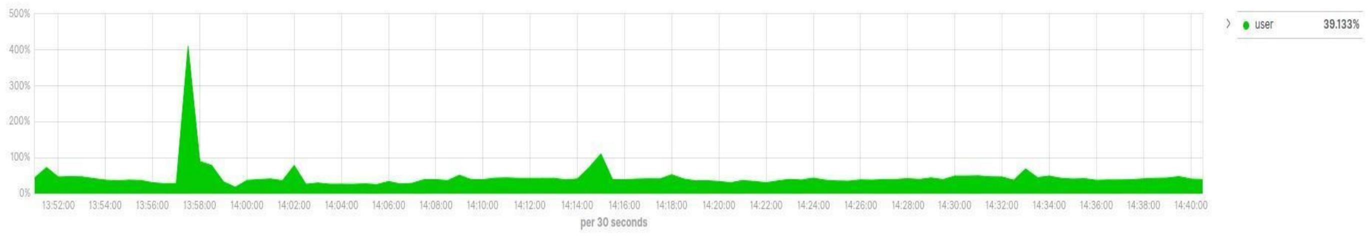


Figure 7 CPU usage during satellite image processing data ingestion

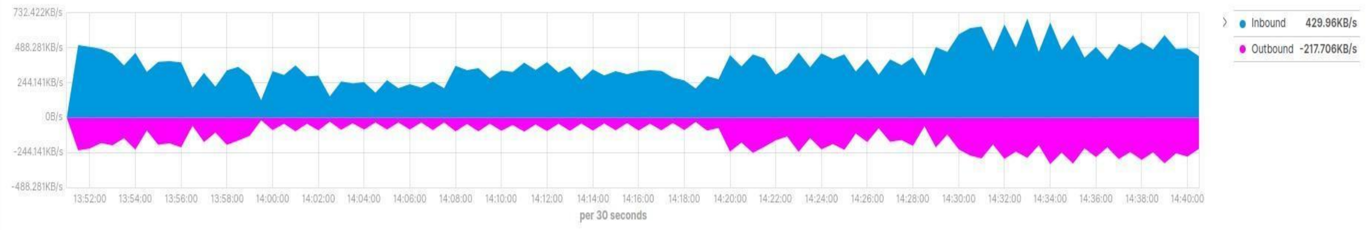


Figure 8 Network usage during satellite image processing data ingestion

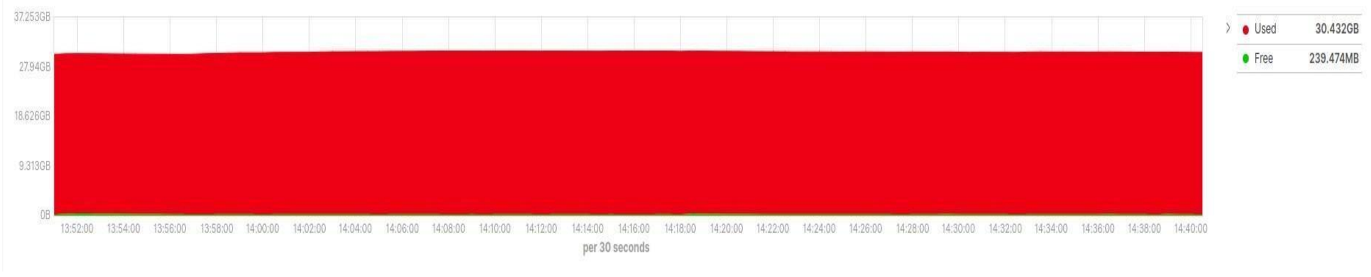


Figure 9 Memory usage during satellite image processing data ingestion

The figures illustrated above present the completion time, CPU usage, network traffic and memory usage of the whole stack, throughout our experimentation for this specific step. As expected, we observe the highest completion time (figure 6) for the second scenario where all of the data is ingested sequentially and a very high decrease in terms of completion time for the final one where 20 concurrent workers are employed. More specifically, 780 seconds are required for the first scenario to be completed, 1,380 for the second and only 3.2 for the final one. In terms of CPU usage (figure 7), each scenario shows on average the same usage, ~40-50%, with a spike of 400% for the first one and a spike of 100% for the second. The network traffic (figure 8) shows a steady increase for each step, starting at 300-350KB/s for the first, 450-500KB/s for the second and ~700KB/s for the final one. As in the previous step, the memory usage (figure 9) remains on the same levels.

Concluding our experimentation for this step, we observed that the best usage scenario is the third one, where 20 concurrent workers are employed. This leads us to believe that the best way to involve the ingestion of satellite images for the BDG stack is through frequent offline tasks with a high number of workers deployed in parallel.

3.1.3.3 Rdfization & Semantic Enrichment

In section 3.3.3.3 we thoroughly experiment on this step with higher volume datasets so do not conduct the same experiment here with the provided datasets.

3.1.3.4 Ingestion of Semantically Enriched Data

In this step we extract the semantically enriched data from GraphDB and store it in BDG's Elasticsearch instance. The three scenarios we use for our experimentation are the following: in one we consider the ingestion of 1 day's data, ~1,000 data points, in the next one month's data, which is approximately 35K data points and in the final one we will attempt to extract and store all of the 330K of data points provided. All of the experiments will be performed using three concurrent processes and in batches of 1,000 objects per request.

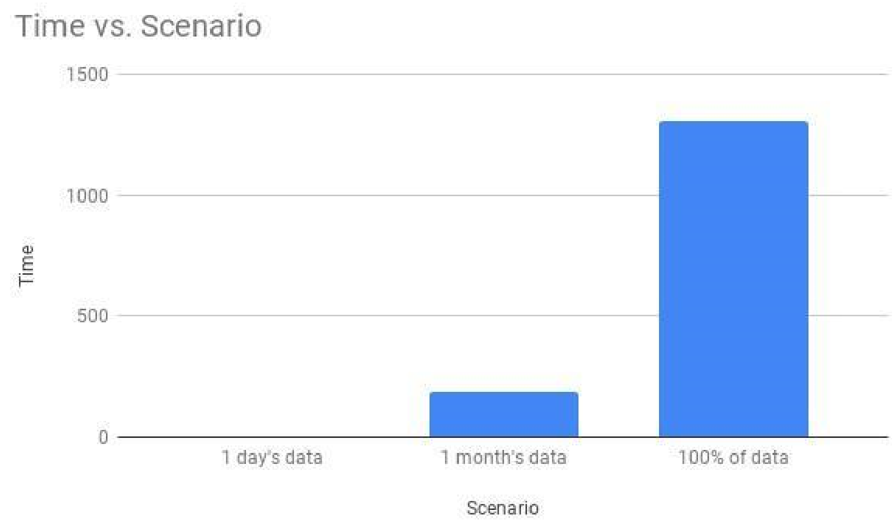


Figure 10 Completion time of semantically enriched data ingestion

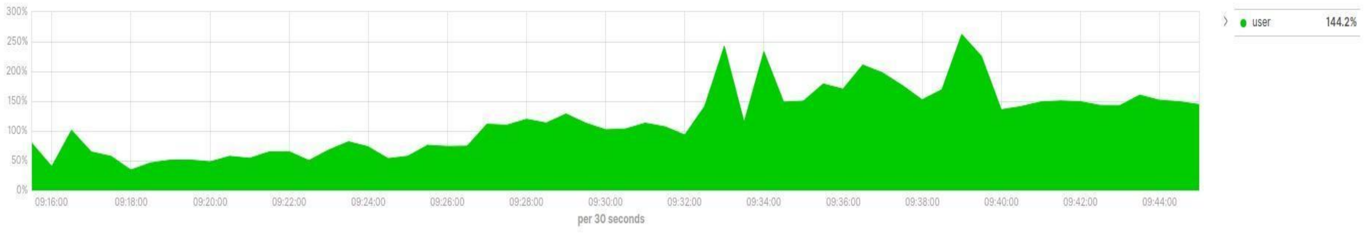


Figure 11 CPU usage during ingestion of semantically enriched data

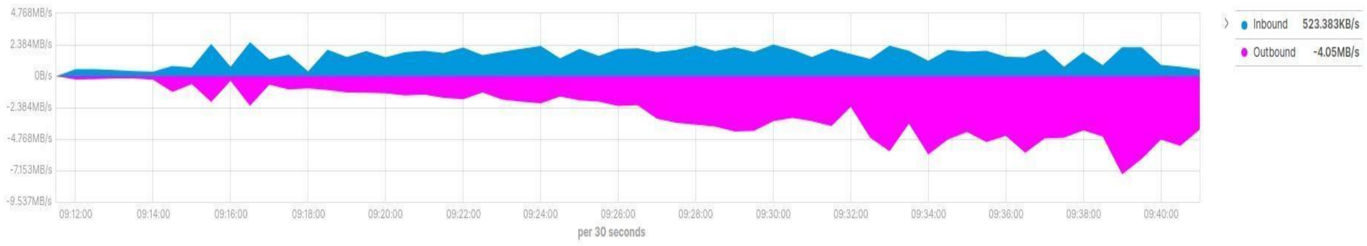


Figure 12 Network usage during ingestion of semantically enriched data

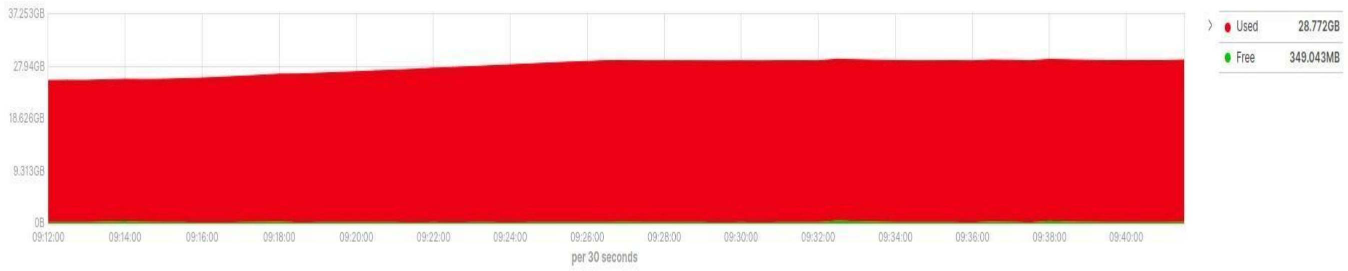


Figure 13 Memory usage during ingestion of semantically enriched data

Figure 10 illustrates the completion time for all of the chosen scenarios for this step. As we can see the ingestion of 1 day's data takes just under 2 seconds to complete, while 184 seconds are required in order to ingest 1 month's data. The ingestion of all of the data takes the longest as it takes 1,307 seconds to complete. In terms of CPU (figure 11) the first two scenarios show on average the same usage (roughly 60-70%), while the last one consumes on average 100% of CPU with a peak of 260%. The network traffic (figure 12) shows the same behavior for the last two scenarios with an average of 2-2.5MB/s while the first one has under 1MB/s, more specifically it consumes 800KB/s. Finally, the memory usage (figure 13) remains around the same levels throughout our experimentation.

As presented by our analysis for this step, we consider the best usage scenario to be the real-life one involving the ingestion of 1 day's data. To that end and taking into account the high network usage of the other two scenarios, we believe that the highest performance for this step can be achieved with the frequent extraction of smaller batches of data, happening on the background of the whole platform.

3.1.3.5 Water stress prediction

The machine learning step related to this task concerns the development of a solution for water stress prediction, a task that allows to monitor the amount of water that is easily usable by the vine, by using meteorological data from weather stations and soil data. For the purpose of this test, we used the dataset related to the "Casato Prime Donne" field, with similar results observed from the "Il Palazzo" field. We identify the following usage scenario for this step: in the first one we compute the water stress prediction for only 10% of the observation of the dataset, i.e., 37 observations. The second scenario on the other hand makes usage of the full dataset, i.e., all the 373 observations. Given the prediction API is very fast to answer, we simulated five minutes of sequential API call with the first scenario and five minutes of sequential API with the second scenario.

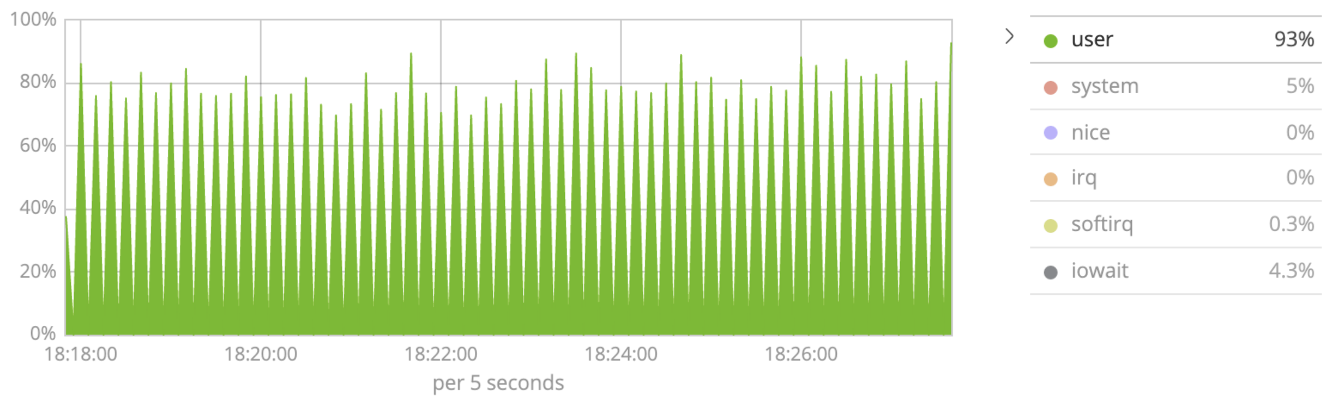


Figure 14 CPU usage during water stress prediction

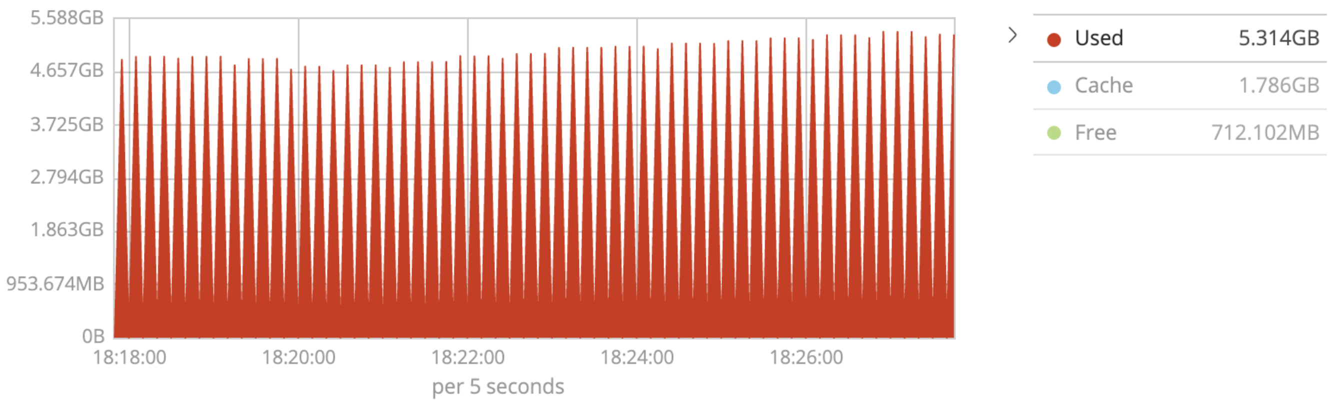


Figure 15 Memory usage during water stress prediction

According to the figures above, the CPU usage is almost constant disregarding the number of observations used for the water stress prediction task, i. e., using only 10% of the observations or 100% of them. The memory usage (figure 15) is showing a similar behavior, with only a marginal increment in the memory used by the prediction module (from 4.7GB to 5.3GB, probably due to the need to load in memory the additional amount of data related to the full dataset vs only 10% of it). Experiments show that the CPU usage (figure 14) is always below 80% and the memory usage is below 5.5 GB. The average latency of each prediction API call is of about 285 milliseconds, with 3.5 requests per seconds on average answered by the platform, and about 1,050 requests answered by each scenario in the 5 minutes of the experimental test.

As to conclude the analysis for this step, we highlight that the number of observations in the dataset is limiting our experimentation in stressing the scalability capacity of the platform. Indeed, we do not observe any increase in the latency or in the resource usage by using only 10% of the observations compared to using the full dataset. We plan to extend the scalability test of this step in D7.3, with artificial generated data of a significantly increased data volume to be predicted by the module.

3.1.4 End to End Report

In conclusion to our experimentation for the data flows, for this specific pilot we can see that overall the performance of the stack shows the necessary scalability. In particular the data pipeline step, involving Apache Kafka, MongoDB and Elasticsearch shows very high scalability when performed with high parallelization and low performance in terms of completion time if done otherwise. For the satellite image processing dataset ingestion, we consider that the best performance is also achieved when increasing its concurrency. However, high network usage is observed in this case, which leads us to believe that further experimentation is needed to ensure that this observation does not create a problem when the stack is used by many concurrent users and pilots. The rdfization process shows a nice performance when performed using the command line tool. Moreover, for the extraction of semantically enriched data, an excellent performance is observed in the real-life scenario. However, as the data volumes increase, high completion times are observed which leads us to believe that further experimentation is needed, along with possible changes in terms of employed technologies/frameworks for this specific step. Finally, the prediction step shows interesting performance in terms of latency and resource usage, disregarding the number of observations to use for prediction. In the future, we plan on further experimenting on this step by using artificial generated data as to stress the performance of the correlation task with an increased amount of data.

It should be noted at this point, as mentioned in the introductory section, that all of the described experimentation has been performed for the completed steps that are integrated into the BDG stack. Upon completion and integration of the rest, the experimentation will be updated to include the new results.

3.2 Natural Cosmetics Pilot

The natural cosmetics pilot focuses on the prediction of the biological efficacy of pharmaceutical plants. Currently this pilot has provided a dataset covering the lab experiments performed on the plants which is linked with satellite image processing of the respective fields in order for the correlation of lab tests and satellite images to be made possible.

3.2.1 Dataset Analysis & Evaluation

As mentioned in the pilot's introductory section, this pilot has provided 2 datasets. One is the lab tests performed on pharmaceutical plants and the other is the geographical information and related metadata of the fields the tests were performed against. Using the latter another dataset is also employed, that of the satellite image processing for these specific fields. In the tables below we describe these datasets and evaluate them against the 4Vs of Big Data.

Table 5 Laboratory tests dataset

Dataset	Laboratory Tests	
Metadata/Description	This dataset contains the laboratory tests performed on the pharmaceutical plants	
Provider	Symbeeosis	
Sample	A sample of this dataset can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume the provided dataset does not show a high value, leading to the need of the generation of synthetic data
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 10 different types of tests, which can be considered valid for our experimentation
	Veracity	In the context of veracity, we will follow the methodologies suggested in D7.1, for the generation of large amounts of synthetic data that follow the statistical distribution of the raw data provided. More specifically the methodology described here will be followed for the generation of synthetic data.

Table 6 Field metadata & Geographical information dataset

Dataset	Field Metadata & Geographical Information	
Metadata/Description	This dataset contains the metadata of the fields the laboratory tests were performed in. It contains geographical information, grape variety cultivated and date the samples were collected	
Provider	Symbeeosis	
Sample	A sample of this dataset can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume the provided dataset does not show a high value, leading to the need of the generation of synthetic data
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 6 different types of data values, which although is not a high value cannot be increased
	Veracity	In the context of veracity, due to the need of synthetic data generation, we will follow the statistical distribution of the raw data provided

This pilot is also using the dataset of satellite image processing which is described in section 3.1.1.

As described in this section the need for synthetic data generation has been identified. We perform this generation following the methodologies described in D7.1.

3.2.2 Data Flows

In this section, we describe the workflows the Natural Cosmetics pilot follows inside the BDG stack, during which the experimentation will be performed. It should be noted that the identified steps are the currently completed and integrated ones into the BDG stack that support this specific use case.

First, the dataset is uploaded into the system, either by using the provided UI (User Interface) per pilot, or by using the specific API endpoint as described in D6.2. Second, the preprocessed dataset is sent over to the Rdfization service for the RDF generation and then it is imported into the GraphDB storage engine for the semantic enrichment of the provided data. The next step of this pilot is to export the enriched data from GraphDB. The exported semantically enriched data is then inserted into the Elasticsearch instance along with the satellite image processing data that cover this pilot's fields. Finally, all of the data that resides into Elasticsearch is exported and used for the correlation of the provided values by querying the specific API endpoint.

3.2.3 Experimentation

In this section, we perform the rigorous testing experimentation for each of the identified steps, using the synthetic dataset generated. The dataset that was generated contains 3M laboratory tests that cover 200K fields and all of the generated records follow the statistical distribution of the initial one provided by the pilot. As described in the methodology section we will perform our experimentation using three scenarios for each step, to better showcase the scalability potential of the deployed stack.

3.2.3.1 Dataset Upload

To experiment on this step, we choose a scenario that involves high amounts of data, along with high concurrency in the requests made towards the respective API endpoint. For the needs of this specific experiment we assume as usage scenario that under which the dataset upload requests involve 10,000 entities and the concurrency of them is set to 1,000, 5,000 and 10,000 respectively for each testing scenario.

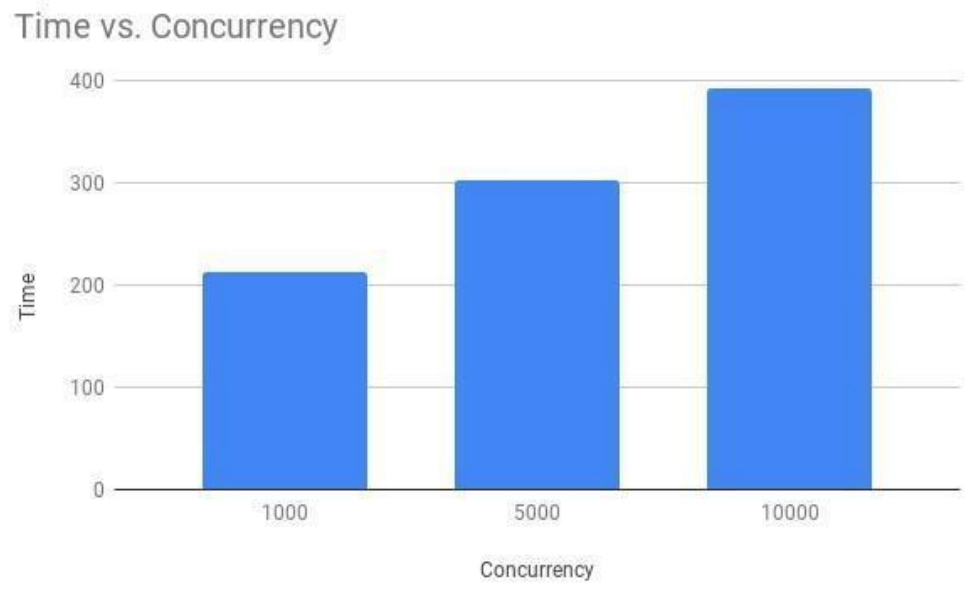


Figure 16 Completion time of dataset upload step

Figure 16 shows the completion time against the concurrency of the requests to the stack. As it is demonstrated in the figure below, completion time shows a high increase when moving from 1000 to 5000 concurrent requests, whereas when moving to 10000 concurrent requests the completion time although increased has a lower increase rate.

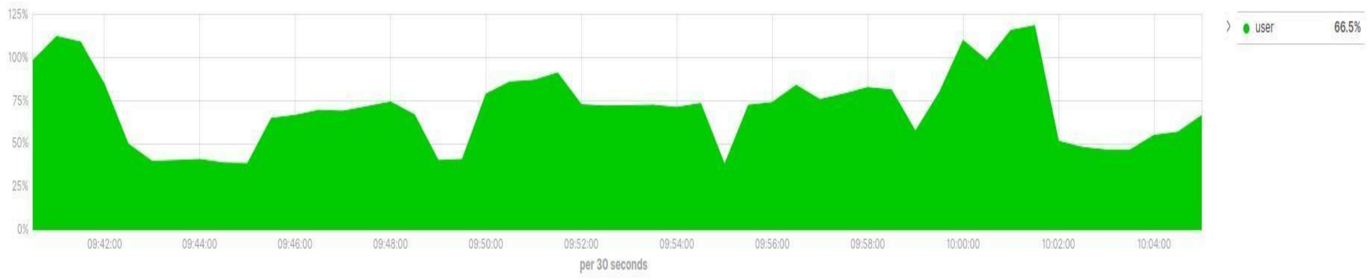


Figure 17 CPU usage during dataset upload

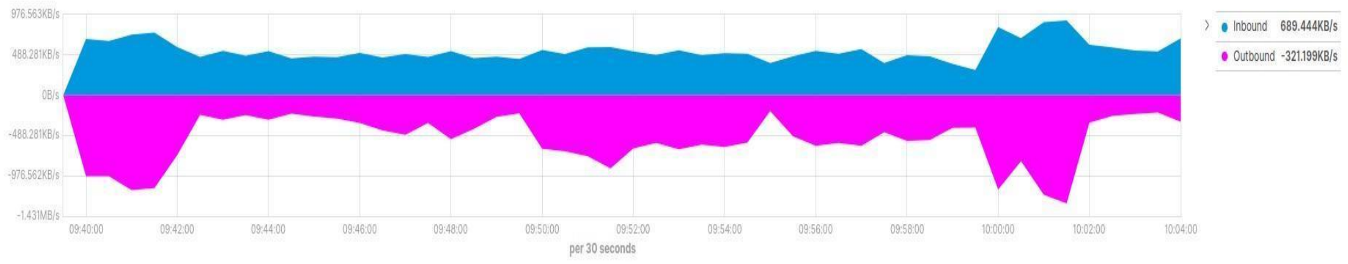


Figure 18 Network usage during dataset upload

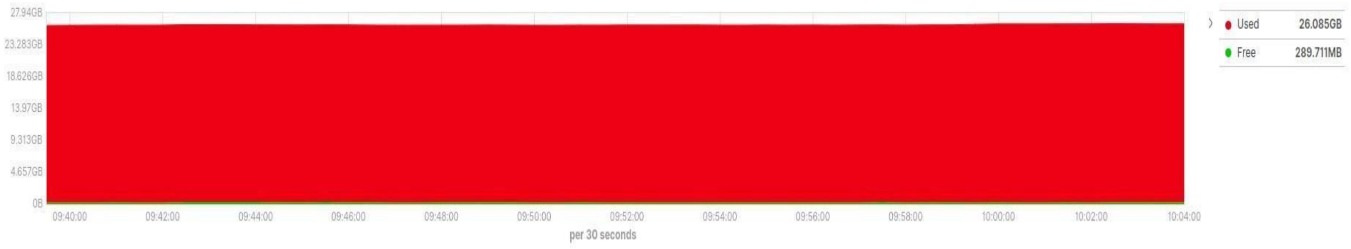


Figure 19 Memory usage during dataset upload

The figures shown above depict the CPU usage, network traffic in bytes and memory used by the stack, throughout the execution of the three scenarios. We observe a slight increase in terms of CPU usage (figure 17) between the scenarios starting from an average of 70% for the one with 1000 concurrent requests, 75% for the next one and an average of 80-85% for the last scenario with the 10000 concurrent requests, with a peak of 120%. In terms of network traffic (figure 18), we observe a steady rate of inbound traffic for the first two scenarios and a high increase for the last one leading to 800KB/s, the outbound traffic shows a linear increase for the three scenarios with an average of 300KB/s, 700KB/s and 1.2MB/s respectively for each. Finally, a stable memory usage (figure 19) is observed which is due to the fact that most of the components in the stack are based on the JVM (Java Virtual Machine) which can be set to reserve the required memory upon startup.

We conclude our experimentation on this step by observing that the best usage of the platform taking into account all of the monitored metrics is the first one, where small amounts of data are ingested into the stack. To that end, we change the initial ingestion of data for this pilot to have its datasets uploaded in small batches of concurrent requests.

3.2.3.2 Rdfization & Semantic Enrichment

In section 3.3.3.3 we thoroughly experiment on this step with higher volume datasets so do not conduct the same experiment here with the provided datasets.

3.2.3.3 Ingestion of Enriched Data

In this step we experiment with the ingestion of the semantically enriched data for the Natural Cosmetics pilot. To fully investigate this step, we identify 3 usage scenarios, based on the data we will extract from GraphDB and ingest into Elasticsearch. To this end, the first scenario involves the ingestion of 100K of the laboratory test, the second one the ingestion of 1M laboratory test and the final one the ingestion of 3M lab tests. The ingestion of the laboratory tests is done in batches of 1000.

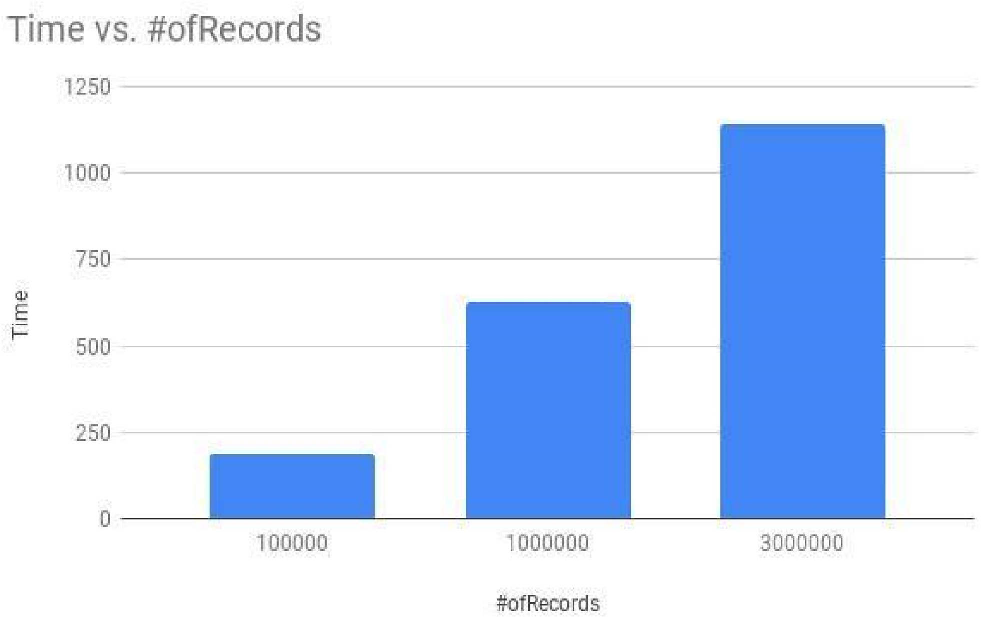


Figure 20 Completion time of enriched data ingestion

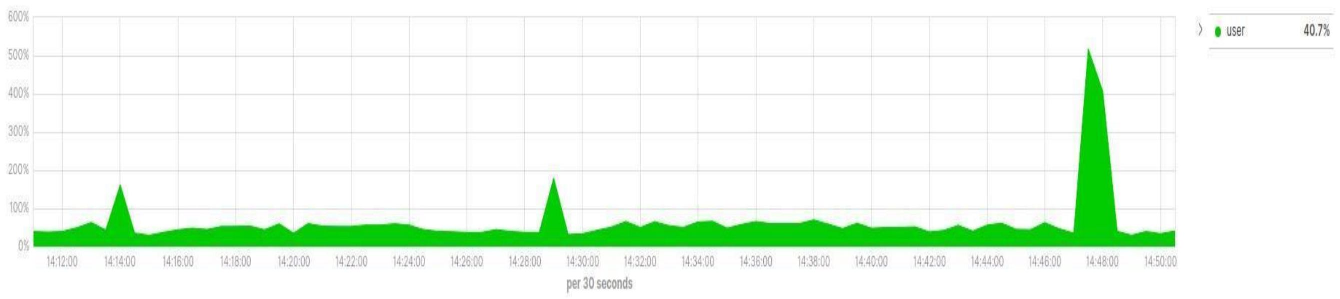


Figure 21 CPU usage during ingestion of enriched data

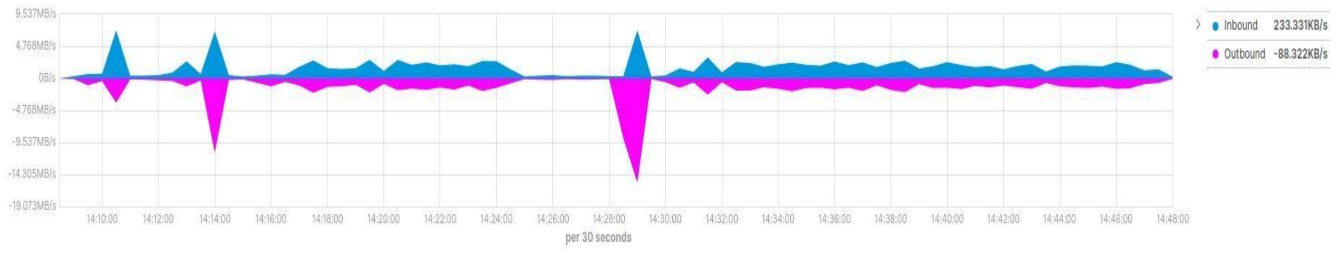


Figure 22 Network usage during enriched data ingestion

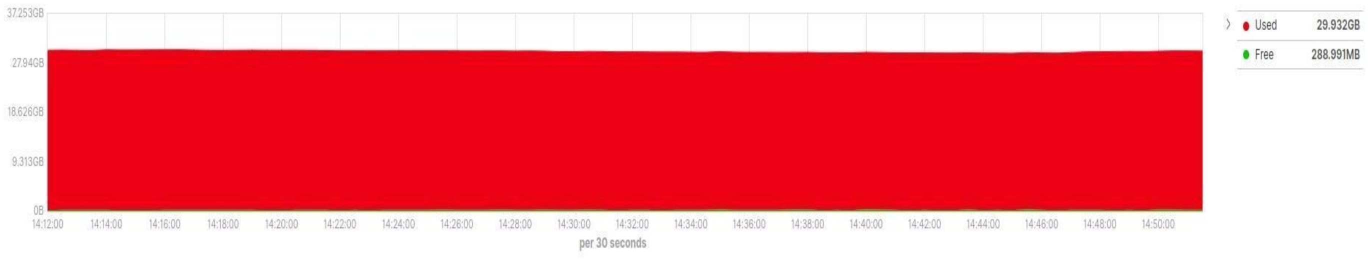


Figure 23 : Memory usage during enriched data ingestion

As demonstrated in the figures above, we observe a steady increase in terms of completion time (figure 20) for each scenario, easily explained by the increase of the total number of records required for each scenario.

Interestingly though the completion time does not show a linear increase with respect to the number of records, since in order to ingest all of the synthetically generated data for this pilot (3M records), we observe a roughly 2 times increase in terms of completion time compared to the second scenario where 1M records are ingested. In terms of CPU usage (figure 21), as shown in the respective figure, we observe a steady usage of an average of 50%, spiking to 150% and 520% for the second and third scenario respectively. The network traffic (figure 22) also shows a steady increase throughout the employed scenarios, starting from 800KB/s for the first scenario, 1.5-2MB/s for the second and 2.5MB/s for the last one. As shown in Figure 23, the memory usage remains steady, as is the case with the previous steps of all the pilots.

In conclusion of the experimentation with the current step, we observe a close to linear increase in terms of completion time as the volume of the ingested data increases. However, due to the high network traffic generated in the last two scenarios, we believe that this ingestion should be triggered once a record is inserted into GraphDB and the enrichment is done, or in smaller batches of record, so that it does not affect other components running at the same time.

3.2.3.4 Ingestion of Satellite Image Processing Data

We move on with the experimentation over the ingestion of the Natural Cosmetics related dataset that comes through GEOCLIDEAN'S API, containing the satellite image processing data. For this step we will follow an approach like the previous one. To that end, since the actual supplied fields by Symbeeosis are 13, we experiment on this step in the following 3 scenarios: one under which the ingestion of 1 field's processed image data occurs, one in which 7 fields are ingested concurrently and finally one where all of the supplied fields are ingested at the same time.

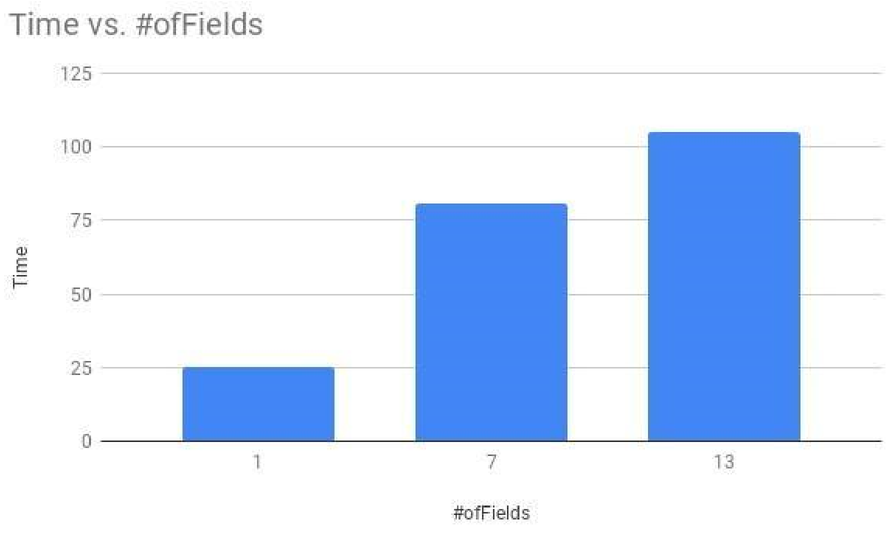


Figure 24 Completion time of satellite image processing data ingestion

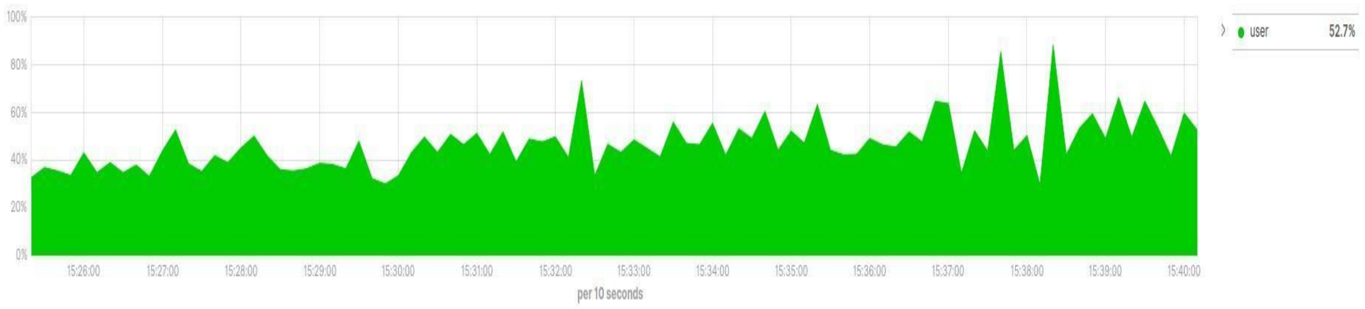


Figure 25 CPU usage during satellite image processing data ingestion

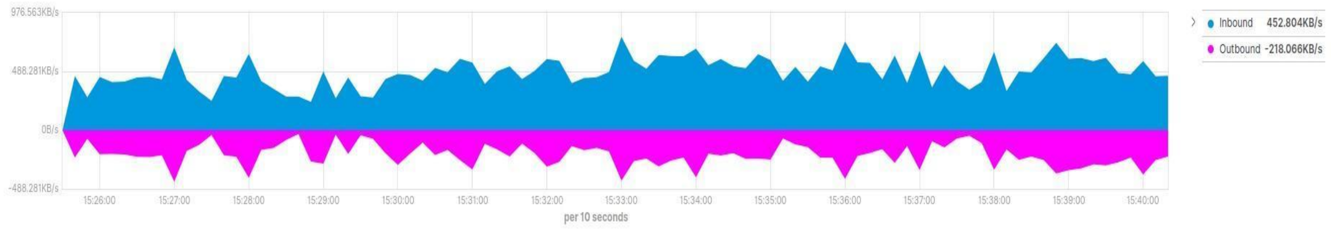


Figure 26 Network usage during satellite image processing data ingestion

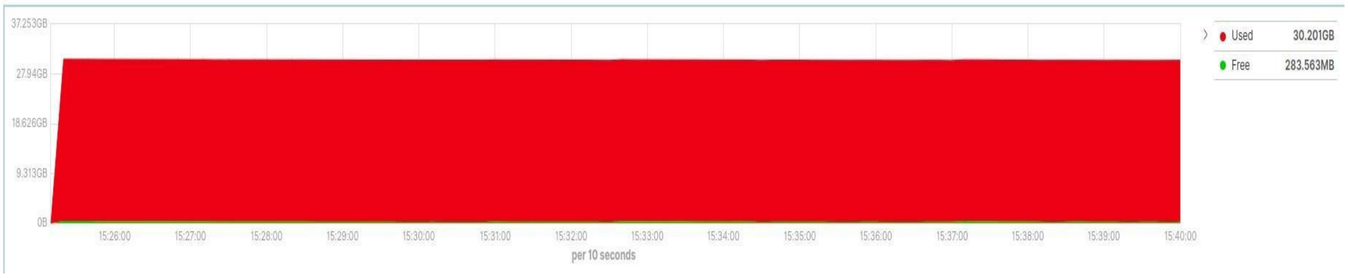


Figure 27 Memory usage during ingestion of satellite image processing data

As shown in the figures above (figure 24), we observe a low completion time for each step, varying from 25 seconds to 110 seconds. In terms of CPU usage (figure 25), a low value is observed, on average 40% for the first two scenarios and a 8090% usage for the last one where all of the image processing data for every field is ingested. Finally, in terms of network usage (figure 26) we observe a steady value of average 500KB/s for each of the scenarios employed and a steady memory usage (figure 27) for the whole stack.

As further described in section 3.1.3.2, where we experimented with the ingestion of all of the data that reside in GEOCLIDEAN's service, we consider this step to have a very high efficiency value considering the metrics we monitor.

3.2.3.5 Correlation of Data

The final step for this pilot's datasets is the correlation one, in which the semantically enriched laboratory tests will be correlated with the results of the satellite image processing. We identify the following usage scenarios for this step: in the first one we will call the correlation API with 230K samples from the lab tests, which is on average the data from one field, in the second one we will move on with the usage of 50% of the fields data, a total of 1.5M records. In the final usage scenario, we will employ all of the 3M synthetically generated data series. It should be noted that in all scenarios covering more than one field, we evenly distribute the lab tests across all of the 13 supplied fields.

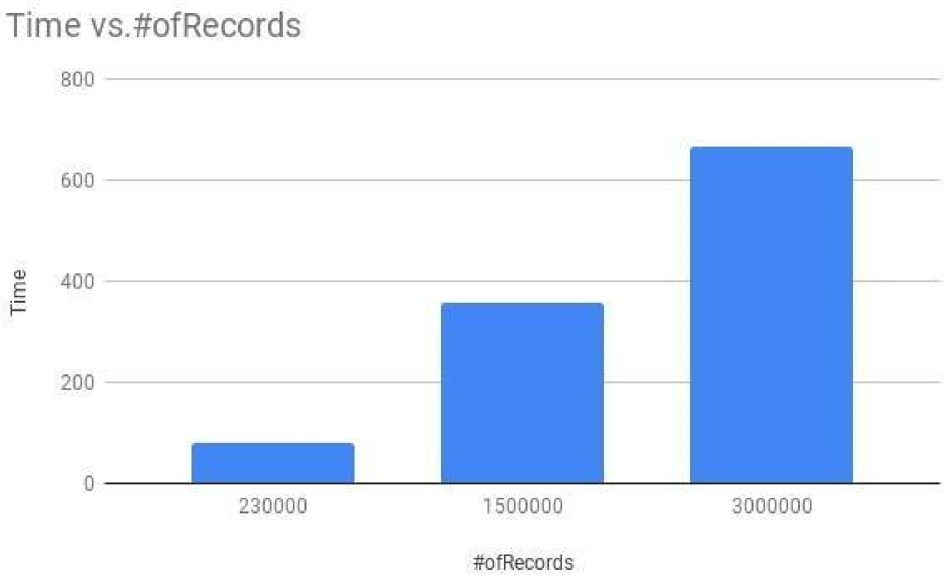


Figure 28 Completion time of data correlation step

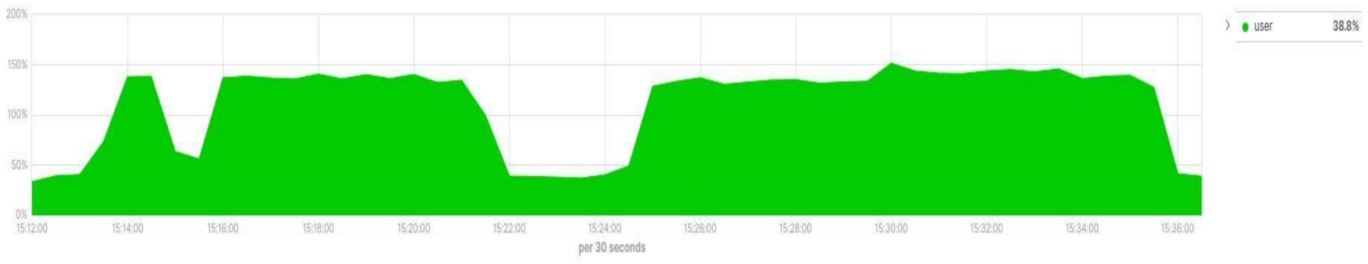


Figure 29 CPU usage during correlation of data

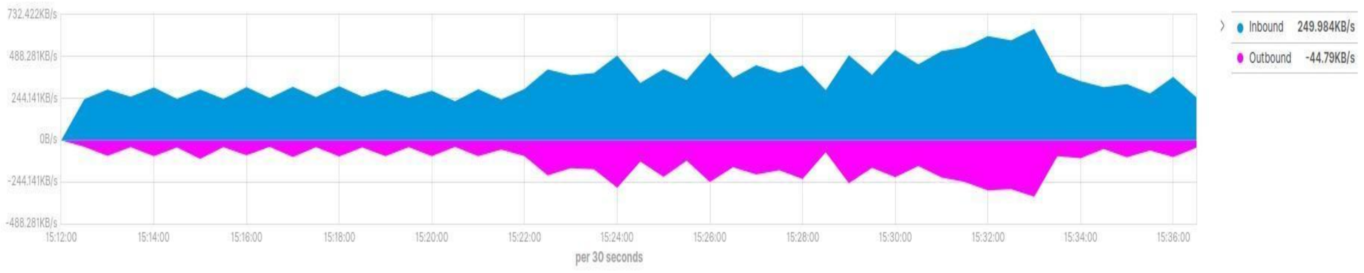


Figure 30 Network usage during correlation of data

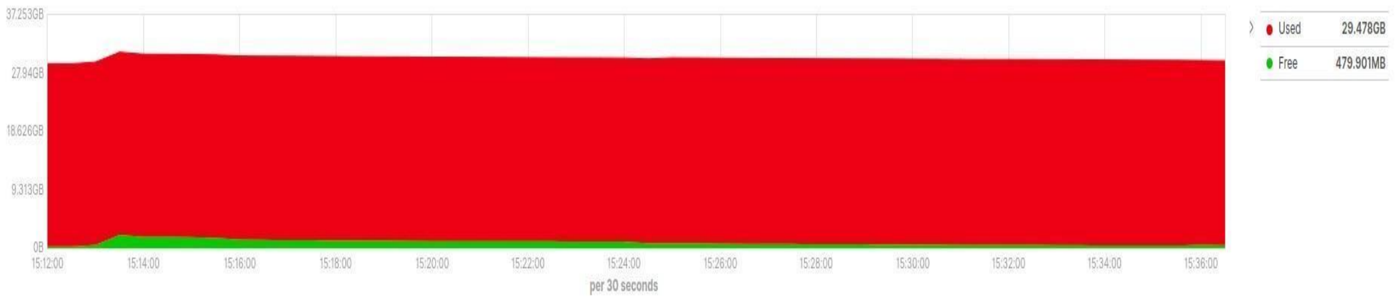


Figure 31 Memory usage during correlation of data

As shown in figure 28 the completion time increases as we supply more data to the correlation script. More specifically, 80 seconds are required for the completion of the first scenario, 356 for the second and 667 for the scenario under which all of the synthetically generated data are provided to the script. Interestingly, regardless of the size of the provided datasets, the CPU usage (figure 29) remains at the same levels across all experimentations, roughly 140%. This can be explained by the fact that the correlation script is configured to run in a single thread, regardless of the dataset's size. In terms of network traffic (figure 30), we observe a steady increase as we progress through the scenarios starting from 300KB/s and leading to 650KB/s. This fluctuation can be explained by the fact that each correlation script execution results in a zip file generation and download by the respective API's consumer. Finally, in terms of memory (figure 31) we see that a steady value is observed, with the exception of an initial cache clear performed by the platform, resulting in an increase in the memory usage.

In conclusion to our experimentation for this step, we consider the last usage scenario as the real-life one. Correlation workflows are heavily dependent on the volume of the input datasets, so the last usage scenario best simulates the desired production state of this step. A drawback of this step is the execution time required for the actual correlation to happen. To ameliorate this, we consider splitting this step into two distinct substeps, one in which the correlation of the provided data happens using offline cron jobs and another that returns the produced histograms in real-time.

3.2.4 End to End Report

In conclusion to our experimentation for the Natural Cosmetics pilot and the data flows its dataset have inside the BDG stack, we consider the overall performance of the distinct steps as very successful. In the context of the data uploading step we observed that this step is an easily scalable one, since even with 10K concurrent requests the respective components did not show any downtime. However, due to the high network usage as the concurrency increases, we note as an upper bound for the performance of the stack that of 5,000-7,000 concurrent requests at most. The rdfization step also shows very good performance when executed using the command line tool, a tool triggered by *cron* jobs [6]installed in the platform. The extraction of the semantically enriched data is the one presenting a bottleneck for this pilot (as was the case for the previous). This leads us to believe that to achieve the best performance for this step, the exported data should either be split into smaller batches or further experimentation employing different components of the stack should be investigated. In the context of the satellite image processing data, we consider this step as a highly performant one, since its completion time and the monitored metrics show very good values for the provided fields of this pilot. Finally, the correlation of data, also shows very good performance, taking into consideration the nature of this specific step. In the future we plan on further experimenting on this step by increasing the concurrency of the requests sent to it to better showcase its scalability.

It should be noted at this point, as mentioned in the pilot's introductory section, that all of the described experimentation has been performed for the completed steps that are integrated into the BDG stack. Furthermore, due to the lack of Big Data coverage for the provided datasets, the experimentation was performed with high amounts of synthetic data, which although follow the statistical distribution of the provided may decrease the accuracy of the results.

[6] https://cron-job.org/en/

3.3 Table and Wine Grapes Pilot

The table and wine grapes pilot focuses on the prediction of the quality of the final product of table and wine grapes. This is achieved through the provided datasets of this pilot: datasets that involve soil and weather data, as well as crop qualitative and quantitative data. In the sections that follow, we analyze the provided datasets against Big Data 4Vs, we identify the flows these datasets follow inside the BDG stack and we perform our experimentation on each using the methodology described in the introductory section.

3.3.1 Dataset Analysis & Evaluation

As mentioned in the pilot's introductory section, this pilot has provided different datasets that cover three different fields, resulting in a total of 22 datasets, since one of the fields is lacking an IoT installation. In the tables below we describe these datasets and evaluate them against the 4Vs of Big Data.

Table 7 Geographical & Field metadata dataset

Dataset	Geographical & Field Metadata	
Metadata/Description	This dataset contains geographical information for each field along with the grape variety cultivated	
Provider	AUA	
Sample	Samples for this dataset can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume the provided datasets cover 3 fields which can be further expanded into 12 considering the different varieties cultivated at each field, technically making it a different field.
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 5 different data types which is not a high variety value
	Veracity	In the context of veracity, we will follow the methodologies suggested in D7.1, BDGS and BigDataBench, for the generation of large amounts of synthetic data that follow the statistical distribution of the raw data provided

Table 8 Soil quality dataset

Dataset	EM38

Metadata/Description	This dataset contains soil quality data
Provider	AUA
Sample	Samples for this datasets can be found here
Evaluation	

Big Data Vs	
Volume	In terms of volume the provided datasets have a total of 67K measurements, which if associated with the different data types lead to a dataset that has satisfying volume
Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
Variety	In terms of variety the provided dataset tracks 10 different data types
Veracity	In terms of veracity, there is no need for synthetic data generation

Table 9 Vegetation indices estimation dataset

Dataset	RapidScan
Metadata/Description	This dataset contains data used to estimate vegetation indices such as NDVI and NDRE indices
Provider	AUA
Sample	Samples for this datasets can be found here
Evaluation	

Big Data Vs	
Volume	In terms of volume the provided datasets on RapidScan have a total of 400 rows, resulting to a need of synthetic data generation
Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity

	Variety	In terms of variety the provided dataset tracks 22 different data types, which is considered valid for our experimentation
	Veracity	In terms of veracity, we will follow the methodologies suggested in D7.1, BDGS and BigDataBench, for the generation of large amounts of synthetic data that follow the statistical distribution of the raw data provided

Table 10 Vegentation indices dataset

Dataset	SpectroSense	
Metadata/Description	This dataset contains data used to estimate LAI (Leaf Area Index) and NDVI vegetation indices	
Provider	AUA	
Sample	Samples for these datasets can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume the 17 provided datasets on RapidScan have a total of ~10.5K rows
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 12 different data types, which is considered valid for our experimentation
	Veracity	In terms of veracity, there is no need for synthetic data generation

Table 11 Radioactive transfer & biophysical characteristics dataset

Dataset	CropCircle
Metadata/Description	This dataset contains data used to estimate the radioactive transfer and the biophysical characteristics of plant canopies.

Provider	AUA	
Sample	Samples for this datasets can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume this dataset has ~100K rows
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset has 6 data types, including geographical information
	Veracity	In terms of veracity, there is no need for synthetic data generation

Table 12 Laboratory tests dataset

Dataset	Laboratory Tests	
Metadata/Description	This dataset contains laboratory tests performed on samples from the fields	
Provider	AUA	
Sample	Samples for these datasets can be found here, and here	
Evaluation	Big Data Vs	
	Volume	In terms of volume this dataset has ~1200 rows, which is not considered a high value
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However for the needs of our experimentation we can mock the desired velocity
	Variety	The provided dataset has 9 different data types
	Veracity	In terms of veracity, we will follow the methodologies suggested in D7.1, BDGS and BigDataBench, for the generation of large amounts of synthetic data that follow the statistical distribution of the raw data provided

Table 13 IoT data

Dataset	IoT data
Metadata/Description	This dataset contains data collected from IoT installations on 2 different fields
Provider	AUA
Sample	Samples for these datasets can be found here
Evaluation	

Big Data Vs	
Volume	In terms of volume the provided API endpoints show a total of 260K of data points that is constantly growing
Velocity	The provided datasets are API endpoints that have bursts of new data every 5 minutes
Variety	In terms of variety the provided dataset tracks 5 different data types, which is not a high value
Veracity	In terms of veracity, there is no need for synthetic data generation due to the combination of volume and variety

Table 14 Yield data dataset

Dataset	Yield data
Metadata/Description	This dataset contains data on the yield of a specific field
Provider	AUA
Sample	Samples for this datasets can be found here, and here
Evaluation	

Big Data Vs	
Volume	In terms of volume the provided dataset has 100 rows, which is not a high value
Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity

	Variety	In terms of variety the provided dataset tracks 4 different data types, along with metadata information on the field, which is not a high value
	Veracity	In terms of veracity,we will follow the methodologies suggested in D7.1, BDGS and BigDataBench, for the generation of large amounts of synthetic data that follow the statistical distribution of the raw provided

Table 15 Qualitative data dataset

Dataset	Qualitative data
Metadata/Description	This dataset contains data on the quality characters of the grapes of a specific field
Provider	AUA
Sample	Samples for these datasets can be found here
Evaluation	

	Big Data Vs	
Evaluation	Volume	In terms of volume the provided dataset has ~50 rows, which is not a high value
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 46 different observations
	Veracity	In terms of veracity, we will follow the methodologies suggested in D7.1, BDGS and BigDataBench, for the generation of large amounts of synthetic data that follow the statistical distribution of the raw data provided

Table 16 Photosynthesis data dataset

Dataset	Photosynthesis data

Metadata/Description	This dataset contains data on the photosynthesis measurements taken from selected plants of a specific field.	
Provider	AUA	
Sample	Samples for these datasets can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume the provided dataset has ~250 rows, which is not a high value
	Velocity	The provided dataset is a static file and does not show a high velocity rate. for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 65 different observations
	Veracity	In terms of veracity, we will follow the methodologies suggested in D7.1, BDGS and BigDataBench, for the generation of large amounts of synthetic data that follow the statistical distribution of the raw data provided

This pilot is also using the dataset of satellite image processing which is described in section 3.1.1.

Due to the high variety observed across all of the AUA provided datasets and taking into account the high volume of most of them, we chose not to generate synthetic data for this use case and focus our experimentation on the provided ones.

3.3.2 Data Flows

In this section we describe the workflows the AUA pilot follows inside the BDG stack, during which the experimentation will be performed. It should be noted that the identified steps are the currently completed and integrated ones into the BDG stack that support this specific use case.

Initially, the datasets are uploaded into the system, either be the provided UI per pilot, or by the respective API endpoint as described in D6.2. Depending on whether it is metadata information of actual data, we will insert them into MongoDB and Elasticsearch respectively. The data provided by the IoT installations follow the data stream pipeline of the stack that involves their ingestion through Apache Kafka. All of the ingested data is initially stored on BDG's Elasticsearch instance. The cleaned and preprocessed data is then sent over to the Rdfization service for the RDF generation and import into the GraphDB storage engine for the semantic enrichment. The next step for this pilot is to export the enriched data from GraphDB and the ingestion of it along with the dataset of the satellite image processing into the Elasticsearch instance. Finally, for this specific pilot's needs there are two different correlations that are expected: one that correlates the lab tests with the results from the satellite image processing and another that attempts to correlate the yield of a field with the respective lab tests and environmental indicators of the provided datasets.

3.3.3 Experimentation

In this section, we perform the rigorous testing experimentation for each of the identified steps, using the data provided by AUA. As mentioned in the dataset evaluation section we do not generate synthetic data for this specific use case and we experiment with the provided datasets that show a total of ~440K records spread across files and API endpoints. As described in the methodology section, we perform our experimentation using three scenarios for each step, to better showcase the scalability potential of the deployed stack.

3.3.3.1 Dataset Upload

Similarly, to the approach followed for the other use cases, we experiment on the dataset upload step using the provided datasets. These show a volume of 180,000 records with a variety of 68 data types, spread across seven datasets. We initially experiment on this step by uploading 50% of the provided data sequentially. We move on by attempting to upload all of the data sequentially and conclude this subsection by uploading all of the provided data with 22 concurrent requests, one per dataset.

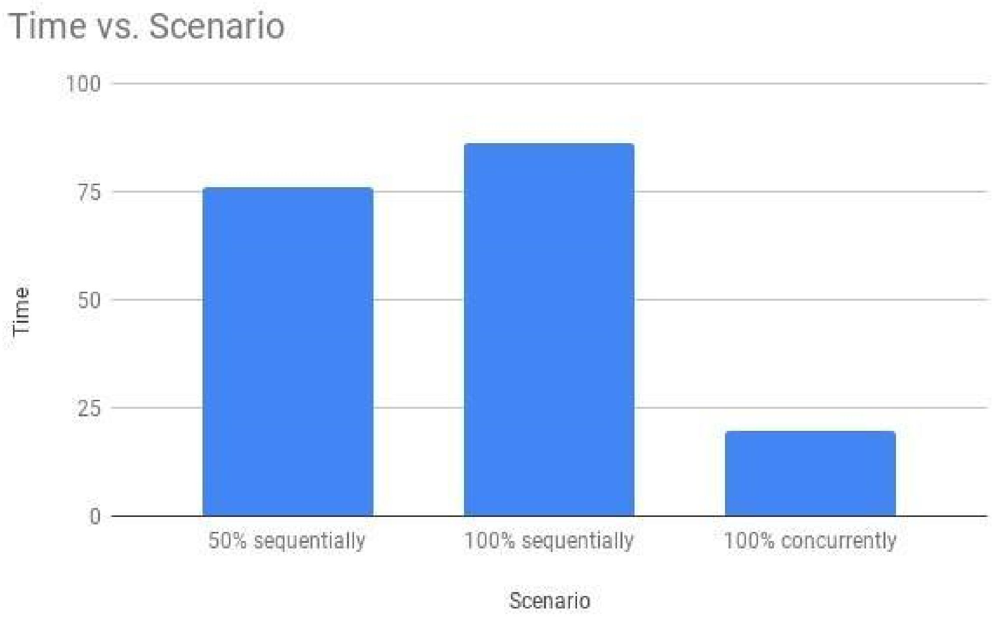


Figure 32 Completion time for dataset upload

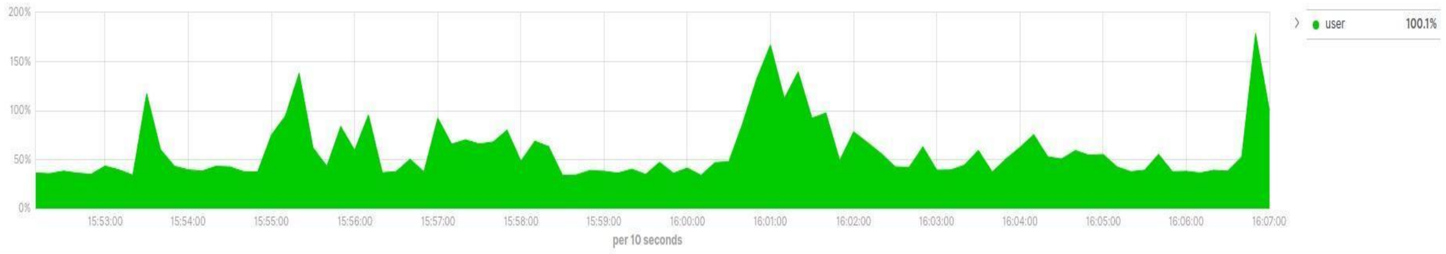


Figure 33 CPU usage during dataset upload

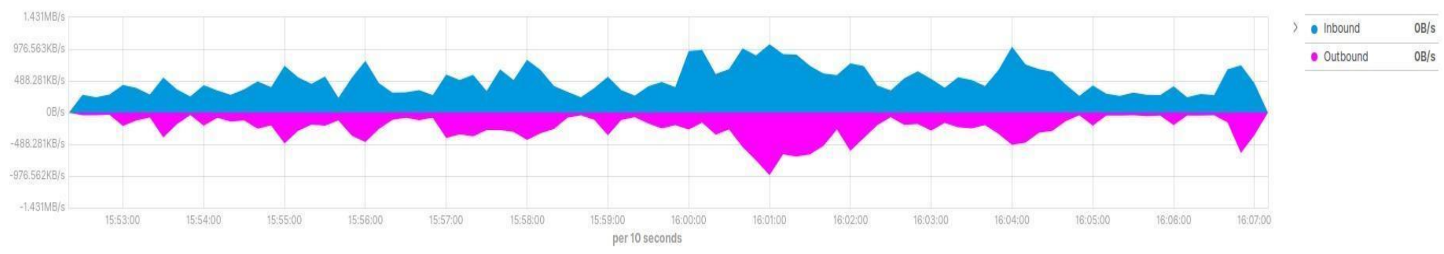


Figure 34 Network usage during dataset upload

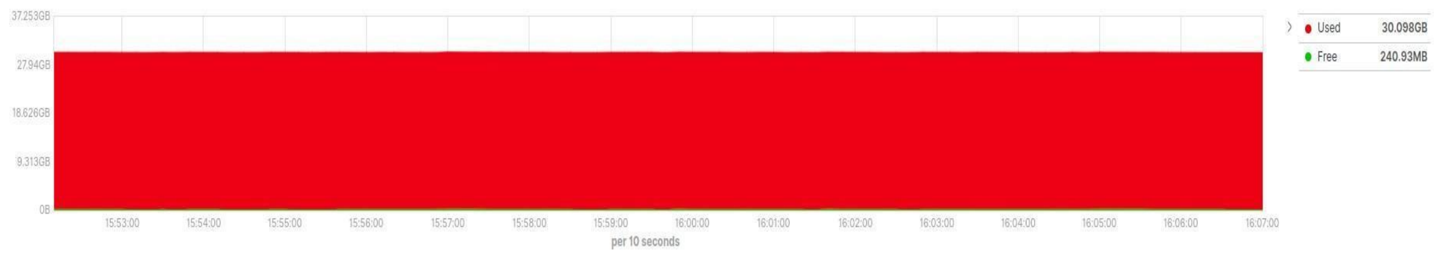


Figure 35 Memory usage during dataset upload

As shown in the figures above we observe a low increase ingesting 50% or all of the data sequentially and a much lower value when we attempted ingesting all of the data concurrently. We also observe a steady CPU usage (figure 33) for both the sequential scenarios of 80% and a peak of 180% for the last fully parallelized one. In terms of network usage (figure 34), we have a steady usage for the first 2 scenarios, an average of 400-500KB/s is observed and a slight increase for the last one of just above 500KB/s. Finally, the memory usage (figure 35) shows a very steady value throughout our experimentation.

In conclusion to this step's experimentation, we have observed that the fully parallelized experiment shows the best performance in consideration with the metrics we monitor, leading to us fully parallelizing this specific step.

3.3.3.2 Data streams Pipeline

To experiment on this step of the data flows for the AUA (Agricultural University of Athens) pilot, we will employ the following three usage scenarios: one under which we will ingest the latest data as they come, which is the reallife scenario, one under which we will ingest one's day data from the 2 available sensor APIs and a final one where we will harvest all of the data existent into the APIs and store them into the BDG stack. The metrics documented for this step involves the whole process of calling the sensor API to get the data, streaming them and performing further calculations on the data streams using Kafka and Kafka streams, modeling them into the internal BDG format and inserting them initially into Apache Cassandra to take advantage of the high write throughput and then into our Elasticsearch instance.

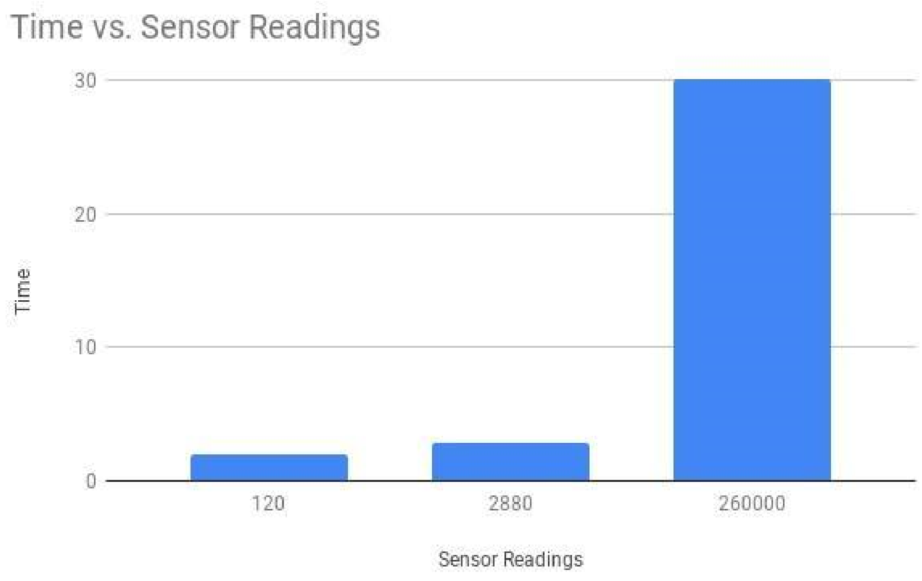


Figure 36 Completion time for data streams pipeline

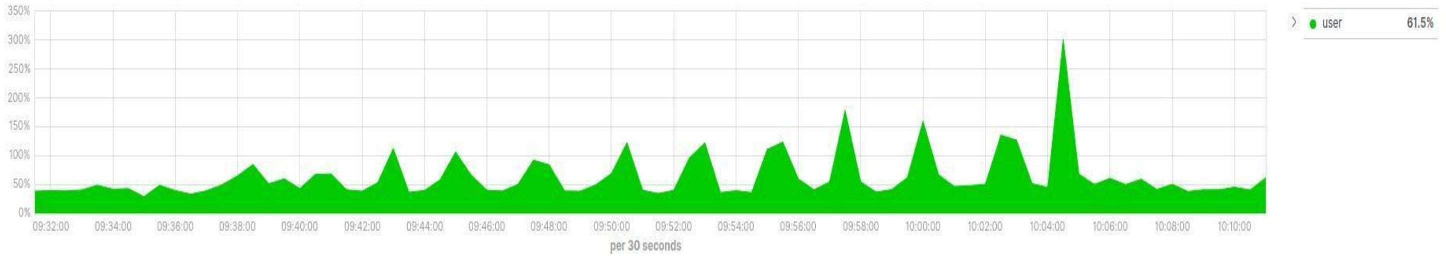


Figure 37 CPU usage during data streams pipeline

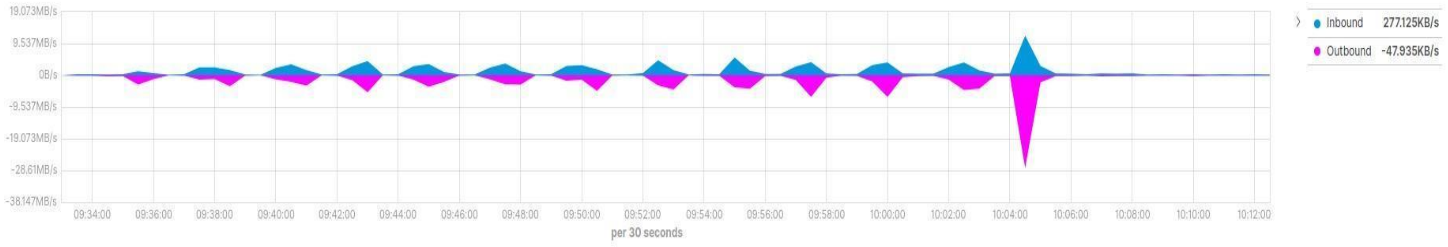


Figure 38 Network usage during data streams pipeline

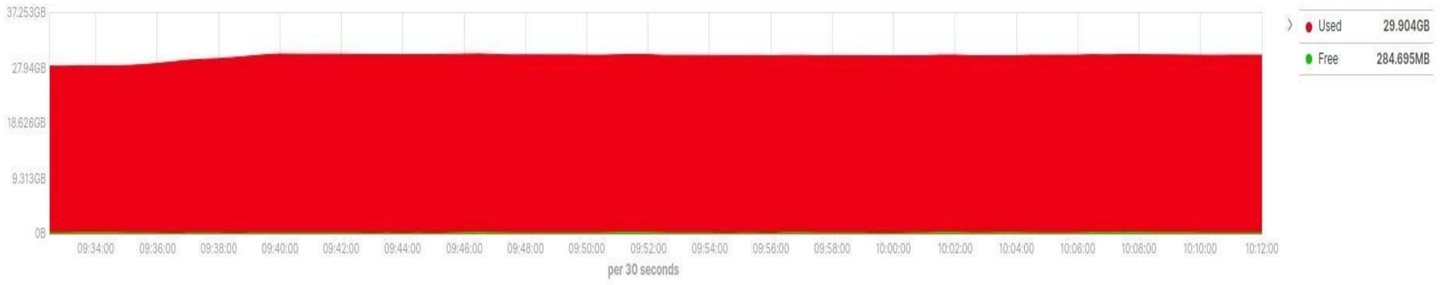


Figure 39 Memory usage during data streams pipeline

During our experimentation with the employed three scenarios for this step, we observe very low completion time (figure 36) for the first scenario where the latest data is ingested, 2 seconds, a minor increase in the case of the ingestion of 1 day's data, just below 3 seconds and an increase when attempting to ingest all of the currently generated data. The CPU usage (figure 37) of the whole stack shows a steady value for the all of the scenarios with an average of 40-45% and a spike of 300% for the last one. In terms of network traffic (figure 38), we observe high fluctuations varying between 600KB/s for the first scenario, 2.3MB/s for the second and 9MB/s in the last scenario. Finally, the memory usage (figure 39) remains around the same value throughout our experimentation.

In conclusion of the experimentation for this step, we consider the usage of BDG's components involved as very successful since the real-life scenario, which is the currently deployed one, shows a very high performance in respect to the monitored metrics.

3.3.3.3 Rdfization & Semantic Enrichment

RDF-ization os the process of converting tabular data to RDF. As described in D6.2, Section 2.2.4.1, we use the TARQL tool to perform this task. TARQL is built from the ground up to handle heavy loads and work on streaming data. To illustrate this, we ran the following test on the most voluminous dataset in the Table and Wines pilot - the IoT-stationary data (available here). This data consists of continuous meteorological observations on a given plot. The data is collected every 5 minutes, thus, one month of data corresponds to roughly 9,000 observations, and 1 year to 110,000 observations. The sensor reports 11 different values. To illustrate the performance of the RDF-ization pipeline we ran an experiment using 3 datasets:

- 1 month of data, corresponding to data for August 2018 - 8839 observations
- the full Palivou IoT data, corresponding to data for months 08-12 2018 - 32095 observations
- A dataset corresponding to 6 years of (simulated) data - 609787 observations Table 18 summarizes the results in terms of runtime[7] and number of resulting RDF triples.

Table 17 Experimentation results of rdfization & semantic enrichment

dataset	period	observations	runtime	resulting triples
palivou-08	1 month	8839	3,70s	141494
palivou-iot	5 months	32095	5,40s	513579
palivou-6y	6 years	609787	34,65s	9756453

As we can see, in the simulated 6y dataset the tool took 34 seconds to produce close to 10 million triples from 6 years' worth of dense meteorological data thus illustrating that its performance is capable of handling large amounts of data in manageable time.

3.3.3.4 Ingestion of Enriched Data

In this step we experiment with the ingestion of the semantically enriched data for the AUA pilot. To fully investigate this step, we identify three usage scenarios, based on the data we will extract from GraphDB and ingest into Elasticsearch. To this end, the first scenario involves the ingestion of 10K rows of data, the second one the ingestion of 200K and the final one the ingestion of all 440K data points provided by this pilot. The ingestion of is done in batches of 1,000. In the first two scenarios we evenly distribute the samples which we will ingest across all of the provided datasets by the pilot.

[7] On a modern workstation with an intel i7 processor

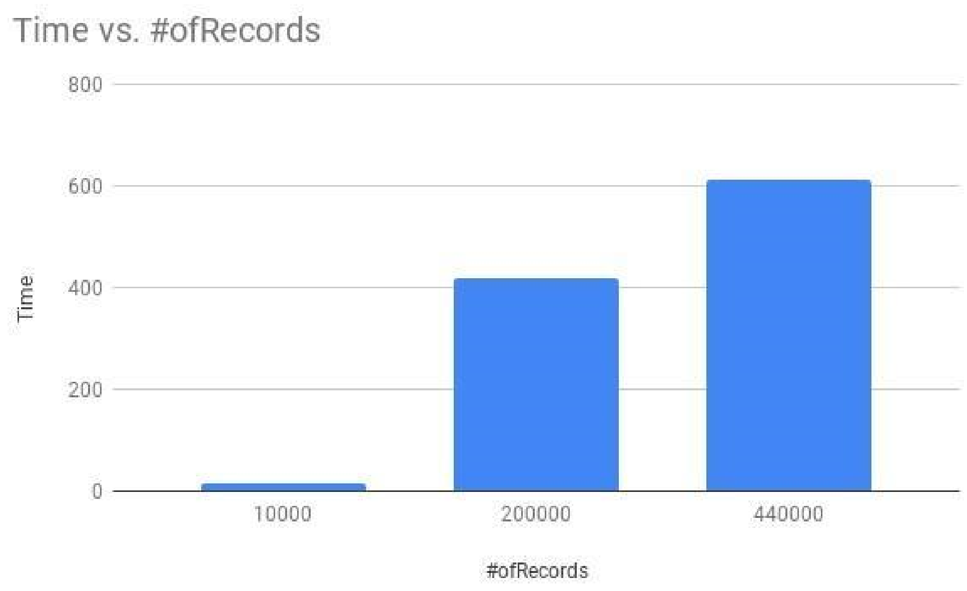


Figure 40 Completion time for ingestion of enriched data

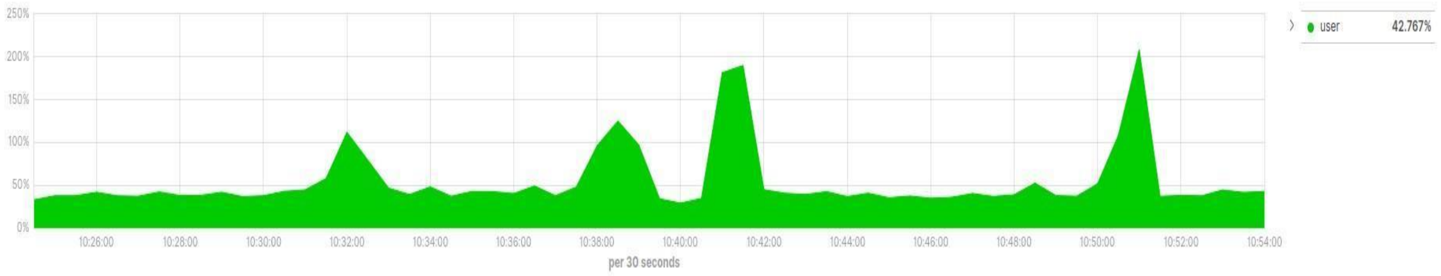


Figure 41 CPU usage during ingestion of enriched data

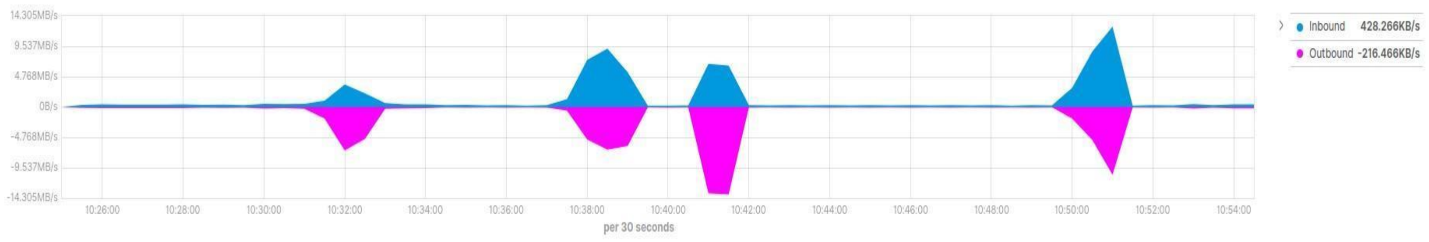


Figure 42 Network usage during ingestion of enriched data

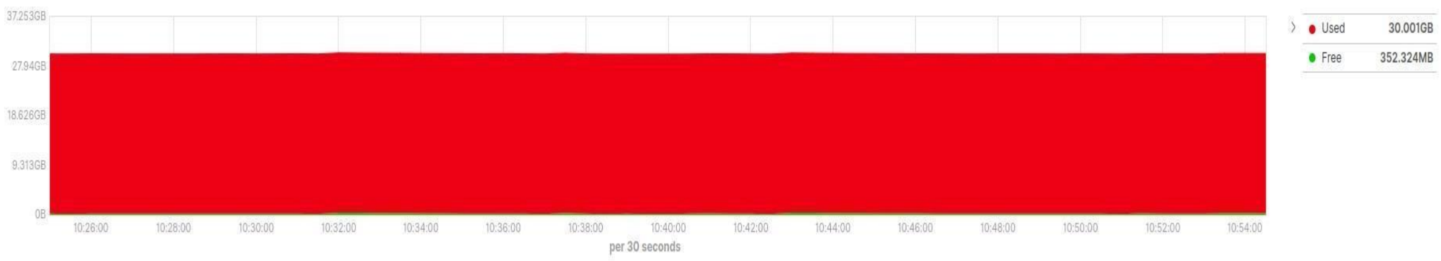


Figure 43 Memory usage during ingestion of enriched data

Following our experimentation for this step, we observe that the completion (figure 40) time for each scenario shows a steady increase, starting from 16 seconds for the ingestion of 10K records, 420 seconds are required to extract and store 200K of enriched data and for the final scenario, the ingestion of all of the provided data takes 613 seconds to complete. In terms of usage (figure 41), we observe the same average usage throughout the scenarios, 50%, with different maximum values for each, 60%, 110% and 200% respectively. In terms of network traffic (figure 42), we are seeing 1MB/s for the first scenario, increasing to 3.5MB/s for the second, leading to a 6.5MB/s for the last. Finally, the memory usage (figure 43) remains on the same values throughout our experimentation.

Concluding our experimentation for this step, we consider the first scenario as the most promising, since it has the lowest values on all metrics monitored, while having an acceptable network traffic. To that end, as was the case with other use cases for this specific step, we consider the best practise to be having offline tasks frequently querying for new data into the GraphDB engine, pushing changes into the Elastic stack.

3.3.3.5 Ingestion of Satellite Image Processing Data

This pilot covers three different fields on which the satellite image processing data are required. However, since this step is identical to the one required by the one described in 3.2.3.4 for the Natural Cosmetics pilot but requires less fields, we believe that the testing reported in that section clearly overlaps with this one, with the previous experimentation being on a higher scale.

3.3.3.6 Correlation analysis

The analytic step related to this pilot concerns the development of a correlation analysis between three data layers: i) ground (location-specific) data, i.e., sensor data; ii) lab data; iii) airborne data, i.e., satellite imagery. The Palivou Mesi field, located in Nemea, is used as a study case and the field is split into 100 cells of equals size. For the purpose of this step, we will use the *<api_url>/correlation/sensorvslab* API handling associations and correlations between precision agriculture information (sensor data) and phenological data and grape chemical analysis (lab data). In particular, we identify the following two scenarios: in the first one we will call the correlation API by using only 10 cells out of 100, i.e., 10% of the field area. In the second scenario, we will use all the 100 cells composing the field, thus using the full field area. It should be noted that we used the ***sensorvslab*** API for this experimentation, with similar results observed by using the ***sensorvssatellite*** and ***labvssatellite*** APIs. Given the correlation API is very fast to answer, we simulated 5 minutes of sequential API call with the first scenario and 5 minutes of sequential API with the second scenario.

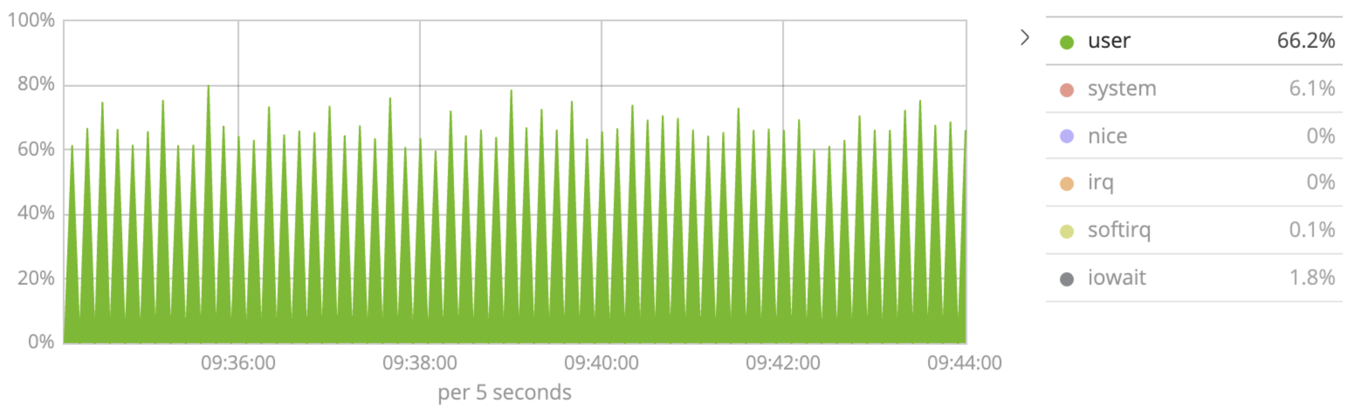


Figure 44 CPU usage during correlation API call

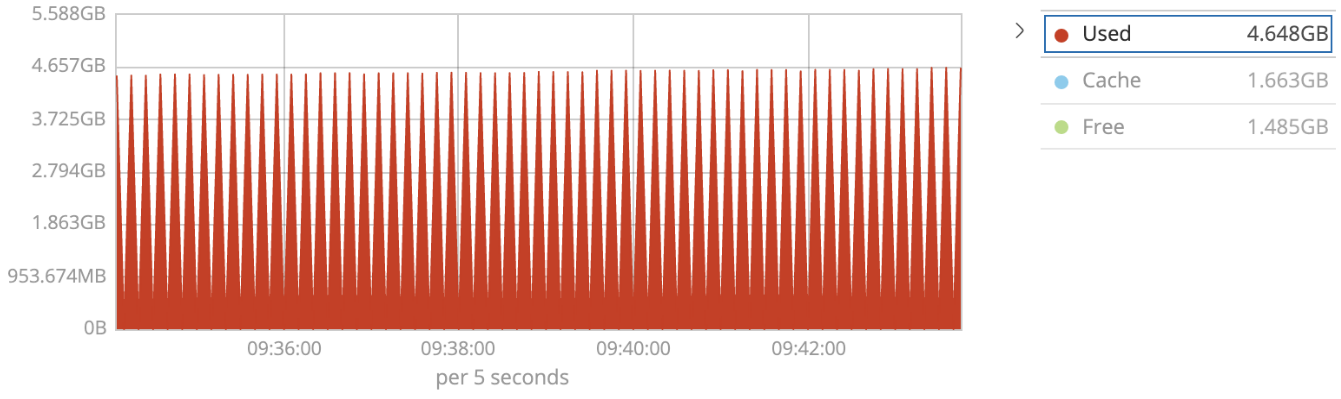


Figure 45 Memory usage during correlation API call

As shown in both figures 44 and 45, the CPU and the memory usage are almost constant disregarding the portion of the field used for the correlation analysis, i.e., using only 10% of the field cells or all the cells do not impact on the usage of the resources. In particular, the CPU usage is always below 80% and the memory usage is constantly below 5 GB. The average latency of each correlation API call is of about 230 milliseconds, with 4.3 requests per seconds on average answered by the platform, and about 1300 requests answered by each scenario in the 5 minutes of the experimental test.

In conclusion to our experimentation for this step, we highlight that the current data volume is not big enough for stressing the scalability of the platform. Indeed, we do not observe any increase in the latency or in the resource usage by using only 10% of the cells of the experimental field compared to using the full field. We plan to test the scalability of the platform for this step in the D7.3, with artificial generated data simulating the same patterns observed in real data with an increase in the data volume.

3.3.4 End to End Report

In this section we conducted a thorough experimentation on the steps the data provided by the Table and Wine Grapes pilot follow inside the BDG stack. The initial step of the dataset upload shows excellent performance and scalability regardless of the increase in concurrency. Since the CPU and network usage show minor increases as we increase the concurrency, we consider this step as a highly performant one. The data pipeline step also shows very good performance as far as the real-life scenario is concerned. Following the conclusions, we also came up for the Farm Management pilot higher concurrency should be employed for this step to overcome high execution time as the volume of data increases. In terms of rdfizing the data, as we observed for the previous cases as well, the steps shows good performance using the respective command line tool. The extraction of the semantic enriched data is the one showing the poorest performance for this specific pilot. As we have also described in the previous cases, we consider the increase in terms of concurrency to help greatly in improving the performance of this step so that no bottleneck is observed. Moreover, the extraction and storage of the satellite image processing dataset demonstrates great performance, keeping under consideration the low number of fields required to cover the pilot's needs. Finally, the analytic step shows good performance in terms of average latency of the correlation APIs, disregarding the quantity of data taken into account for the correlation. In the future we plan on further experimenting on this step by using artificial generated data as to stress the performance of the correlation task with an increased amount of data.

It should be noted at this point, as mentioned in the introductory section, that all of the described experimentation has been performed for the completed steps that are integrated into the BDG stack. Upon completion and integration of the rest, the experimentation will be updated to include the new additions.

3.4 Wine Making Pilot

This pilot deals with research in the fields of viticulture and oenology with a focus on the quality of the final product and its association with indicators collected from the field and the laboratory. The provided datasets include genetic data, weather and sensor data as well as laboratory experiments and field management activities. In the sections that follow, we analyze the provided datasets and heavily experiment on each of the identified data flow steps for this specific use case.

3.4.1 Dataset Analysis & Evaluation

As mentioned in the introductory section, this pilot has provided 8 different datasets. In the tables below we describe these datasets and evaluate them against the 4Vs of Big Data.

Table 18 Genetic data dataset

Dataset	Genetic data	
Metadata/Description	This dataset contains genetic data	
Provider	INRA	
Sample	Samples for these datasets can be found [here]()	
Evaluation	Big Data Vs	
	Volume	In terms of volume the provided dataset has ~2.6K rows, which is not a high value
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 7 different data types, along with metadata information
	Veracity	In terms of veracity, we will follow the methodologies suggested in D7.1, BDGS and BigDataBench, for the generation of large amounts of synthetic data that follow the statistical distribution of the raw data provided

Table 19 Soil characteristics dataset

Dataset	Soil Characteristics
Metadata/Description	This dataset contains data on soil characteristics
Provider	INRA

Sample	Samples for this datasets can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume the provided dataset has ~100 rows, which is not a high value
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 13 different data types, along with metadata information
	Veracity	In terms of veracity, we will follow the methodologies suggested in D7.1, BDGS and BigDataBench, for the generation of large amounts of synthetic data that follow the statistical distribution of the raw data provided

Table 20 Plot management dataset

Dataset	Plot Management	
Metadata/Description	This dataset contains data on plot management	
Provider	INRA	
Sample	Samples for this datasets can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume the provided dataset has ~400 rows, which is not a high value
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 32 different data types, along with metadata information
	Veracity	In terms of veracity, we will follow the methodologies suggested in D7.1, BDGS and BigDataBench, for the generation of large amounts of synthetic data that follow the statistical distribution of the raw data provided

Table 21 Climatic data dataset

Dataset	Climatic Data	
Metadata/Description	This dataset contains climatic data	
Provider	INRA	
Sample	Samples for this datasets can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume the provided dataset has ~2.5K rows, which is not a high value
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 58 different data types, along with metadata information
	Veracity	In terms of veracity, considering the high variety of this dataset we do not see the need for synthetic data generation

Table 22 Laboratory analysis dataset

Dataset	Laboratory Analysis	
Metadata/Description	This dataset contains the results of laboratory analysis	
Provider	INRA	
Sample	Samples for this datasets can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume the provided dataset has ~1.6K rows, which is not a high value

	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 95 different data types, along with metadata information, spread across 4 datasets
	Veracity	In terms of veracity, considering the high variety of this dataset we do not see the need for synthetic data generation

Table 23 Winemaking activities dataset

Dataset	Winemaking Activities	
Metadata/Description	This dataset contains data on winemaking activities	
Provider	INRA	
Sample	Samples for this datasets can be found here	
Evaluation		
	Big Data Vs	
	Volume	In terms of volume the provided dataset has ~550 rows, which is not a high value
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 59 different data types, along with metadata information, spread across 4 datasets
	Veracity	In terms of veracity, considering the high variety of this dataset we do not see the need for synthetic data generation

Table 24 Sensory analysis dataset

Dataset	Sensory Analysis
Metadata/Description	This dataset contains data from sensory analysis
Provider	INRA

Sample	Samples for this datasets can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume the provided dataset has ~3.6K rows, which is not a high value
	Velocity	The provided dataset is a static file and does not show a high velocity rate. However, for the needs of our experimentation we can mock the desired velocity
	Variety	In terms of variety the provided dataset tracks 12 different data types, along with metadata information, spread across 2 datasets
	Veracity	In terms of veracity, considering the combination of volume and variety we do not see a high need for synthetic data generation

On an overall analysis of the provided data we observe a very high variety value, having a total of 276 different data types, spread across 14 datasets. The volume of the provided data is lacking, having a total of ~11K records. To ameliorate this, following the methodologies described in D7.1, we increased the volume of each dataset by a factor of 10, following the statistical distribution of each one so to carry out the experimentation with an aggregation of 2.7M data points, spread across 276 different data types and 14 datasets.

3.4.2 Data Flows

In this section we describe the workflows the Wine Making pilot follows inside the BDG stack, during which the experimentation will be performed. It should be noted that the identified steps are the currently completed and integrated ones into the BDG stack that support this specific use case.

Initially, similarly to the previous use cases the 14 datasets are uploaded into the BDG platform and are stored into our Elasticsearch instance. The next step for this pilot is to convert the data into RDF and store them into GraphDB for the semantic enrichment to take place. After that the enriched data is exported from GraphDB and ingested back to Elasticsearch. Following the storage of the data into the Elastic stack of BDG the data goes through two predictive and analytics steps, one is the uncertainty-aware visual component available here and the other one is the wine quality prediction available here. The latter two are not taken into consideration in the context of the current version of the experimentation since they have not been integrated into the BDG stack.

3.4.3 Experimentation

In this section we conduct our experimentations on the provided datasets for each of the identified steps of the data flows they follow inside the BDG stack. For our experimentation we use our synthetically generated datasets that follow the statistical distributions of the provided ones and will test three usage scenarios per step.

3.4.3.1 Dataset Upload

In this step all of the datasets provided are uploaded into the BDG stack. Similar to the approach described in section 3.3.3.1, we will showcase three usage scenarios. One in which 50% of the datasets will be uploaded sequentially, another where all of the datasets will be uploaded sequentially and a final one involving the upload of all of the datasets concurrently.

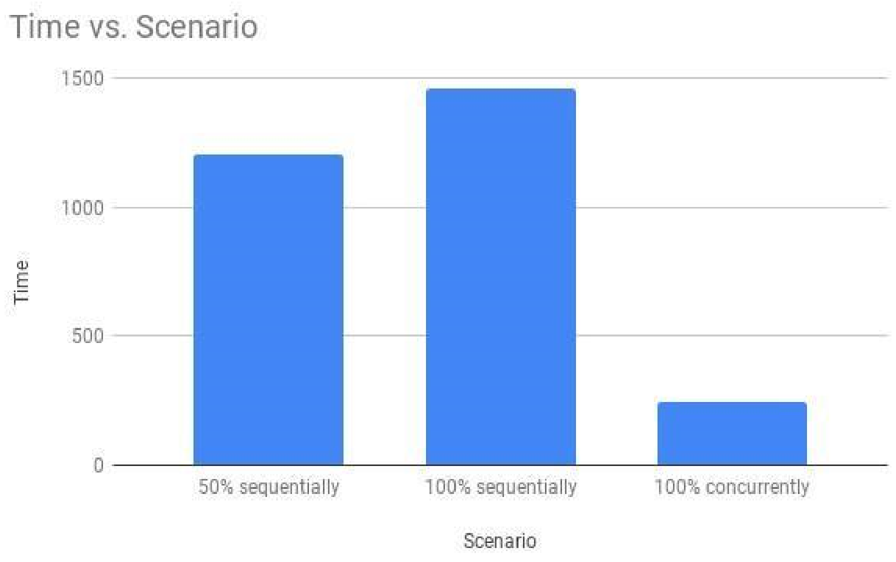


Figure 46 Completion time for dataset upload

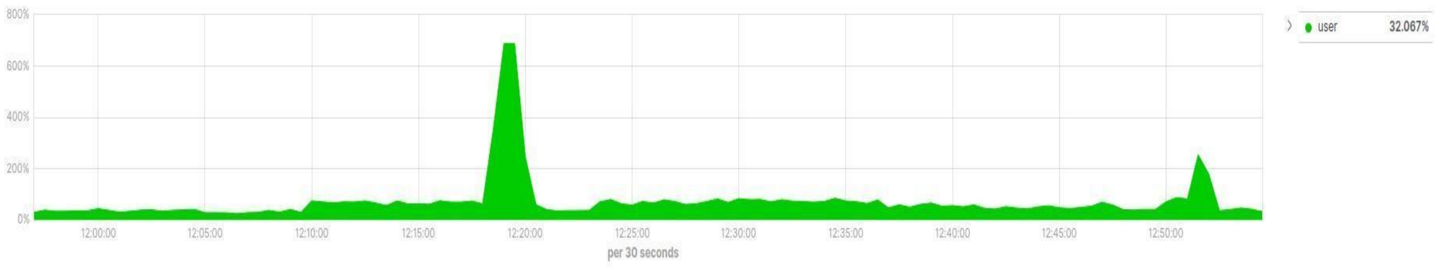


Figure 47 CPU usage during dataset upload

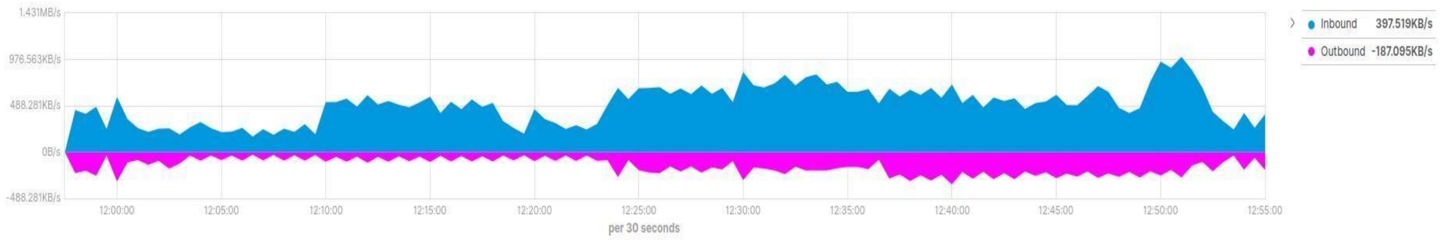


Figure 48 Network usage during dataset upload

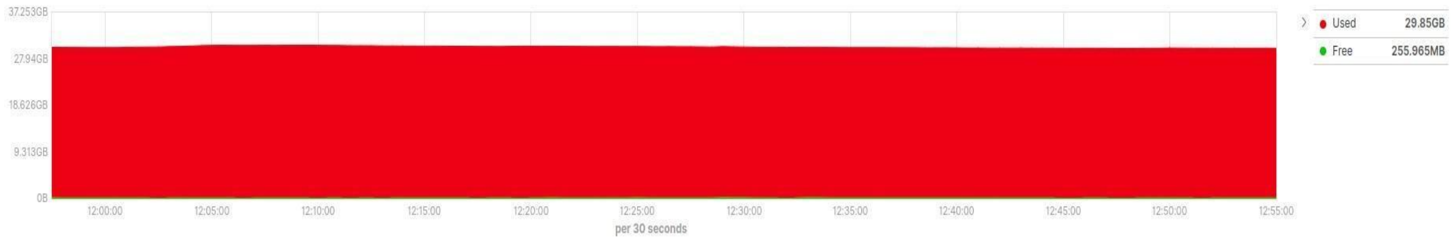


Figure 49 : Memory usage during dataset upload

As shown in the figure 46 above, the first scenario takes approximately 1,200 seconds, while for the second one we observe an increase of 300 seconds, in the time required for completion. The last scenario where all of the data is ingested concurrently shows a great decrease in terms of completion time, taking 245 seconds to finish. In terms of CPU (figure 47) we observe a steady increase across the 3 scenarios starting at 35% for the first, 80% for the second and resulting in a 250% for the last one. The same behavior is also observed for the network traffic (figure 48) of the stack, with an average increase of 350KB/s per scenario, starting from 350KB/s and leading to 1MB/s for the last one. Finally, the memory usage (figure 49) shows a very steady value across all scenarios.

In conclusion to the experimentation for this step, we see that the performance of the last scenario with a high concurrency value and volume performs very well with the deployed BDG stack, resulting, as is the case with other use cases for this specific step as well, to a high parallelization of the involved components.

3.4.3.2 Rdfization & Semantic Enrichment

In section 3.3.3.3 we thoroughly experiment on this step with higher volume datasets so do not conduct the same experiment here with the provided datasets.

3.4.3.3 Ingestion of Semantically Enriched Data

This step concerns the extraction of the semantically enriched data from the GraphDB instance of the stack and storage of this data into the Elasticsearch one. We split this experimentation into 3 scenarios. In the first one we will monitor the performance concurrent ingestion of 10% of the synthetically generated data, in the next one we will ingest 50% of it and on the final one we will extract and ingest all of the 2.7M data points. All of the scenarios were executed using 100 concurrent requests with batches of 10000.

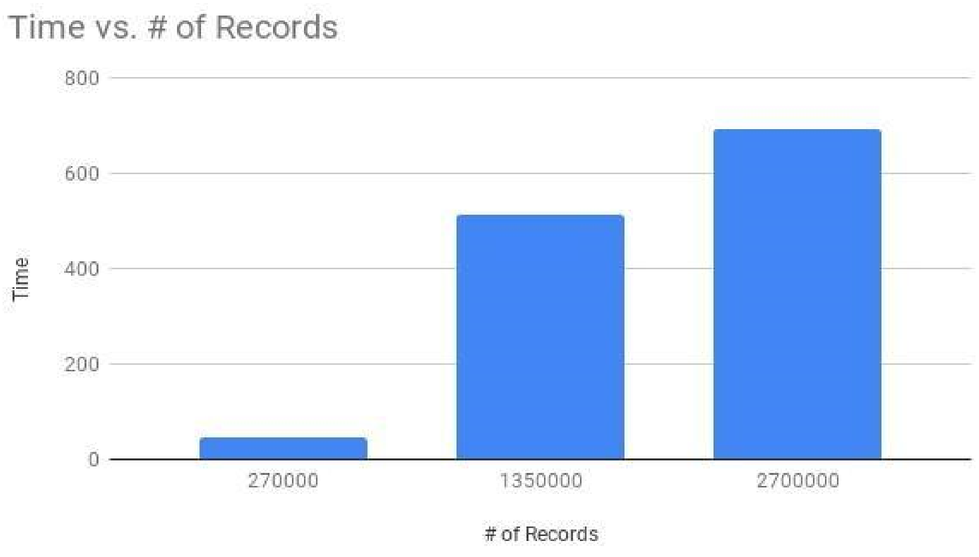


Figure 50 Completion time for the ingestion of semantically enriched data

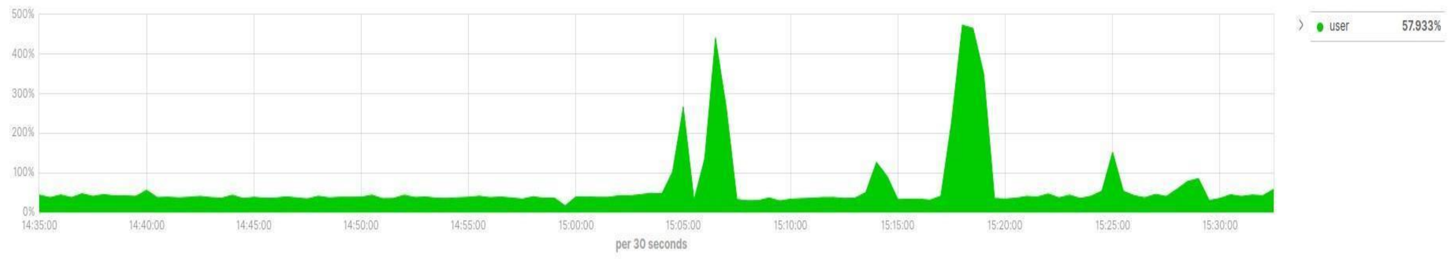


Figure 51 CPU usage during the ingestion of semantically enriched data

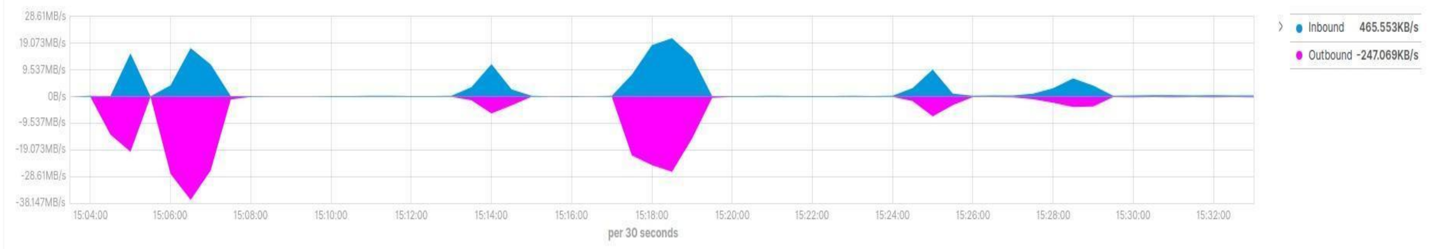


Figure 52 Network usage during the ingestion of semantically enriched data

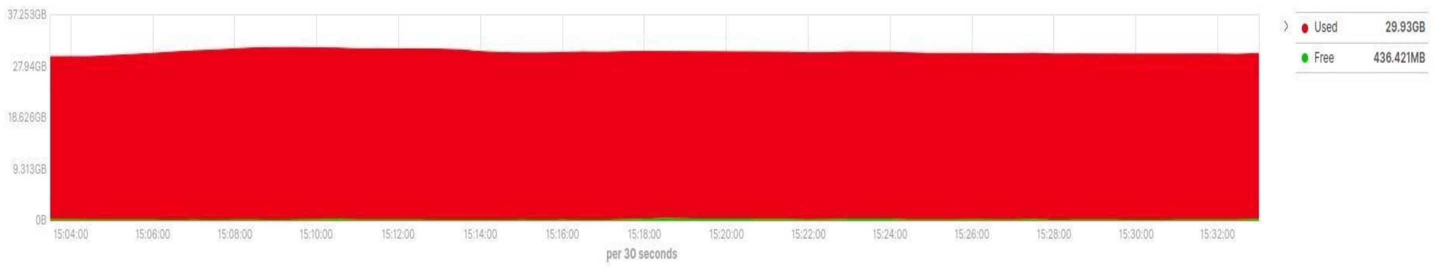


Figure 53 Memory usage during ingestion of semantically enriched data

The figures shown above present the behaviour of the monitored metrics for the three chosen scenarios for this specific step. The first one, where 270K of records are extracted from GraphDB and stored into Elasticsearch, was completed in 46 seconds (figure 50) with an average CPU (figure 51) usage of 265% and a network traffic (figure 52) of 15MB/s. For the ingestion of 1.35M of records, 512 seconds were required for the completion, with a CPU usage peak of 440% and we also observe a minor increase in terms of network traffic, leading to an average of 17MB/s for this scenario. Finally, the last scenario, where all of the data are extracted and inserted into Elasticsearch, 692 seconds are required for the completion, while a small increase of CPU usage is observed (on average 470%). In terms of network traffic, this scenario shows similar behaviour to the previous, having 18-20MB/s throughout this experimentation. The memory (figure 53) usage remains on the same levels for all the chosen scenarios.

Similarly to other use cases employing this step, we observe very high usage of all the metrics throughout the BDG stack, which leads us to create background tasks that frequently sync the data between the two frameworks.

3.4.3.5 Prediction of Leaves

The machine learning step related to this pilot concerns the development of a machine-learned pipeline aiming at counting leaves from side-view grapevine images taken into the imaging cabin of the PhenoArch platform. We identify the following usage scenario for this step: in the first one we will call the prediction API with 5,425 side-view images of grapevine, which is 10% of the total dataset. In the second scenario, we will move on with the usage of 50% of images, i.e., 27,124 images. In the final usage scenario, we will employ all of the 54,248 grapevine images. It should be noted that we used the images taken in the 2012 experimentation of the PhenoArch platform. Similar numbers were observed on the 2013 version. Given the machine learned solution developed for this particular pilot is based on a Deep Neural Network solution, a model that is know from the literature to be particularly efficient when scored using a Graphics processing unit (GPU) due to the enhancement in terms of time performance of the matrix multiplication operation, we collected the several metrics shown below on a machine equipped with two Intel(R) Xeon CPU E5-2630 v3 @ 2.40GHz (dual CPU, 8 cores with hyper-threading, 32 total concurrent threads), 192 GB of DD3 RAM and a Nvidia TITAN Xp GPU (12 GB of GDDR5 memory, 3840 Nvidia CUDA cores).

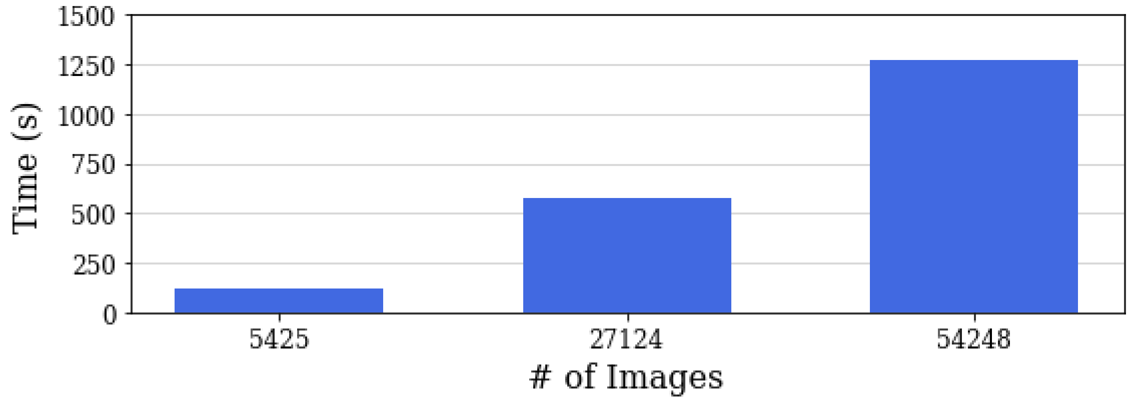


Figure 54 Completion time of leaves prediction

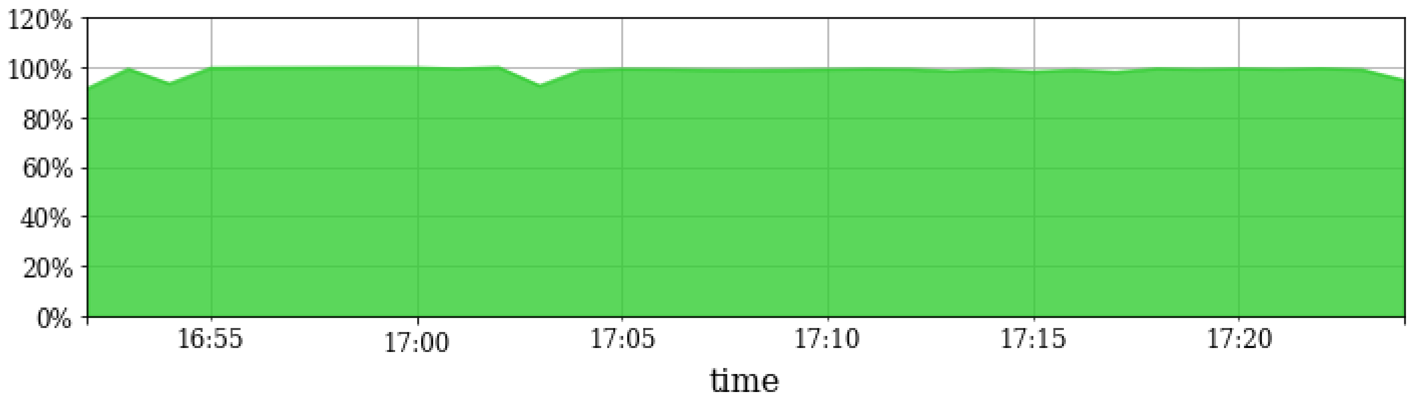


Figure 55 CPU usage during prediction of leaves

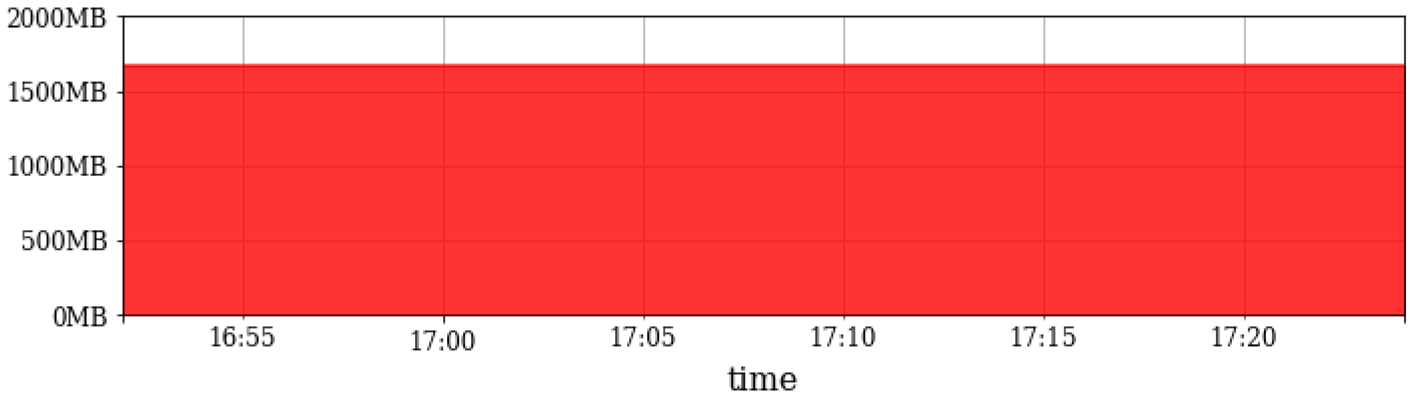


Figure 56 Memory usage during prediction of leaves

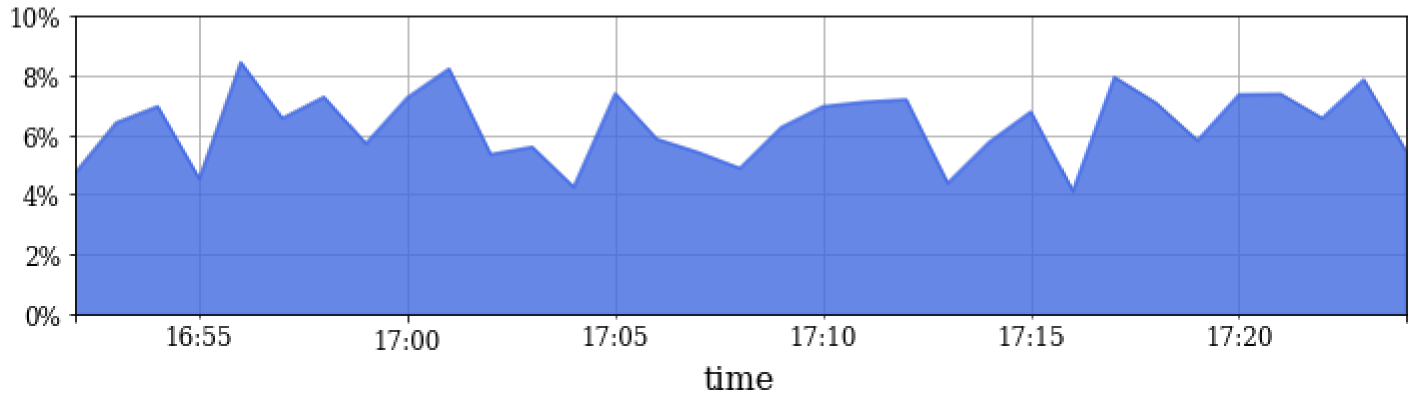


Figure 57 GPU usage during prediction of leaves

As shown in figure 54, the completion time increases linearly as we supply more data to the prediction API. In particular, 120 seconds are required to predict the leaf count of 5,425 images (corresponding to 10% of the dataset), 575 seconds for 27,124 images (50% of the dataset) and 1,272 seconds for all the 54,248 images composing the full dataset. The full pipeline consisting in reading the image from the file system, preprocessing it (cropping and lowering the resolution), and feeding it to the GPU for prediction employs thus on average only 32 milliseconds per image. By analyzing the CPU usage of the component, we can conclude we are saturating the CPU (100% of the available computational power) regardless of the size of the provided dataset. This can be explained by the fact that reading the images from the file system (I/O bound) and preprocessing it (CPU bound) are operations performed in parallel by the API component, thus exploiting all the computation resources available in the CPU (multiple CPUs, each one with several cores with hyper-threading technology). A similar consideration allows us to conclude that the memory usage of the component is steady regardless of the dataset size, because we are already exploiting the full parallelism of the processor. Finally, in figure 57 we observe that the GPU usage is fluctuating in between 4% and 8%, thus highlighting that we are not fully exploiting the computational power provided by the graphic unit. This effect can be explained by the fact the preprocessing operations are quite heavy, in particular reading the image from the file system, given each image is saved on a file of about 13Mb. These heavy operations, confirmed by the saturated CPU usage plot, confirms that the inference time of the model is only marginally affecting the efficiency of the component, while the preprocessing steps, despite fully exploiting the parallelism of the machine, represents a bottleneck in the current architecture.

In conclusion to our experimentation for this step, we consider the last usage scenario as the real-life one. Prediction workflows are heavily dependent on the volume of the input datasets, so the last usage scenario best simulates the desired production state of this step. The execution time required for the prediction of the number of leaves from a side-image is compatible with the requirements of the task, given 32ms are needed for the full pipeline evaluation while taking a single grapevine photo would require much more time. However, it would be possible to improve this execution time by pre-processing the photos before saving them on the file system. This operation will result on the one hand in a reduction by a large extent of the completion time of the task, and on the other hand in a better usage of the GPU power with a more efficient exploitation of the graphics unit.

3.4.4 End to End Report

In this section we experimented on the datasets provided by the Wine Making pilot and the datasets it has provided. In terms of the data flows specific for this pilot we have identified that the initial step the dataset upload step, presents great performance in respect to the completion time as well as the CPU, network and memory usage. It is a step that can be easily made with a high degree of concurrency without seriously affecting the rest of the stack. Following the experimentation, we performed on the rdfization step, we consider this step to have a good performance regardless of the volume of the data, using the command line tool developed for this step. Interestingly for the step, that of the semantically enriched data extraction and storage into Elasticsearch, we observed a slightly different behaviour than the other pilots. This difference can be explained considering the differences in the data model for each pilot and the better performance of this step is directly affected by the work done in D3.1. Finally, also the prediction step shows very good performance, taking into consideration the difficult nature of the task and the complexity of the prediction model

It should be noted at this point, as mentioned in the pilot's introductory section, that all of the described experimentation has been performed for the completed steps that are integrated into the BDG stack. Furthermore, due to the lack of Big Data coverage for the provided datasets, the experimentation was performed with high amounts of synthetic data, which although follow the statistical distribution of the provided may decrease the accuracy of the results.

3.5 Food Protection Pilot

Working with real industry scenarios we define which are the main parameters that need to be predicted. Based on these findings we test several machine learning and deep learning algorithms to predict parameters like the chemical risk, pricing and fraud. The goal is to combine different datasets to achieve the optimum performance for the prediction of the risk in raw materials (e.g. grapes) and finished products (wine). A large number of different data sources and data types has been used. We used textual information that includes mainly announcements about food recalls and border rejections (FDA, RASAFF, FSSA, AUSTRALIAN FOOD SAFETY AUTHORITY). We also use numeric data that is lab test and prices. The main source of our datasets is the open data published by the governments. Moreover, private data that includes ingredients that a company uses for food production, list of suppliers and internal lab test. Risk assessment module, price prediction dashboard and recall prediction have developed.

3.5.1 Dataset Analysis & Evaluation

As mentioned in the introductory section, this pilot has provided 3 datasets. One is the lab tests performed on food ingredients, food incidents like recalls and border rejections and the food prices data. In the tables below we describe these datasets and evaluate them against the 4Vs of Big Data.

Table 25 Laboratory tests dataset

Dataset	Laboratory Tests
Metadata/Description	This dataset contains the laboratory tests performed on food ingredients
Provider	Agroknow
Sample	Samples for this dataset can be found here
Evaluation	Big Data Vs

	Volume	In terms of volume the provided dataset has ~91.185.000 rows
	Velocity	The provided dataset is announced on a yearly basis by each Food safety Authority worldwide
	Variety	In terms of variety the data are of a specific type with specific attributes and in an xls file following a different schema for each authority
	Veracity	In terms of veracity, the data is considered to cover well Big Data aspects so no need to generate more exists

Table 26 Food price dataset

Dataset	Food price	
Metadata/Description	This dataset includes food price data for more than 800 products	
Provider	Agroknow	
Sample	Samples for this dataset can be found here	
Evaluation		
	Big Data Vs	
	Volume	In terms of volume the provided dataset does not show a high value (~388.000 rows)
	Velocity	The provided dataset is Announced on a daily/weekly and yearly basis by Hellenic Food Market, European Commission and FAO
	Variety	In terms of variety the data are of a specific type with specific attributes and following a different schema for each authority
	Veracity	In terms of veracity, considering the variety we don't see a high need for synthetic data generation

Table 27 Food incidents dataset

Dataset	Food incidents	
Metadata/Description	This dataset contains the food ingredients (recalls and border rejections)	
Provider	Agroknow	
Sample	Samples for this dataset can be found here	
Evaluation	Big Data Vs	
	Volume	In terms of volume the provided dataset has ~450.000 rows
	Velocity	The provided dataset is Announced on a yearly basis by each Food safety Authority worldwide
	Variety	In terms of variety the data has a variety types (food recalls happening of a market level, border rejections, information for attention etc) and with specific attributes following a different schema for each authority
	Veracity	In terms of veracity, the data is not considered to cover well Big Data aspects so need to generate more

As described in this section the need for synthetic data generation has been identified. We perform this generation following the methodologies described in D7.1.

3.5.2 Data Flows

In this section, we describe the workflows food protection pilot follows inside the BDG stack, during which the experimentation will be performed. It should be noted that the identified steps are the currently completed and integrated ones into the BDG stack that support this specific use case.

First, the dataset is uploaded into the system, either by using the provided UI (User Interface) per pilot, or by using the specific API endpoint as described in D6.2. Finally, all of the data that resides into Elasticsearch is exported and used for the prediction experiments components and visualizations by querying the specific API endpoint.

3.5.3 Experimentation

In this section we perform the rigorous testing experimentation for each of the identified steps, using data are stored in the BigDataGrapes platform. As described in the methodology section we will perform our experimentation using three scenarios for the dataset upload, 20 different cases for recall prediction experiment and three different cases for Price prediction experiments and risk assessment experiments to better showcase the scalability potential of the deployed stack.

3.5.3.1 Dataset Upload

To experiment on this step, all the recall datasets provided are uploaded into the BDG stack. One in which 25% (~115K) of the datasets will be uploaded sequentially, another in which 50% (~225K) of the datasets will be uploaded sequentially in 5 batches with 45K per batch, and a final one where all (~450K) of the datasets will be uploaded sequentially in 5 batches with 90K for each batch.

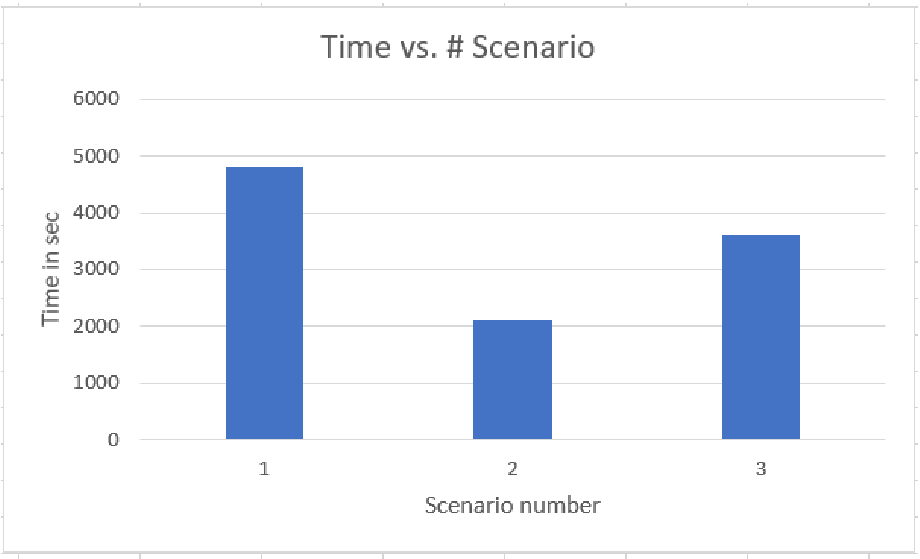


Figure 58 Completion time for dataset upload

Figure 58 shows the completion time (second) against the concurrency of the requests to the stack. As it is demonstrated in this figure, completion time shows a high increase in the first scenario where 25% of the dataset uploaded row by row. Moreover, the second-high increase is observed in the third scenario where all the dataset uploaded using 5 concurrent requests. The second scenario where the 50% of the dataset uploaded using 5 concurrent requests. We conclude that the second scenario has the best efficiency in terms of time.

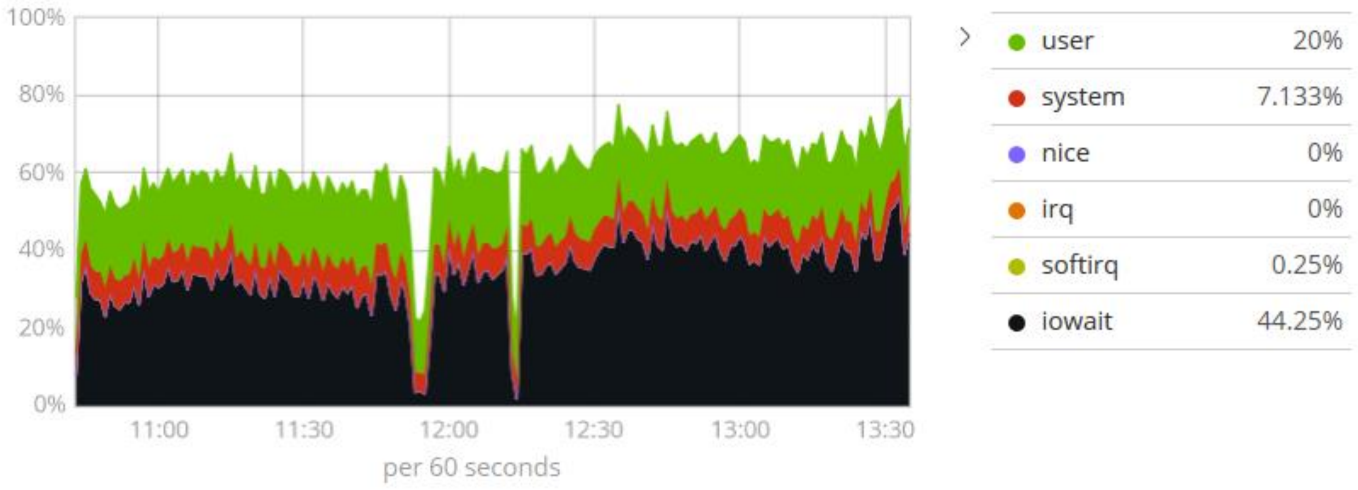


Figure 59 CPU usage during dataset upload

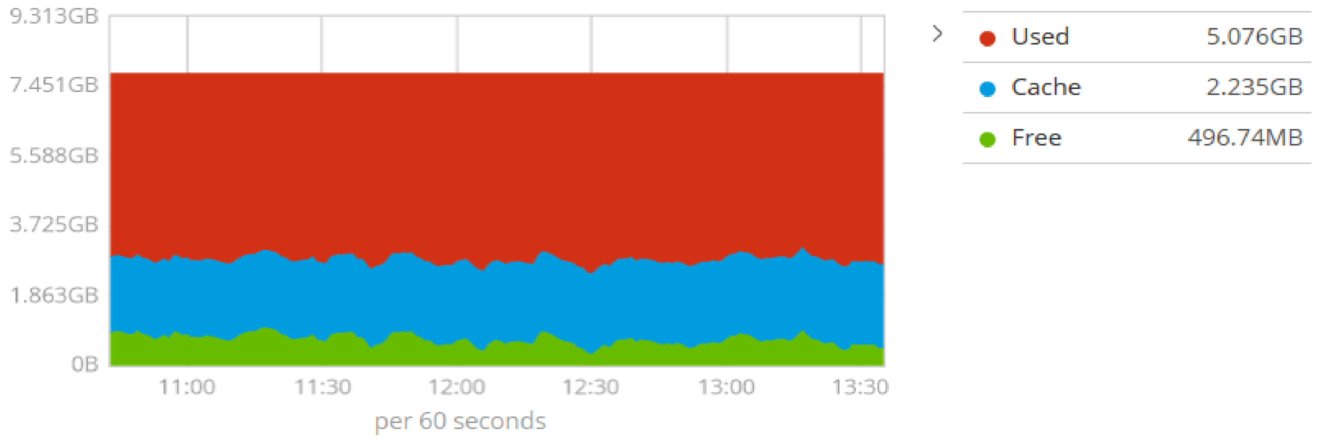


Figure 60 Memory usage during dataset upload

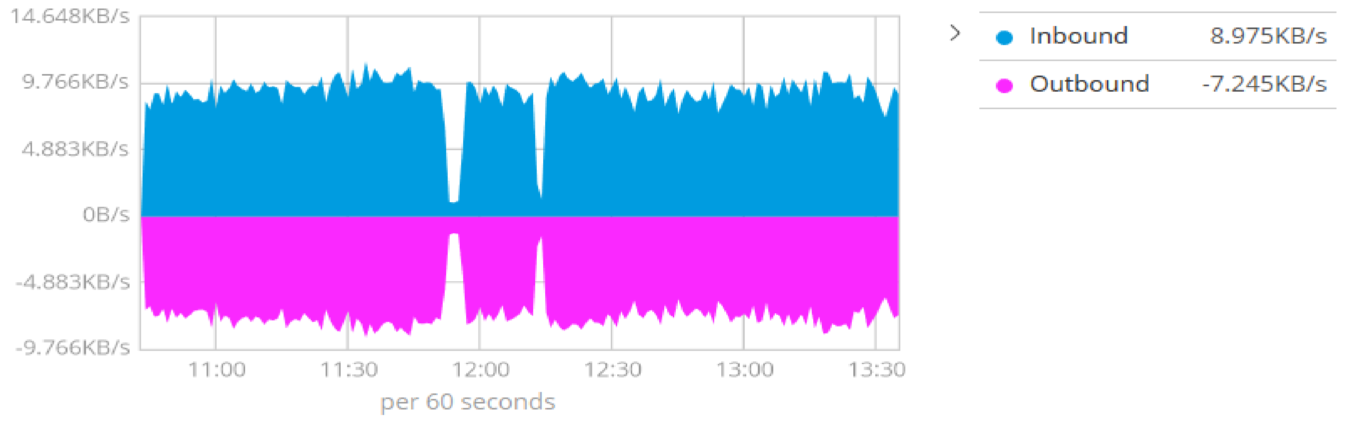


Figure 61 Network usage during dataset upload

The figures shown above depict the CPU usage (figure 59), network traffic in bytes (figure 61), and memory (figure 60) used by the stack, throughout the execution of the three scenarios. We observe a slight increase in terms of CPU usage between the scenarios starting from an average of 60% for the one with a sample of the data but for storing them row by row, 65% for the next one and an average of 75% for the last case with all the dataset sending using 5 concurrent requests, with a peak of 80%. In terms of network traffic, we observe a steady rate of inbound traffic for the 3 scenarios and an average of 9.700KB/s. Finally, stable memory usage is observed which is due to the fact that most of the components in the stack are based on the JVM (Java Virtual Machine) which can be set to reserve the required memory upon startup.

We conclude our experimentation on this step by observing that the best usage of the platform taking into account all of the monitored metrics is the second one, where small amounts of data are ingested into the stack using batches of data per request. To that end we change the initial ingestion of data for this pilot to have its datasets uploaded in small batches of concurrent requests

3.5.3.2 Rdfization & Semantic Enrichment

In section 3.5.3.3 we thoroughly experiment on this step with higher volume and variety datasets so do not conduct the same experiment here with the provided datasets.

3.5.3.5 Price prediction API

The purpose of this step is to develop a machine learned solution aimed at predicting the future price of specific goods in the grapes and wines supply chain based on their historical price in the past seven days. The dataset used for this step is made up of the historical price of 14 products. We identify the following two scenarios for this step: in the first scenario, we will predict the price of a specific good out of the 14 goods of the dataset. In the second scenario, we will predict the price of all the 14 goods in the dataset. Given the prediction API is very fast to answer, we simulated 5 minutes of sequential API call with the first scenario and 5 minutes of se API call with the second scenario, with a parallelism equals to the number of products in the dataset, i.e., 14 threads.

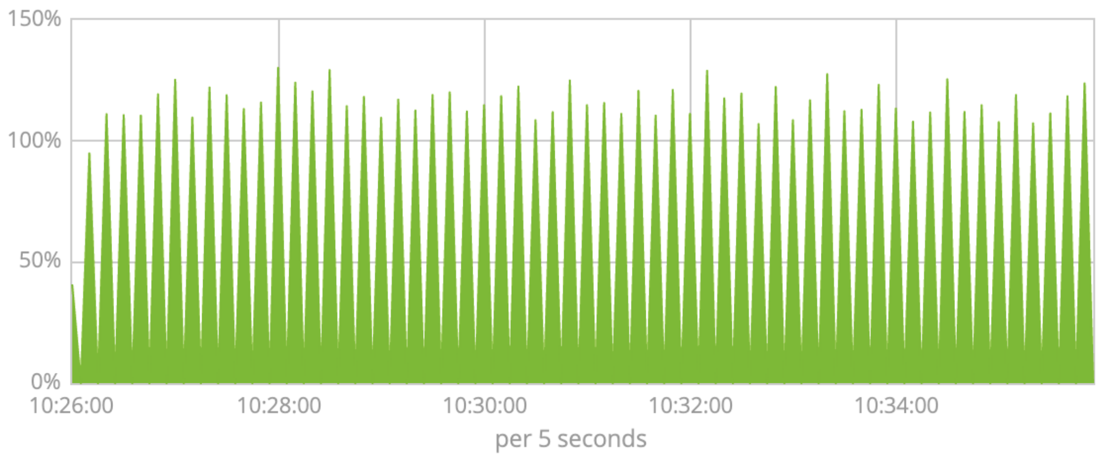


Figure 62 CPU usage during price prediction

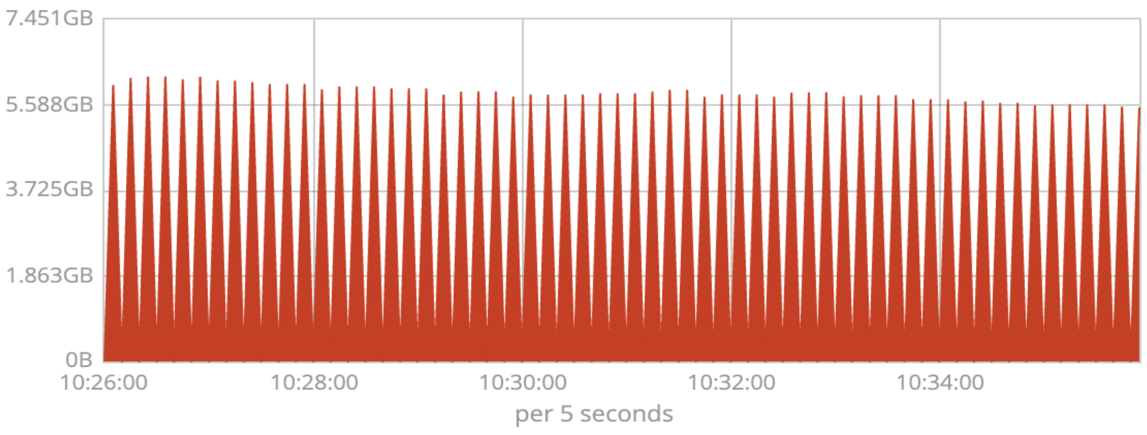


Figure 63 Memory usage during price prediction

The figures above highlight that the two scenarios use a constant amount of CPU and memory, in particular the CPU (figure 62) usage is always in between 110% and 130% and the memory (figure 63) usage is in between 5.6GB and 6.1GB. The average latency of each correlation API call is of about 850 milliseconds, with about 1.2 requests per seconds on average answered by the platform, and about 360 requests answered by each scenario in the 5 minutes of the experimental test. The high latency is due to the complexity of the model (LSTM neural network), which could easily benefit from the usage of a Graphical Processing Unit (GPU) for the inference part, lowering the latency. An additional insight is represented by the average latency per product, which is constant as well. This evidence is due to the architecture of the neural network, which forces the inference time to be independent from the specific product at hand. The resulting latency for the second scenario is thus 14x of the first scenario, given the prediction time for each product is constant and the products are 14.

In conclusion to our experimentation for this step, we highlight that the prediction workflows are heavily dependent on the volume of the input datasets, i.e., the number of products for which to predicts their future price. There is a constant usage of the resources for the predictions of each product, and the resulting latency can be lowered by using more appropriate hardware (GPU or TPU) in place of the CPU.

3.5.3.5 Incidents prediction API

The aim of this step is to use pilot's dataset for experimental purposes and in particular for the case of the incident's prediction. Food incidents data is used to train a deep learning prediction model to predict food incidents in the future. The Incident's dataset becomes available through the Search API (D6.2). First, we sent a request to search API with the product we want to predict, for example, grapes and then we filter the result by selecting a country, for example, Greece or/and a specific hazard for instance biological. The resulting sub dataset is given as an input to the Incidents prediction API which is used to train the model. The prediction API results are future incidents for this product. For this experiment, we test the platform using the trend calculation and search endpoints. On the one hand, the trend calculation endpoint includes the training of the prophet model and the prediction of the food recalls in the next 12 months. On the other hand, the search and point include a search query to the database to return the prediction results.

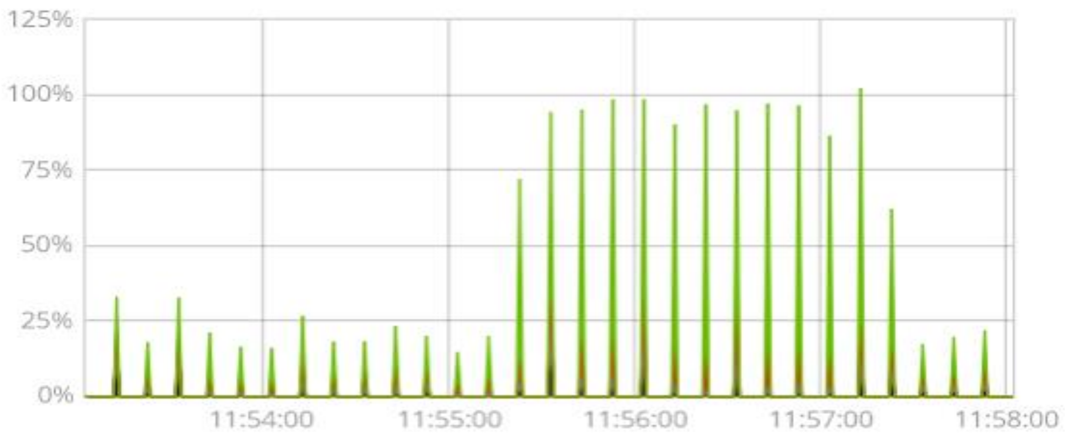


Figure 64 CPU usage during using recall prediction API

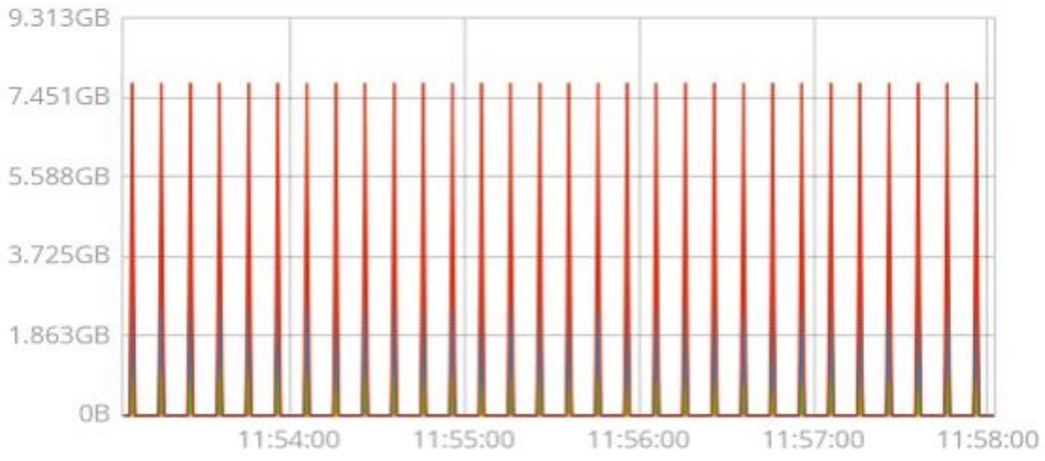


Figure 65 Memory usage during recall prediction API

For this experiment, all recall data as described above was used. The experiment requires 5 seconds to complete, in a very short time if we taking as a consideration that this API includes model training. In particular, 29 requests are made to test the recall API based on 29 different product categories. Each product category includes 15K data points average. We observe that the Memory figure does not consist of a straight line. This makes a lot of sense due to the fact that the requests for the endpoint *calculate-trend* and the *search* take place alternately for each product category. The *calculate-trend* endpoint takes more time to run because it includes the training of the model while the *search* endpoint includes a search query. For this reason, there is this change in the figure line. The picks of the line correspond to *calculate-trend* endpoint requests while the lower points of the line correspond to search endpoint requests. A similar behavior is observed in CPU (figure 64) and memory usage figures (figure 65). The training of the model requires more computing power. For this reason, the use of CPU when training the model can exceed 100%. On the other hand, in the memory usage figure, we notice a recurring change with a maximum value of 7.5 GB. In both cases, the results are expected due to the alternation of the endpoints mentioned above.

In conclusion to our experimentation for this step, we consider this experiment usage scenario as the real-life one. Prediction workflows in terms of testing are heavily dependent on the volume of the input datasets and in this API max three parameters are needed, that is product, hazard, and origin

Risk assessment module

In this step for food incidents datasets is to use them for experimental purposes and in particular for the case of the Risk assessment module. The main goal of the risk assessment module is to help Food Safety Experts to identify the ingredients with unacceptable hazard risk. To experiment on this step, we defined three scenarios. The first scenario includes a request to Risk API in order to calculate the risk for a specific ingredient. The second includes a request to Risk API to calculate the risk for a specific ingredient, origin, and hazard. Finally, the last scenario includes a request to Risk API to generate a batch of risk time series.

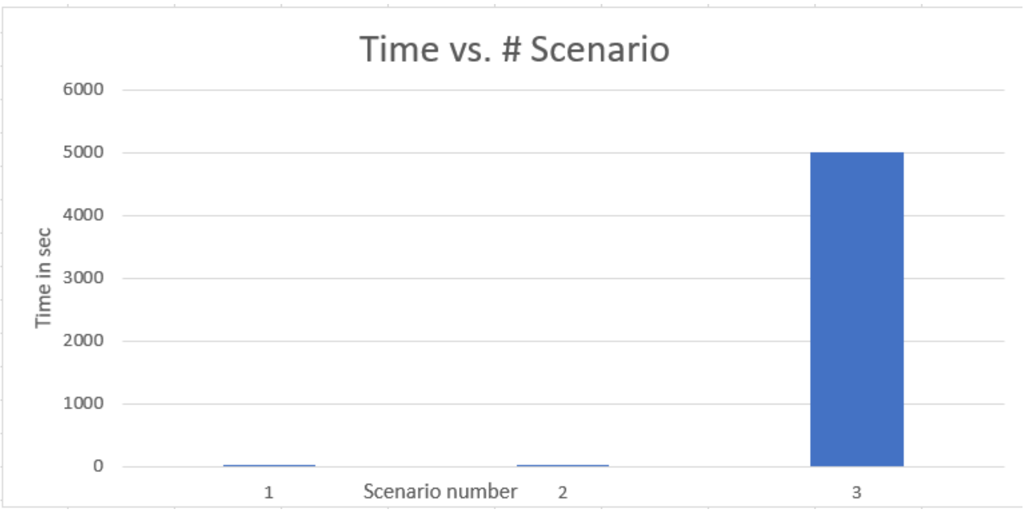
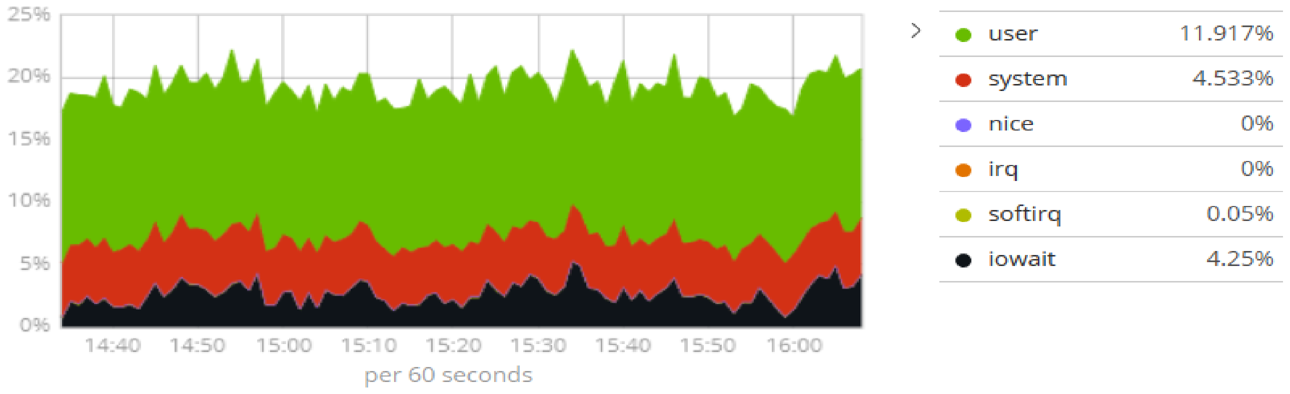
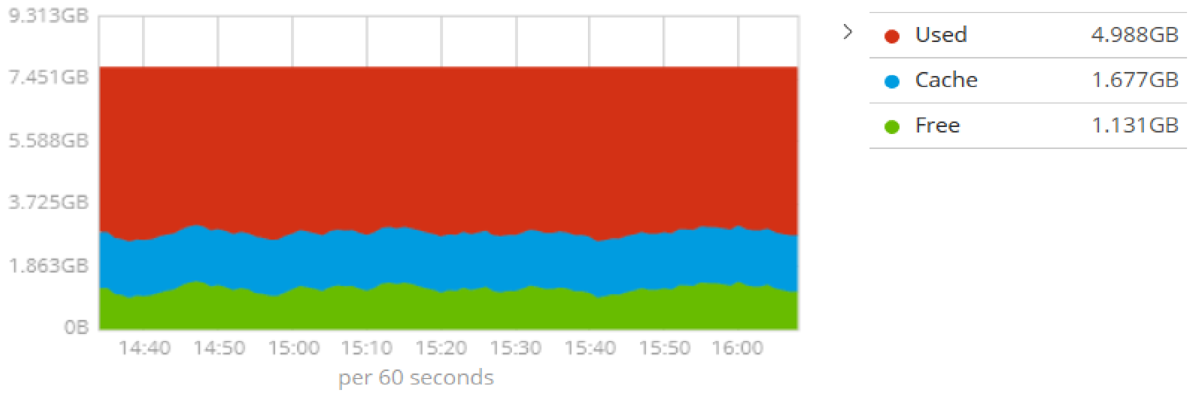


Figure 66 Completion time for risk assessment



●	user	11.917%
●	system	4.533%
●	nice	0%
●	irq	0%
●	softirq	0.05%
●	iowait	4.25%

Figure 67 CPU usage during using risk assessment



●	Used	4.988GB
●	Cache	1.677GB
●	Free	1.131GB

Figure 68 Memory usage during risk assessment

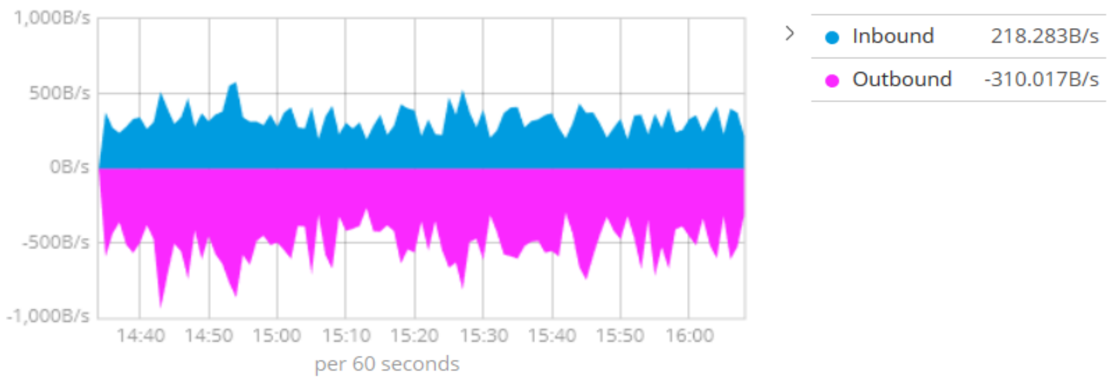


Figure 69 Network usage during risk assessment

For this experiment, a sub recall dataset as described above was used. The experiment requires 5100 seconds to complete, in a short time if we taking as a consideration that this API includes the generation of a batch risk time series. In particular, three requests are made to test the Risk API based on three different scenarios as mentioned above. The first two scenarios take less time to run because they include the calculation of the risk for one ingredient in the first scenario and the second one calculates the risk for a specific ingredient, hazard, and origin (figure 66). We can easily conclude that in the first two scenarios a sub dataset of the total volume of data is used. The last scenario includes the generation of time series risk that needs more time. In terms of Memory (figure 68), we observe constant use. This makes sense because the requests for the three scenarios take place one after another and use all memory resources with a maximum value of 7.5 GB. As mentioned above, a low value is observed, on average 20% (figure 67). Finally, in terms of network usage (figure 69), we observe a steady value of average 500KB/s for each of the scenarios employed and steady memory usage for the whole stack.

In conclusion to our experimentation for this step, we consider this experiment usage scenario as the real-life one. Risk assessment workflow in terms of testing are heavily dependent on the volume of the input datasets.

3.4.4 End to End Report

In this section we experimented on the datasets provided by the food protection .In terms of the data flows specific for this pilot we have identified that the initial step the dataset upload step, presents good performance in respect to the completion time as well as the CPU, network and memory usage. It is a step that can be easily made with a high degree of concurrency without seriously affecting the rest of the stack. Following the experimentation part, recall prediction API presents very good performance in respect to the completion time as well as the CPU, network and memory usage. Performance increased by lowering the volumes of data handled. Increasing data needs more time, more CPU and memory to be trained. The price prediction API as well shows good performance, with a constant latency per product and an overall latency that is dependent from the number of products in the dataset. The CPU and memory usages are constant as well, indicating the resources are not limiting the performance of the module.

It should be noted at this point, as mentioned in the pilot’s introductory section, that all of the described experimentation has been performed for the completed steps that are integrated into the BDG stack. Furthermore, due to well amount of Big Data coverage for the provided datasets, the experimentation was performed with real dataset and not synthetic data.

4 Experimentation workflow and panel for machine learning solutions

In the context of the BigDataGrapes, we developed an intelligent panel that facilitates the process of parameterization and testing of machine learning solutions. The panel is part of the Big Data Platform and implements an experimentation workflow for the execution pre-processing, parametrization and training of machine learning solutions. The intelligent panel (http://vision.bigdatagrapes.eu/) can be used to train and test several machine learning and deep learning algorithms, using datasets that are available in the Big Data Platform.

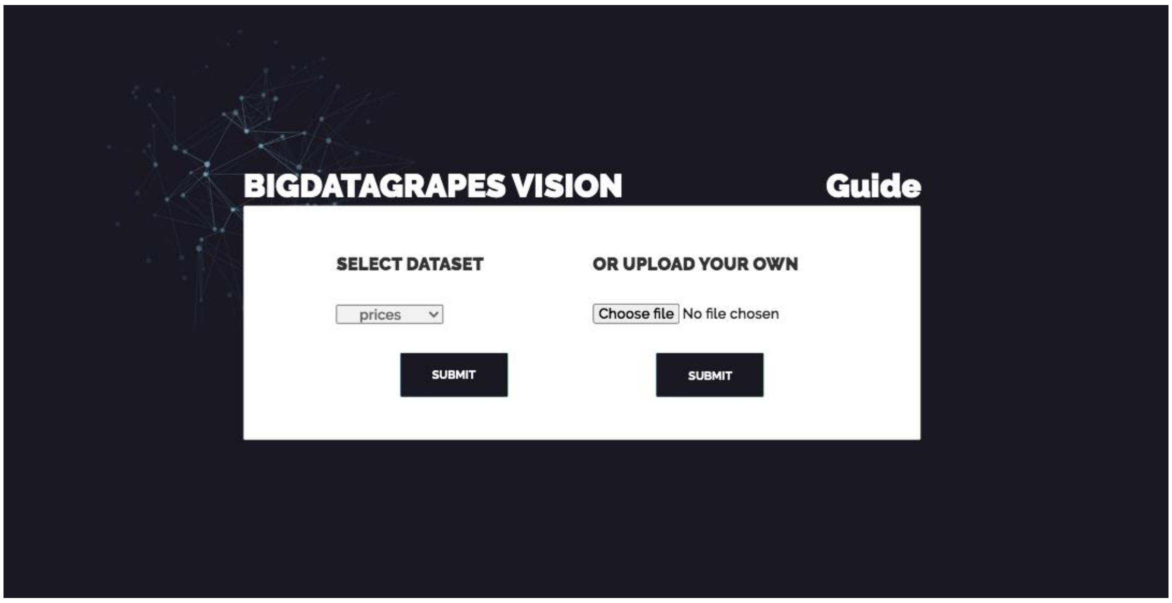


Figure 70: Intelligent panel of the BigDataPlatform

4.1 EXPERIMENTATION STEPS OF THE INTELLIGENT PANEL

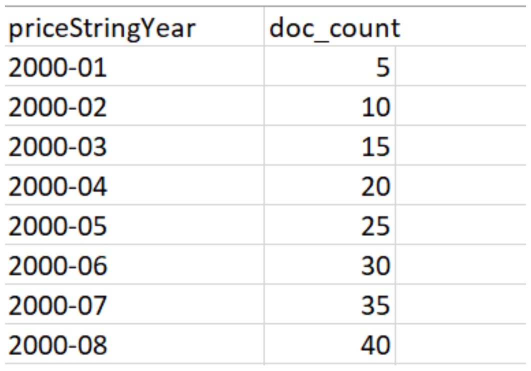
1. **Select Dataset:** The user picks a price or recall dataset that is provided by us or chooses a custom dataset.
2. **Craft your own experimentation dataset:** The user picks product details, data sources and country.
3. **Algorithm Selection:** The user picks an algorithm from the provided list.
4. **Algorithm Customization:** The user customizes the algorithm that they chose. Each algorithm has different parameters.
5. **Results:** The Results are displayed to the user. A metric (RMSE) and a plot depict the accuracy.

4.2 CUSTOM DATASET UPLOAD

You can upload your custom dataset provided it meets the conditions described below:

1. It is a .csv file. The platform will not accept any other form, for security and compatibility reasons.
2. It has a specific structure. In order for the models to work we need to have two columns that correspond to date and values

An example of a custom-made dataset is presented below:



priceStringYear	doc_count
2000-01	5
2000-02	10
2000-03	15
2000-04	20
2000-05	25
2000-06	30
2000-07	35
2000-08	40

4.3 MACHINE LEARNING MODELS

The Intelligent Panel includes the following models:

Table 28: Machine learning models available in the Intelligent panel

Model	Parameters to customize
LSTM	Number of epochs, loss parameters, metric parameters.
KNN	Number of neighbours, Train/Test percentage.
Random Forest Regression	Number of neighbours, Train/Test percentage.
Prophet	Months to forecast ahead.
Neural Prophet	Months to forecast ahead.

It should be noted that the models could be further customizable, however, an important portion is covered.

4.4 MONITORING AND MODEL SAVING

The intelligent panel uses **MLflow** (https://mlflow.org/) to keep track of our model accuracy and save our models with no manual work done. MLflow is an open source platform to manage the ML lifecycle, including experimentation, reproducibility, deployment, and a central model registry.

The MLflow Tracking component is an API and UI for logging parameters, code versions, metrics, and output files when running the machine learning code and for later visualizing the results. This means that each time the users selects an algorithm all the work is done by MLflow and the user must only visit its API to view the results.

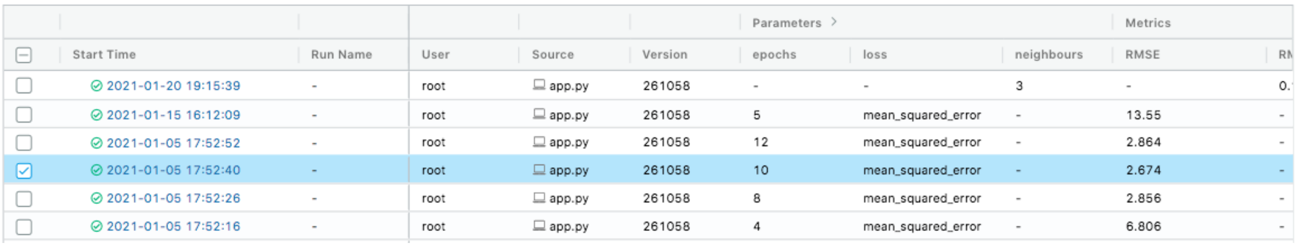
MLflow is an open source platform for managing the end-to-end machine learning lifecycle. It tackles four primary functions:

1. Tracking experiments to record and compare parameters and results (MLflow Tracking).
2. Packaging ML code in a reusable, reproducible form in order to share with other data scientists or transfer to production (MLflow Projects).
3. Managing and deploying models from a variety of ML libraries to a variety of model serving and inference platforms (MLflow Models).

4. Providing a central model store to collaboratively manage the full lifecycle of an MLflow Model, including model versioning, stage transitions, and annotations (MLflow Model Registry).

4.5 DEPLOYING MODELS ON THE BIG DATA PLATFORM

The main goal of following an experimentation workflow when developing a solution that is based on machine learning is to identify the best performing models for the business use case and the available data.

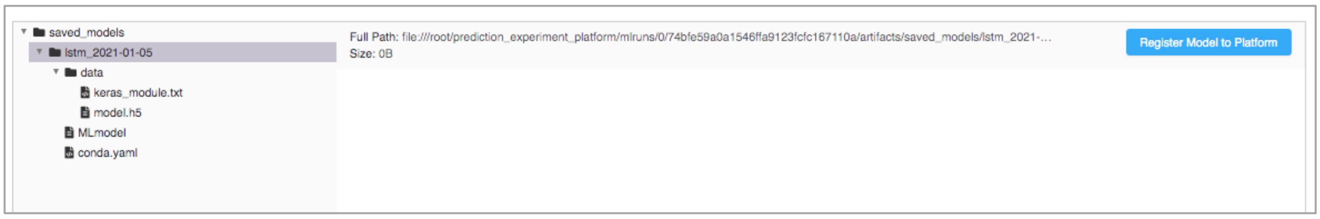


						Parameters >			Metrics	
⊟	Start Time	Run Name	User	Source	Version	epochs	loss	neighbours	RMSE	R
☐	✓ 2021-01-20 19:15:39	-	root	app.py	261058	-	-	3	-	0.
☐	✓ 2021-01-15 16:12:09	-	root	app.py	261058	5	mean_squared_error	-	13.55	-
☐	✓ 2021-01-05 17:52:52	-	root	app.py	261058	12	mean_squared_error	-	2.864	-
☑	✓ 2021-01-05 17:52:40	-	root	app.py	261058	10	mean_squared_error	-	2.674	-
☐	✓ 2021-01-05 17:52:26	-	root	app.py	261058	8	mean_squared_error	-	2.856	-
☐	✓ 2021-01-05 17:52:16	-	root	app.py	261058	4	mean_squared_error	-	6.806	-

Figure 71: Comparing different models based on their performance

All the developed models along with their parameters and metrics are stored in the Intelligent panel and can be studied and compared with other models through a user friendly user interface.

The best performing prediction models are deployed in the Big Data Platform and are available through the APIs of the intelligent layer to the prediction dashboards that are used by the food industry experts.



```
▾ saved_models
  ▾ lstm_2021-01-05
    ▾ data
        keras_module.txt
        model.h5
      MLmodel
      conda.yaml
```

Full Path: file:///root/prediction_experiment_platform/mlruns/0/74bfe59a0a1546ffa9123fcfc167110a/artifacts/saved_models/lstm_2021-...
Size: 0B

Figure 72: Save and register the prediction model to the Big Data Platform

5 Conclusions

In this document we performed rigorous experimentation of the provided datasets using the BDG stack. To measure the performance of the experimentation we used the methodology and a number of metrics suggested in D7.1 and described in the respective section of this document. We moved on analyzing each of the provided datasets against Big Data Vs and we generated synthetic data following the same statistical distribution where needed. For each pilot, we identified and abstractly described the data flows its datasets follow inside the BDG stack, with respect to the desired outcomes for each. Every completed and integrated step was then experimented upon using three different usage scenarios in an attempt to stress test the stack. Finally, for each pilot we created an end to end report on the observed performance of each of the steps and components it utilizes inside the stack. Furthermore, in this document we presented the intelligent panel that we developed in the context of the project to facilitate the experimentation with different datasets and algorithms.

We should note that due to the current version of the BDG stack being the final one, we plan to further expand our experimentation to D7.3 using projected datasets.

Table 29 Experimentation outcomes

Dataflow Step	Experimentation Outcomes	Suggestions
Sensor data ingestion / Data streams pipeline	Highest performance was observed as the parallelization was increased.	We suggest a high parallelization value for this step having as upper limit the number of data types collected.
Satellite image processing data ingestion	Highest performance was observed by having an increased concurrency value.	We suggest an increased concurrency value, up to the number of fields.
Dataset upload	Performance increased by lowering the volumes of data handled or by increasing concurrency.	We suggest either increasing the parallelization value or splitting the uploaded datasets into small batches.
Rdfization & Semantic enrichment	Lowest completion time observed with the usage of the command line tool.	We suggest this step to be scheduled as a cron job handling the creation or update/delete of data.
Ingestion of semantically enriched data	Best performance observed with low volumes of data.	We suggest this step to be scheduled as a cron job handling small batches of data.
Correlation of data	Performance was increased as the volume increased.	We suggest the split of this step into 2 substeps, triggering the correlation through offline jobs.
Prediction of data	Performance increased by lowering the volumes of data handled. Working for every numerical dataset	Working well in low data volume, no actions need. Working with a high volume of datasets we suggest train prediction algorithms, save the results, and search for the results when needed.