Project Title	High-performance data-centric stack for big data applications and operations
Project Acronym	BigDataStack
Grant Agreement No	779747
Instrument	Research and Innovation action
Call	Information and Communication Technologies Call (H2020-ICT-2016-2017)
Start Date of Project	01/01/2018
Duration of Project	36 months
Project Website	http://bigdatastack.eu/

D3.2 – WP 3 Scientific Report and Prototype Description – Y2

Work Package	WP3 – Data-driven Infrastructure Management
Lead Author (Org)	Orlando Avila-García (ATOS)
Contributing Author(s) (Org)	Ismael Cuadrado-Cordero, Bernat Quesada (ATOS), Jean Didier Totow, Christos Lyvas (UPRC), Sophia Karagiorgou (UBI), Nikos Drosos (SILO), Mauricio Fadel Argerich, Bin Cheng (NEC), Patricio Martinez Gracia, Jose Maria Zaragoza (LXS), Richard McCreadie, Zaiqiao Meng, Craig Macdonald (GLA), Luis Tomas Bolivar (RHT)
Internal Reviewer(s)	Yosef Moatti (IBM), Ricardo Jimenez-Peris (LXS), Dimosthenis Kyriazis (UPRC)
Due Date	30.11.2019
Date	29.11.2019
Version	1.0

The work described in this document has been conducted within the project BigDataStack. This project has received funding from the European Union's Horizon 2020 (H2020) research and innovation programme under the Grant Agreement no 779747. This document does not represent the opinion of the European Union, and the European Union is not responsible for any use that might be made of such content.

Dissemination Level

X	PU: Public (*on-line platform)
	PP: Restricted to other programme participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

Versioning and contribution history

Version	Date	Author	Notes
0.1	30.09.2019	Orlando Avila-García (ATOS)	Creation of the skeleton and first draft as a copy of D3.1. Sections 1, 2 and 4 updated for Y2. Updating description of scenarios by ATOS WDL.
0.2	07.10.2019	Orlando Avila-García (ATOS)	Changing the template structure for the description of high-level components.
0.3	18.11.2019	Orlando Avila-García (ATOS)	Adding updated component design, implementation and experimentations from GLA, ATOS, UPRC and UBI within sections 7, 8 and 9. Updates on Sections 2 and 4 by ATOS. Adding global experimentation outcomes from GLA within Section 10 and references.
0.4	19.11.2019	Orlando Avila-García (ATOS)	Adding updated component design, implementation and experimentations from RHT and NEC within sections 5 and 6.
0.5	25.11.2019	Orlando Avila-García (ATOS)	Adding comments and corrections from internal review by IBM and LXS. Adding Section 7.4.3 from ATOS.
0.6	28/11/2019	Orlando Avila-García (ATOS)	Adding amendments from GLA, ATOS, UPRC, NEC, UBI and RHT, addressing the internal review's corrections and amendments.
1.0	29/11/2019	Orlando Avila-García (ATOS)	Final format amendments and release of final version.

Table of Contents

List of tables

List of figures

Acronyms

ADS	Application and Data Services
ADW	Application Dimensioning Workbench
AWS	Amazon Web Services
CD	Continuous Delivery
CDP	Candidate Deployment Pattern
CEP	Complex Event Processing
CI	Continuous Integration
CNI	Container Network Interface
CRD	Kubernetes Custom Resource Definition
DDIM	Data-Driven Infrastructure Management
DNS	Domain Naming System
DO	Dynamic Orchestration
EKS	AWS Elastic Kubernetes Service
GCP	Google Cloud Platform
KPI	Key-Performance Indicators
K8S	Kubernetes
LbaaS	Load Balancer as a Service
OKD	Openshift Origin Kubernetes Distribution
OVN	Open Virtual Networking
OVS	Open Virtual Service
QoS	Quality of service
QoSE	Quality of service evaluation
RL	Reinforcement Learning
TME	Triple Monitoring Engine
SDN	Software-Defined Network
SLA	Service-Level Agreement
SLO	Service-Level Objective
NIC	Network Interface Controller
VM	Virtual Machine

1. Executive Summary

This is the Scientific Report and Prototype Description (Y2), reflecting the work done in the scope of the Data-Driven Infrastructure Management (DDIM) capability of the overall BigDataStack environment. The document describes the DDIM solution as assembled at M23 of the project (i.e. November 2019) in terms of updated design specifications, implementation, integration details, experimentation outcomes and next steps for the high-level components comprising the DDIM solution: Cluster Management, Dynamic Orchestration, ADS Ranking & Deploy, Triple Monitoring & QoS Evaluation, and Information-Driven Networking. Regarding research results, it focuses on the research conducted to optimize the two components bringing artificial intelligence (AI) to the solution: the ADS Ranking—responsible for ranking and selecting the best application deployment configurations—and the Dynamic Orchestration—in charge of making re-deployment decisions. Both components make use of machine learning (ML) techniques to bring the data-driven aspect to the DDIM capability. The rest of the components complement the above towards the overall data-driven functionality targeted at the level of infrastructure management: Cluster Management and Information-Driven Networking, responsible for the management and monitoring of the infrastructure resources (compute, storage and networking), and the Triple Monitoring & QoS Evaluation, responsible for the monitoring and evaluation of the QoS at different levels of the solution, that is, infrastructure resources, data services and application services. This document concludes with the description of global experimentation outcomes related to the ranking of the performance of grocery product recommendation techniques for the *Connected Consumer* use case.

2. Introduction

This deliverable presents the *Scientific Report and Prototype Description* of the work carried out during the second year (Y2) of the project, within WP3, to develop the Data-Driven Infrastructure Management (DDIM) capability of the BigDataStack environment. The document presents the details of the solution implemented during Y2 as well as the experimental results obtained from the research conducted to bring intelligence to the DDIM, that is, on the component ranking and enacting the best application deployment configurations (ADS Ranking and Deploy) and the component making re-deployment decisions (Dynamic Orchestrator). In particular, the specific machine learning (ML) algorithms used to bring data-driven decisions to the infrastructure self-adaption process are explained, as well as the research conducted to validate them. The rest of services playing a more conventional role in the infrastructure management are also presented: from the management of the infrastructure resources (Cluster Management and Information-Driven Networking) to the monitoring of those resources as well as the performance of the application and data services making use of them (Triple Monitoring and QoS Evaluation).

2.1. Relation to other deliverables

This document is related to the following past and immediately upcoming deliverables:

- D2.5 – Conceptual model and Reference architecture II (M18). The description of the high-level architecture of BigDataStack as well as the interplay and integration between the main components. The architecture of the Data-Driven Infrastructure Management as well as the design of the components have been devised to fit into the overall architecture.
- D2.3 – Requirements & State of the Art Analysis III (M22). The specification of BigDataStack requirements is centralized in this deliverable. The architecture of the Data-Driven Infrastructure Management (DDIM) as well as the design of the components have been devised to satisfy those requirements. *Please note that for the reader's convenience, the requirements related to each one of the DDIM components have also been included (literally brought from D2.3) in the present deliverable, specifically, at subsections 5.1, 6.1, 7, 8.1 and 9.1.*
- D3.1 – WP3 Scientific Report and Prototype Description - Y1 (M23). It described the solution as well as the experimental results produced in Y1. D3.2 presents the results obtained in Y2, which are necessarily an increment or refinement with respect to those presented in D3.1. Therefore, *please note those aspects of the solution that did not change during Y2 may appear in the same form in D3.2 and D3.1.*
- D4.2 – WP4 Scientific Report and Prototype Description – Y2 (M23). D3.2 makes references to some of the requirements and components which are designed, implemented and experimented with at WP4, while also the D4.2 references and raises requirements that are being described in the current document. In fact, the Data-Driven Infrastructure Management is meant to provide infrastructure services (Infrastructure-as-a-Service) to those components.
- D5.2 – WP5 Scientific Report and Prototype Description – Y2 (M23). D3.2 makes

references to some of the requirements and components which are designed, implemented and experimented with at WP5; this is because the tools developed at WP5 will interact with the services and resources provided by the infrastructure to implement certain functionality supporting the different BigDataStack stakeholders.

2.2. Relevant aspects unchanged from D3.1

As described in the previous section, this deliverable presents the Scientific Report and Prototype Description for Y2 for the work done in WP3. Therefore, much of the development and research work reported in this deliverable is a continuation or extension of the work reported in an equivalent report for Y1 (D3.1). However, in order to avoid the duplication of content, those aspects of the work which have remained unchanged for the last year are not reported again here but property referred to in D3.1. This is the case for:

i. The Solution Architecture (Section 3), including the architecture vision, assumptions, platform roles, example scenarios and the high-level design of the Data-Driven Infrastructure Management (DDIM) capability.

ii. Some of the experimental settings described in Section 4, the setting 1 and setting 2.

iii. The design specification of three out of the five building blocks of the architecture remained unchanged for the most part: Cluster Management (Section 5), ADS Ranking & Deploy (Section 7) and Information-Driven Networking (Section 9).

2.3. Document structure

The document is structured as follows: Section 3 describes the solution architecture of the Data-Driven Infrastructure Management (DDIM) capability of BigDataStack. Section 4 reports the Implementation and Experimentation: Starting with the experimental settings (Section 4.1), it describes the solution implementation roadmap giving support to the research (Section 4.2), and then finalizes with the description of experimental scenarios (Section 4.3).

The following five sections are dedicated to the requirements specification, design specifications, the presentation of experimental results, the description of interesting aspects of the implementation and integration of the component within the whole architecture, and some next steps: Cluster Management (Section 5), Dynamic Orchestration (Section 6), ADS Ranking & Deploy (Section 7), Triple Monitoring & QoS Evaluation (Section 8) and Information-Driven Networking (Section 9).

Finally, Section 10 presents global experimentation outcomes for the whole DDIM capability as developed at M23.

3. Solution Architecture

For a full description of the technical solution for the Data-driven Infrastructure Management (DDIM) capability, including architecture vision (context, goal, main functions or services), assumptions that the WP3 makes about the environment that BigDataStack will be deployed within, the platform roles engaged in the use of the capability, a full example scenario, and the global high-level design of the solution, please refer to D3.1 (WP3 Scientific Report and Prototype Description – Y1).

To better understand the structure and content of this deliverable, we bring from D3.1 the description of the five solution building blocks (components) the DDIM is made of:

1. Cluster Management (WP3-T3.1): Resource (compute and data) cluster services to BigDataStack, based on OpenShift container orchestration platform running on either OpenStack infrastructure-as-a-service (IaaS) or bare metal.
2. Dynamic Orchestration (WP3-T3.2): Runtime adaptation service in charge of resource re-allocation, storage and analytics re-distribution, re-compilation of network functions and re-deployment or applications and data services.
3. ADS Ranking & ADS Deployment (WP3-T3.3). Self-optimized deployment service for application components and data services, which are orchestrated following resource, application and data-aware deployment patterns.
4. Triple Monitoring and QoS Evaluation (WP3-T3.5). It consists of the resource clusters, data and application-level metrics collectors, the monitoring manager (which also gathers database related metrics) and the QoS evaluator, which evaluates Service-Level Objectives (SLOs) over those metrics.
5. Networking (WP3-T3.4). Data-driven networking services (self-)optimize the diverse networking needs among computing and storage resources as well as application components and data services (see Section 9).

4. Implementation and Experimentation

This section introduces the experimental scenarios and the methodological approach WP3 is taking to answer important questions and validate certain hypothesis to develop the Data-Driven Infrastructure Management (DDIM) capability. This section firstly presents four experimental settings; secondly, it describes the implementation roadmap to develop them; finally, it presents the experimental scenarios where they were enacted.

4.1. Experimental Settings

During Y2, we selected the *Connected Consumer* use case application to experiment with a series of DDIM capability prototypes (please refer to D3.1 for some use case highlights and D2.3 for a full description of it). In the following sections, we describe the experimental settings supporting such experiments, with an increasing level of complexity in terms of the number of BigDataStack components engaged.

4.1.1. Setting 1[1]: Recommendation Inference without Data Storage

An application engineer wants to deploy a recommendation model implemented by a data scientist. This recommendation system will provide product recommendations for customers that are visiting the company's e-commerce web site. Customer events in such a site will continuously feed the system to improve the recommendation model.

- The analytics application is made of two services (see Figure 1):
    - Normalization: which receives customer events and updates the Customer Preferences table with the customer activity. This table is then used as input in the Inference process.
    - Inference: takes the up-to-date Customer Preferences table and compute Product Recommendations table, which contains the list of products recommended per user.
- These application services contain state (i.e. Customer Preferences and Product Recommendations tables). Therefore, they cannot scale horizontally unless we provide a persistent storage. It is not integrated with a datastore, so we must flush the data to an already made, in-memory, distributed cache, so that the application services can become stateless and therefore horizontally scalable.

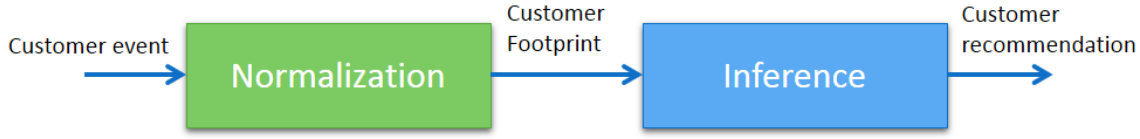


Figure 1 – Experimental scenario 1: Inference without data access (data flow view).

[1] Extracted unchanged from D3.1 (WP3 Scientific Report and Prototype Description – Y2). It is reported again in this document for the reader's convenience. See Section 2.3 for more details.

Requirements and constrains:

- A cache (in-memory) service was required to be deployed alongside the Normalization and Inference application services to store Customer Preferences and Product Recommendations tables and hence let them scale out (horizontally).
- The Inference is based on cross-selling by "collaborative filtering." The algorithm used is one of those already implemented in the NumPy library for Spark.
- Different experiments on the performance of the recommendation system have been accomplished, including the evaluation of latency for the Normalization service and throughput for the Inference service.
- Different experiments executing the Inference process in batches of different sizes.
- The Inference is being executed on a Spark engine, which is bundled and deployed together with the recommendation algorithm in a single container (stand-alone deployment). The single-node Spark configuration seek to serve as a first step to deploy Spark operations: In scenarios 2 and 3 the configuration becomes a more realistic multi-node cluster.

Customer events

The analytics application which is the subject of the scenarios is meant to provide service to EROSKI's e-commerce web site, specifically, product recommendations to customers. The analytics application service computes recommendations and the web application uses those recommendations to decide which products to show to the customer visiting the web; for each product it gives the option to view the detail of the product, add the product to the car, or discard that product so that it is not shown again as a recommendation to that customer.

Al these customer actions are captured as events and notified to the Normalization service which registers them in the Customer Preferences table, which in turn serves as input to the Inference service to update the Product Recommendations table. The definition of those events is the following:

- Recommendation shown (attributes: customer id, id recommendation, list of product ids). It will be used to discard a recommended product if the client has not shown interest in it (has not displayed it and has not added it to the car) after being shown as a recommendation a certain (configurable) number of times.
- Product added to the cart (attributes: id client, id recommendation, id product). We will give more weight to the recommendation of this product for this client.
- Product displayed (attributes: id client, id recommendation, id product). We will give more weight to the recommendation of this product for this client (but less than if you add it to the cart).
- Product discarded (attributes: id client, id recommendation, id product). Directly this product will be eliminated from the list of product recommendations for the given customers.

Deployment

Both services are expected to be containerized and deployed on Kubernetes as a single pod. This means that the scaling of the services will the carried out together, that is, increasing or decreasing the number of replicas at the pod level and not at the container level (i.e. scaling in and out).

The other action that can be carried out to dynamically adapt the deployment is to change the number of vCPUs per container (i.e. scaling up and down).

Quality of service

In different settings, the data scientist will need both processes to run with varying constraints of response time. Moreover, the throughout will be also an important consideration for the application engineer.

Other application-specific metrics (e.g., precision of the prediction, the success rate of the product recommendation) are not considered in this scenario.

4.1.2. Setting 2[2]: Recommendation Inference with Data Storage

Scenario 1 is enhanced by considering the persistence of both Customer Preferences and Product Recommendations tables in a data store, LeanXcale database.

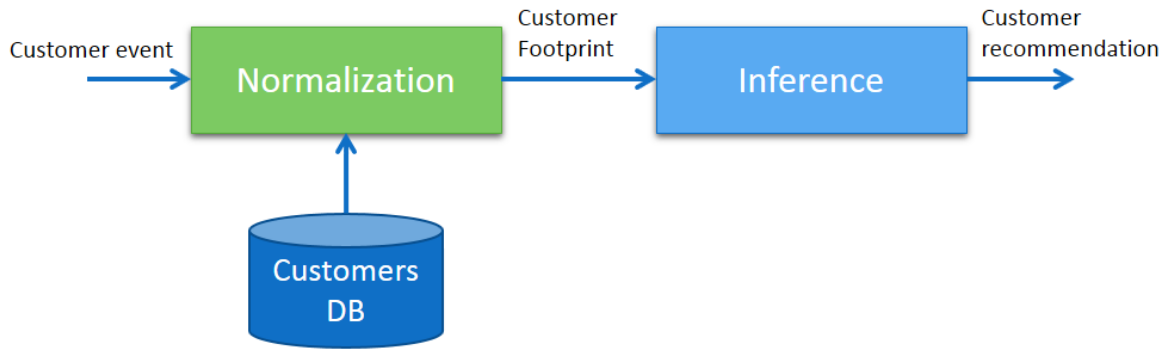


Figure 2 – Experimental scenario 2: Inference with data access (data flow view).

Requirements and constrains (refine scenario 1):

- A cache (in-memory) service is required to be deployed alongside the Normalization and Inference application services to store Customer Preferences and Product Recommendations tables and hence let them scale out (horizontally).
- The cache (in-memory) service permanently stores Customer Preferences and Product Recommendations tables in a LeanXcale database every time there is write operation.
- The Inference is based on “customer habits” by “individual behavioural analytics.”

[2] Extracted unchanged from D3.1 (WP3 Scientific Report and Prototype Description – Y2). It is reported again in this document for the reader’s convenience. See Section 2.3 for more details.

  - Instead of producing the whole recommendation table for all customers in every run, the inference process updates just the product recommendations for those customer/s whose events were received in a given time window.
  - The algorithm used will be one of those already implemented in the NumPy library for Spark.
- The Inference will be executed on a multi-node Spark cluster, so there is a need to come up with its optimal deployment (e.g., number of nodes, flavour of VMs, etc.).
- Different experiments executing the Inference as streaming analytics in micro-batches and real-time (i.e., with the arrival of every single event) will be accomplished.

Deployment

The application components are deployed in Kubernetes in the same way as in Scenario 1. For this scenario, the LeanXcale data base is expected to be deployed and operated as a WP4 component. This means that its deployment is not part of this scenario, which focuses on the integration between WP3 and WP4 regarding the storage layer and the impact on the analytics application layer.

Quality of service

Like in the previous scenario, different experimental settings with different QoS targeting low response time and high throughput will be run.

At least one application-specific metric (e.g., precision of the prediction or the success rate of the product recommendation) will be considered in this scenario.

4.1.3. Setting 3: Recommendation Process Modelling

Scenario 2 at M15 has been enhanced by considering the use of the application process modelling tool, a component from WP5. The Normalization + Inference process will be declaratively modelled in the tool and the automated deployment from that tool validated. Note this tool will be implemented over Node-RED at T5.2 (see D5.2 for more details).

4.1.4. Setting 4: Customer Event Stream Cleansing

Scenario 2 at M15 has been further enhanced by considering the integration with the data cleansing pre-processing service developed in WP4. In this experimental setting, such a service will be used to clean and enrich the streaming of customer behavioural events before being submitted to the Normalization service. Note this process will be implemented through the Real-time Complex Event Processing (CEP) resulting from T4.6 (see D4.2 for more details).

4.2. Implementation

Table 1 summarizes the experimentation (evaluation and validation) plan for the Data-driven Infrastructure Management capability between M15 and M24:

	M15	M18	M21	M24
Milestone	Prototype Validation	Implementation	Corrections and Design Evolution	Performance Optimization

Objective	WP3 starts deploying services in the (cloud native) WP3-provided Kubernetes-based computing infrastructure.	WP3 and WP5 components are integrated and use the same (cloud native) WP3-provided Kubernetes-based computing infrastructure.	WP3, WP4 and WP5 components are integrated and use the same (cloud native) WP3-provided Kubernetes-based computing infrastructure.	WP3, WP4 and WP5 components as well as their collaboration optimized to provide cost-effective orchestrated capabilities.
Success criteria	ALL WP3 services are deployed and running on Kubernetes to test the platform.	ALL WP3 and WP5 services are integrated and deployed on Kubernetes to evaluate the platform.	ALL WP3, WP4 and WP5 services are integrated and deployed on Kubernetes to evaluate the platform.	ALL WP3, WP4 and WP5 services are fully integrated and deployed on Kubernetes, providing a better platform performance than at M21.
	Experimentation with Setting 1	Experimentation with Setting 2 and 3	Experimentation with Setting 2 and 3	Experimentation with Setting 3 and 4

Table 1 - Data-driven Infrastructure Management capability experimentation phases.

Table 2 summarizes the Data-driven Infrastructure Management capability implementation roadmap for the past seven months:

	M15	M18	M21	M24
Experimental setting supported	1	1, 2	1, 2, 3	1, 2, 3, 4
Experimental scenario enacted	2	1, 2	1, 2	1, 2, 3
Cluster Management	OpenStack integration, Operators, Gateway	OpenStack integration, Cluster performance improvements Operators, Gateway	OpenStack integration, Cluster performance improvements, Operators, Gateway East/West Distributed Load Balancing	OpenStack integration, Cluster performance improvements, Operators, Gateway East/West Distributed Load Balancing
Dynamic Orchestrator	Agent Interpreter	Agent Interpreter	Agent Interpreter	Agent Interpreter

			ADS Interplay	ADS Interplay
Ranking & Deployment	ADS-Ranking, ADS-Deploy	ADS-Ranking, ADS-Deploy	ADS-Ranking, ADS-Deploy GDT	ADS-Ranking, ADS-Deploy GDT
Triple Monitoring & QoS Evaluation	SLALite, Prometheus, Graphana, Data metrics, Application metrics	SLALite, Prometheus, Graphana, Data metrics, Application metrics, Manager	SLALite, Prometheus, Graphana, Data metrics, Application metrics, Networking metrics, Manager	SLALite, Prometheus, Graphana, Data metrics, Application metrics, Networking metrics, Manager, Resource Cluster metrics
Information-driven Networking	Native	Native, Kubernetes, Networking & Policies Enforcement	Native, Kubernetes Networking & Policies Enforcement, Istio	Native, Kubernetes Networking & Policies Enforcement, Istio

Table 2 - Data-driven Infrastructure Management capability implementation plan.

4.3. Experimental Scenarios

This section explains the experimental use case scenarios, including success criteria which as used in the context of WP3 to verify & validate the behavioural invariances of the different components in order to ensure trustworthy run of component-specific experiments.

4.3.1. Scenario 1: Product recommendation service scalability

This scenario represents a situation where the application suffers a traffic spike that obligates the DDIM to scale out the application deployment so to keep its QoS, by increasing the number of one of the services' instances or replicas—e.g. scaling out.

ID	WP3-EXPSCE-01
Use Case	ATOS Worldline
Name	Scalability of the product recommendation service
Situation	Spike in the volume of traffic (requests per second - *rps*) to the online serving layer of the product recommendation system.
Settings	
Preconditions	What happened in the system before running the test? Initial conditions or state; e.g. **the product recommendation system is deployed.**

Trigger	What triggers this scenario, the entire use case, e.g. **the traffic or requests per second (*rps*) to the product recommendation service spikes.**
QoS requirements	Response time < 300ms
QoS preferences	Response time < 100ms
Postcondition	Expected result, e.g. the response time meets the QoS requirements.
Scenario	
Steps	1. We increase the rps to the product recommendation service from 0 to 1000. The response time remains under the SLO warning threshold. 2. We rise to 2000 rps. The response time goes beyond the SLO warning threshold but still below the SLO error threshold. 3. We rise to 3000 rps. The response time goes beyond the SLO error threshold. 4. The QoS Evaluator notifies a QoS violation to the DO. 5. The DO makes the decision to increase by one the number of replicas of the product recommendation service. It sends a request to the ADS-Ranking. 6. The ADS-Ranking produces the best re-redeployment to enact the DO decision and sends a request to the ADS-Deploy. 7. The ADS-Deploy executed the deployment specified by the ADS-Ranking by sending request to the cluster manager (Openshift). 8. Openshift increases by one the number of replicas of the product recommendation pod. 9. The response time of the product recommendation service drops below the SLO warning threshold.

4.3.2. Scenario 2: Product recommendation service steadiness

This scenario represents a stable situation, where the application can cope with the increase in traffic or requests per second (rps) as its keeps QoS as well as its deployment steady.

ID	WP3-EXPSCE-02
Use Case	ATOS Worldline
Name	Steady state of the product recommendation service
Situation	Volume of traffic (requests per second - *rps*) to the online serving layer of the product recommendation system remains constant.

Settings	
Preconditions	What happened in the system before running the test? Initial conditions or state; e.g. **the product recommendation system is deployed.**
Trigger	What triggers this scenario, the entire use case, e.g. **the traffic or requests per second (*rps*) to the product recommendation service increases, but the application can cope with it remaining in steady state.**
QoS requirements	Response time < 300ms
QoS preferences	Response time < 100ms
Postcondition	Expected result, e.g. the response time meets the QoS requirements.
Scenario	
Steps	1. We increase the rps to the product recommendation service from 0 to 1000. The response time remains under the SLO warning threshold. 2. We keep 1000 rps for 5 minutes. The response time remains under the SLO warning threshold, so no decisions to change the application deployment occur. 3. We rise to 2000 rps. The response time goes beyond the SLO warning threshold but still below the SLO error threshold, so no decisions to change the application deployment occur. 4. We drop to 1000 rps for 5 minutes. The response time remains under the SLO error threshold, so no decisions to change the application deployment occur.

4.3.3. Scenario 3: Product recommendation service cost-effectiveness

This scenario represents a situation where the application suffers a traffic spike that obligates the DDIM to scale out the application deployment so to keep its response time at certain SLO, like in Scenario 1, but adding a second SLO to ensure operational costs remain under certain threshold. Thus, this scenario exemplifies how the operational "cost" can be managed and enforced as just another SLO or QoS attribute by the DDIM.

ID	WP3-EXPSCE-03
Use Case	ATOS Worldline
Name	Cost-effectiveness of the product recommendation service

Situation	Spike in the volume of traffic (requests per second - *rps*) to the online serving layer of the product recommendation system.
Settings	
Preconditions	What happened in the system before running the test? Initial conditions or state; e.g. **the product recommendation system is deployed.**
Trigger	What triggers this scenario, the entire use case, e.g. **the traffic or requests per second (*rps*) to the product recommendation service spikes.**
QoS requirements	Response time < 300ms Compute resource cost < 2$ per hour
QoS preferences	Response time < 100ms Compute resource cost < 1$ per hour
Postcondition	Expected result, e.g. the response time as well as the compute resource cost (CRC) meets the QoS requirements.
Scenario	
Steps	1. The CRC is under 1$ per hour. 2. We increase the rps to the product recommendation service from 0 to 1000. The response time remains under the SLO warning threshold. 3. We rise to 2000 rps. The response time goes beyond the SLO warning threshold but still below the SLO error threshold. 4. We rise to 3000 rps. The response time goes beyond the SLO error threshold. 5. The QoS Evaluator notifies a QoS violation to the DO. 6. The DO makes the decision to increase by one the number of replicas of the product recommendation service. It sends a request to the ADS-Ranking. 7. The ADS-Ranking produces the best re-redeployment to enact the DO decision and sends a request to the ADS-Deploy. 8. The ADS-Deploy executed the deployment specified by the ADS-Ranking by sending request to the cluster manager (Openshift). 9. Openshift increases by one the number of replicas of the product recommendation pod. 10. The response time of the product recommendation service drops below the SLO warning threshold. 11. The CRC rises to 1.5$ per hour so beyond the SLO warning threshold but still below the SLO error threshold.

5. Cluster Management

The cluster management component's responsibilities are both to deploy the BigDataStack components as requested and to keep its status healthy overtime. This will not only include the containers but the related services and even the OpenShift Origen Kubernetes Distribution (OKD) cluster itself. In addition, it is in charge to adapt the current deployments to the new preferred status requested by the upper layers, in order for example to increase the size of the cluster, or scale up/down a given application.

5.1. Requirements

To facilitate the understanding of the design as well as the challenges addressed by this component, the requirements related to this component have been brought from D2.3 and literally included into this section. Please note the following requirement tables are compiled together with the rest of requirements of BigDataStack in D2.3, and that they are included in here for the reader's convenience.

	Id	Level of detail	Type	Actor	Priority
	REQ-CM-01	Software	FUNC	Application Engineer, Data Engineer	MAN
Name	Support OpenShift installation on OpenStack VMs				
Description	Include the needed steps on the OpenShift installer to handle OpenShift cluster installation on top of OpenStack resources, i.e., VMs, networks, volumes, etc.				
Additional Information	This needs to be done in the 'upstream' way so that it is supported also after the project lifecycle. It entails modification to different repositories, not only the Openshift/installer[3] but also other related repositories such as: - cluster-network-operator[4] - cluster-api-provider-openstack[5] - gophercloud[6]				

Table 3 - Requirement (1) for Cluster Management

	Id	Level of detail	Type	Actor	Priority
	REQ-CM-02	Software	PERF	Application Engineer, Data Engineer	MAN
Name	Avoid double encapsulation of network packages				

[3] https://github.com/openshift/installer
[4] https://github.com/openshift/cluster-network-operator
[5] https://github.com/kubernetes-sigs/cluster-api-provider-openstack
[6] https://github.com/gophercloud/gophercloud

Description	Integrate Kuryr into the OpenShift installer to avoid the double encapsulation problem due to using 2 different overlays (OpenStack SDN and OpenShift SDN on top). Kuryr enables containers running on top of OpenStack VMs to use the same SDN as the VMs itself, i.e., the OpenStack SDN. Thus, avoiding the double encapsulation and enabling a remarkable throughput gain.
Additional Information	Similarly, to REQ-CM-01, this needs to be done in the 'upstream' way so that it is supported after the project. It entails modifications to the same repositories plus the addition of a kuryr operator that will handle the kuryr related operational actions,

Table 4 - Requirement (2) for Cluster Management

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-CM-03	Software	FUNC	Application Engineer, Data Engineer	MAN
Name	Kubernetes Network Policy support at Kuryr-Kubernetes				
Description	As we are integrating kuryr to get network performance optimizations when running OpenShift on top of OpenStack, we need to include the mechanisms needed for kuryr to be able to enforce Kubernetes network policies, i.e., to define in a fine grain manner how pods can communicate with each other				
Additional Information	Similarly, to REQ-CM-01, this needs to be done in the 'upstream' way so that it is supported after the project. It entails modifications to the Kuryr-Kubernetes repositories.				

Table 5 - Requirement (3) for Cluster Management

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-CM-04	System	PERF	Application Engineer, Data Engineer	DES
Name	OVN-base distributed load balancer for Kubernetes services				
Description	Kubernetes services are implemented through Octavia when using Kuryr. This means that for each Kubernetes service an Octavia amphora VM is created. This adds extra latency on the communication, is a single point of failure, adds extra resources need, and it adds delays on the control plane actions. By integrating the OVN distributed load balancer (as a new ovn-Octavia driver) and making Kuryr use it, we avoid all those problems by implementing the load balancing directly with ovn flows. This remove the need for VM resources and speed up both control and data planes.				
Additional Information	Similarly, to REQ-CM-01, this needs to be done in an 'upstream' way so that it is supported after the project. It entails modifications and integration in several upstream projects: - OVN				

	- OpenStack Octavia - OpenStack networking-ovn - Kuryr-Kubernetes - OpenShift Cluster Network Operator

Table 6 - Requirement (4) for Cluster Management

	Id	Level of detail	Type	Actor	Priority
	REQ-CM-05	System	FUNC	Application Engineer, Data Engineer	MAN
Name	API managed OpenShift cluster (CAPO)				
Description	The OpenShift cluster installed on top of OpenStack consists of X VMs on OpenStack. We need to extend the Cluster API Provider OpenStack in order to allow the modification of the cluster size through Kubernetes API calls. This allows flexibility on the OpenShift cluster to adapt to the current needs				
Additional Information	Similarly, to REQ-CM-01, this needs to be done in the 'upstream' way so that it is supported after the project. It entails modifications to the next upstream projects: - openshift/cluster-api - openshift/cluster-api-provider-openstack - openshift/installer				

Table 7 - Requirement (5) for Cluster Management

	Id	Level of detail	Type	Actor	Priority
	REQ-CM-06	System	ENV	Data Engineer	DES
Name	Spark operator				
Description	This operator will be responsible for handling the Spark cluster, not only its installation but also the scaling actions. It will offer an API to the Spark management through the OpenShift API.				
Additional Information	This is related to the dynamic orchestrator, as the optimization actions could be then simply triggered through standard OpenShift API commands (e.g., modifying the information at the associated spark ConfigMap)				

Table 8 - Requirement (6) for Cluster Management

	Id	Level of detail	Type	Actor	Priority
	REQ-CM-07	System	ENV	Adaptable Distributed Storage	DES
Name	Accept requests to allocate additional resources to storage components				
Description	The Adaptable Distributed Storage component can be scaled in/out				

	independently, considering decisions based on its internal metrics and handle on its own the reconfiguration of the internal data regions. Due to this, it is necessary from the Cluster Management to provide a mechanism that allows the storage layer to request for additional resources or the release of already provided ones.
Additional Information	This is closely related to requirement REQ-ADS-04 "Be able to request additional resources from the infrastructure layer," described in D4.1.

Table 9 - Requirement (7) for Cluster Management

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-CM-08	System	ENV	Adaptable Distributed Storage	OPT
Name	Force the storage layer to release some of its available resources				
Description	The cluster management might identify that the overall BigDataStack platform is running out of available resources. To ensure the execution of crucial components, it might decide to reduce resources for some services, to the benefit of others. Due to this, it should be able to request the release of the storage resources and wait for its proper response. The storage should be able to reject such requests, in cases that could lead to data loss.				
Additional Information	This is close related with requirement REQ-ADS-05 "Being able to release resources and adapt if resources are deallocated from the infrastructure," as described in more details in D4.1.				

Table 10 - Requirement (8) for Cluster Management

5.2. Design Specifications

The design for this component (specified in Section 5 of D3.1) has mostly remained valid for Y2. The following sections describe the aspects of the design that have been changed.

5.2.1. Cluster performance improvements

Due to the BigDataStack requirements, not only related to fast data processing but also speeding up communications between the different components running on top of OpenShift, there is a need for performance improvements into the network data plane. Simply installing OpenShift/Kubernetes on top of OpenStack VMs means that, on the one hand you have the OpenStack network overlay (to manage the traffic between the VMs), and on the other hand the OpenShift SDN (e.g., *openshift-sdn*). This leads to the so-called "double encapsulation problem" which impose severe performance degradation on the network throughput (besides the added complexity on network management and debugging upon failures). To avoid this problem Red Hat has been working on an OpenStack project named ***Kuryr***[7] that enables the usage of OpenStack Software-Defined Networks (SDNs) in the OpenShift cluster running on top of the OpenStack VMs. This allows to have a

[7] https://docs.openstack.org/kuryr-kubernetes/latest/

single SDN for both systems (OpenStack and OpenShift), as well as to avoid the double encapsulation problem when having one SDN (openshift-sdn) on top of the other (Neutron SDN) as there is no VXLAN over VXLAN. In addition, we are working at integrating Kuryr on the OpenShift installer as well as creating an operator for its management (see Figure 3). After discussing with the community, it was agreed that Kuryr components should be part of the Cluster Network Operator and therefore are installed as part of its process.

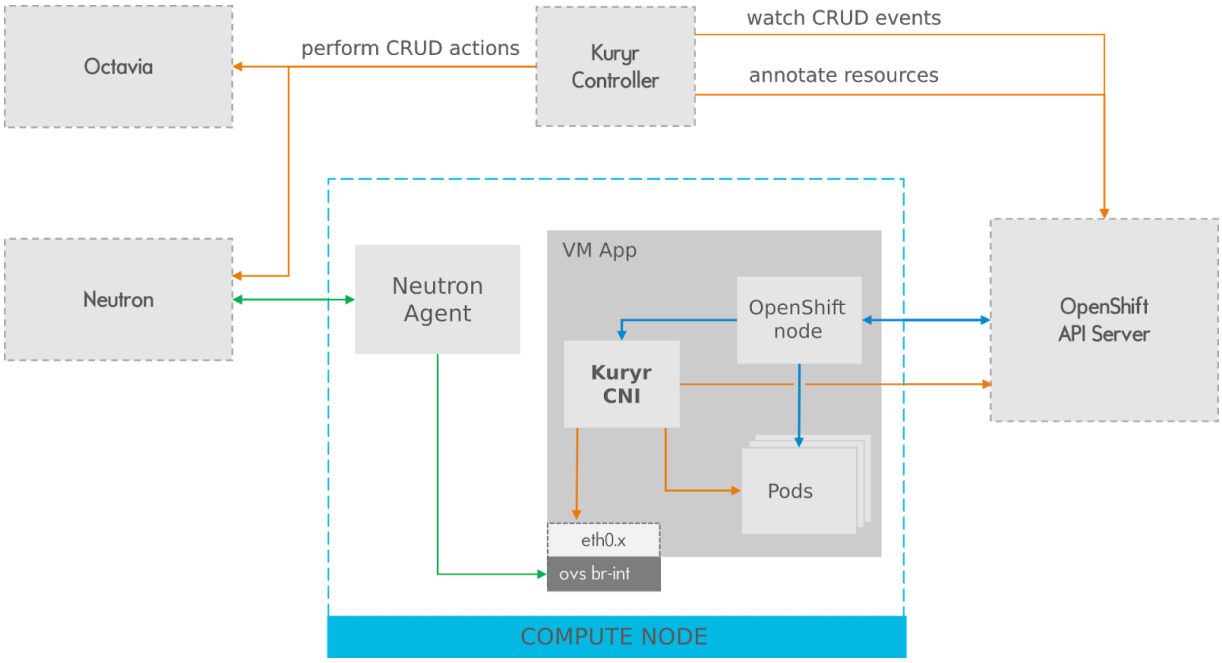


Figure 3 – Red Hat Kuryr's architecture to avoid the "double encapsulation problem."[7]

This however also imposes certain requirement on the OpenStack side. The next components need to be installed or have specific configuration:

- Octavia (Load Balancer as a Service) component need to be installed, and with it, its dependencies such as Barbican in case of using TLS termination
- Neutron needs to be configured with Trunk ports support. Depending on the used ml2 driver, the configuration can be slightly different. For instance, it is out of the box if OVN is being used, but if ML2/OVS is being used, it needs to be enabled, and the openvswitch driver needs to be set to enforce security group policies on the containers.
- Depending on the installed OpenShift version (3.11 or 4.X), Heat is also needed to create a stack containing all the OpenShift related resources, i.e., VMs, Volumes, Networks, LbaaS, … This is needed for the 3.11 OpenShift installer
- Depending on the installed OpenShift version (3.11 or 4.x), Swift is also needed to store the initial ignition files used to configure the OpenStack VMs. This is needed for the 4.X OpenShift installer.
- The user quota needs to be adapted to the container deployments scale, i.e., it will not be enough with just a few neutron ports as each container will be using one. Consequently, some of the resource's quota need to be increased by an order of

magnitude (depending on the size of the OpenShift deployment), specially quotas related to Neutron resources, such as ports, networks/subnets and security groups.

5.2.2. Gateway

The gateway for the BigDataStack engine can also be implemented as part of OpenShift, in 2 different ways depending on the final requirements:

- By using OpenShift routes: Route is a way to expose OpenShift services by giving it an externally reachable hostname, like www.example.com. It has the option to perform the routing based on paths, i.e., we can use it to redirect some queries to the CEP component (i.e., www.example.com/cep/…) and others to the Alarm component (i.e., www.example.com/alarms/...). The initial design targets to use this, being able to assign a common OpenStack Floating IP for all the ingress traffic to OpenShift Apps, in this case BigDataStack components.
- By using Istio service mesh: A service mesh is a network of microservices that enables applications and the interactions among them. It offers functionality like load-balancing, fine grain traffic control, access control, logging, tracing, etc., through *sidecards* containers associated to the applications pods. One offered functionality is Istio-Gateways which controls the exposure of services at the edge of the mesh. This could be used to tie gateways to specific virtual services that can perform the extra required actions that the gateway may require besides redirecting the traffic to the desired endpoint.

5.2.3. East/West Distributed Load Balancing

In Kubernetes and OpenShift, the communication between the different application components and between applications (i.e., between the Pods) is not meant to be *pod* to *pod* (and using IPs) since pods are supposed to be disposable and therefore they can be replaced/deleted at any time. Pods are usually behind a service which abstracts the IP/name of the container(s) that is pointing to. This way, pods can talk to known services IPs (and names) and containers after that service can be recreated at any time without impacting the way the caller pods uses to reach them.

Given the above, the pod to svc to pod communication performance is quite important as it is the most usual pattern. When using Kuryr, Services are implemented as Octavia load balancers. This means that each K8s service will require Octavia load balancer, and with the default ‘amphora’ driver that means an OpenStack VM. This has 4 main implications:

1. Resource waste since lots of VMs will be needed for backing the services.
2. User experience as services will need more time to be up and running since the amphora VM must be created and configure.
3. Single point of failure for services as if the VM dies, a new one will need to be created to replace it.
4. Network latency as traffic needs to do extra hops to reach the amphora VM.

For these reasons we plan to work on the integration of OVN load balancer into OpenStack, including Octavia and Kuryr. The OVN load balancer is a distributed load balancer based on OVS/OVN flows. This means that it does not require any amphora VM to load balance the traffic and simply creates the needed flows locally on each OpenStack compute node. To make it easier to understand, it is like if an *iptable* rule was changing the Kubernetes service

IP by one of the Kubernetes endpoints (pods) IPs and then the traffic was directly forwarded to the selected pod.

By working on this integration, the next advantages are obtained:

- Time to create a K8s service will be of a few seconds instead of around 1 minute
- No extra resource waste for services, just a few OVS flows
- No single point of failure
- Distributed routing as the traffic goes directly pod to pod instead of having to jump to the OpenStack node that has the amphora VM and back
- Reduced latency, increased throughput
- No need to parse Security Groups at amphora load balancer to apply Kubernetes Network Policies, with the consequent reduction on Neutron OpenStack load.

5.3. Implementation and Integration Highlights

Initial support for OpenStack has been included into the OpenShift installer to handle the creation of OpenStack resources. Currently the prototype is based on OpenShift 3.11 as OpenShift 4.X was being developed and an OpenShift cluster was needed so as not to block the other components. As soon as we have OpenShift 4 working on OpenStack we will move to the installation type—that is based on operators.

This support extends the OpenShift installer to create OpenStack VMs and later install the packages, configuration files, keys, services, etc., needed to install and configure the OpenShift cluster on top of them. It includes the basic operators and prepares the system for the new ones to be created as part of the BigDataStack project.

We followed the best practices (configuration) for deploying OpenShift on top of OpenStack already outlined within D3.1; also refer to that deliverable to see an account of the minimum number of each OpenStack resource types that are needed for a minimal installation of OpenShift on top of OpenStack.

5.4. Experimentation Outcomes

In this second year, the experimentation has focused on Initial integration testing and scale testing of OVN-Octavia distributed load balancing for Kubernetes Services.

A performance comparison between Kuryr and OpenShift SDN was carried out, proving a performance boost of up to nine times better for throughput, as presented in the following figures, while additional results have been published online at the OpenShift blog[8].

[8] https://blog.openshift.com/accelerate-your-openshift-network-performance-on-openstack-with-kuryr

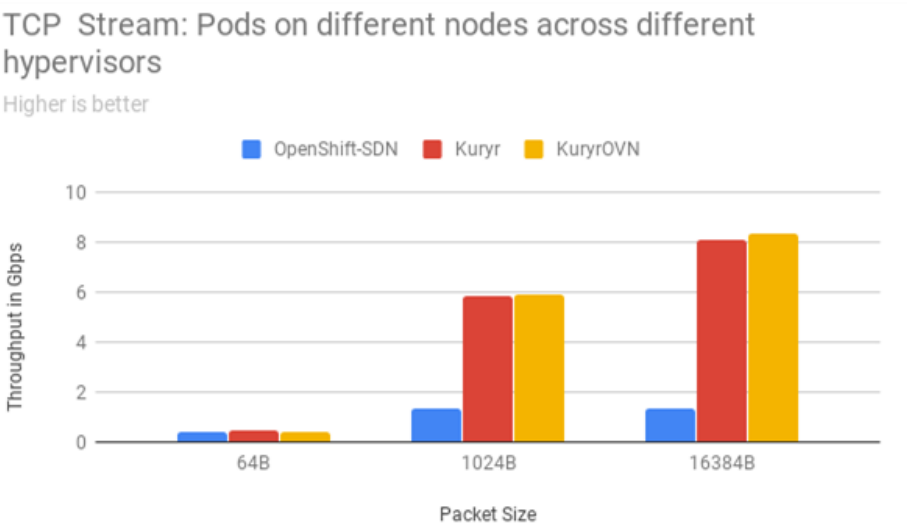


Figure 4 – Throughput improvements (POD to POD)

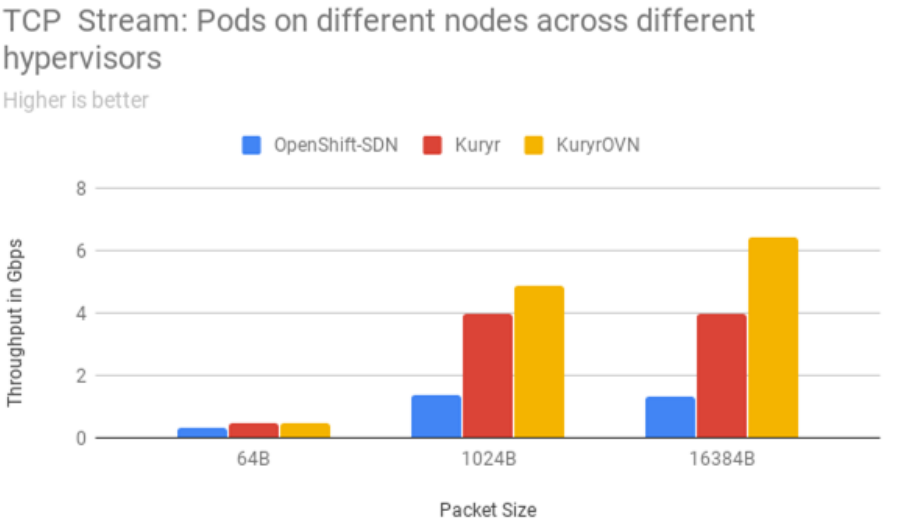


Figure 5 – Throughput improvements (POD to SVC)

5.5. Next Steps

The plan is to continue with the improvements for running OpenShift on top of OpenStack, both in terms of performance and operability. The first focus will be on distributed OVN load balancer integration. And then we will also explore OpenShift autoscaling mechanisms based on the API extensions made during the first half of the project.

Another focus will be on more performance storage and network integration solution. For the storage the focus will be at enabling the usage of high-performance storage into the OpenShift VMs (work already started). For the network, the path is to explore how to enable secondary NICs in the pods, when Kuryr CNI is used. This should be linked to the usage of SRIOV NICs on the OpenShift VMs and their exposure to the OpenShift nodes through *multus* plugin[9].

[9] https://kubernetes.io/docs/concepts/cluster-administration/networking/#multus-a-multi-network-plugin

6. Dynamic Orchestration

The Dynamic Orchestrator will provide more flexibility and enhanced performance for applications that utilize the BigDataStack. The application's performance and compliance with its requirements will be monitored during runtime and when a requirement violation is detected, the Dynamic Orchestrator will change the application's deployment in order to comply with all requirements.

6.1. Requirements

To facilitate the understanding of the design as well as the challenges addressed by this component, the requirements related to this component have been brought from D2.3 and included into this section. Please note the following requirement tables are compiled together with the rest of requirements of BigDataStack in D2.3, and that they are included in here for the reader's convenience.

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-01	System	FUNC	Application Engineer, Data Engineer	MAN
Name	Playbook Enrichment				
Description	The Dynamic Orchestrator shall ingest the Playbook when an application or service is deployed and enrich this playbook with information about the QoS metrics and intervals to be considered by the Triple Monitoring to monitor the QoS during runtime.				
Additional Information	N/A				

Table 11 - Requirement (1) for Dynamic Orchestrator

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-02	Stakeholder	FUNC	Application Engineer, Data Engineer	MAN
Name	Runtime Re-deployment				
Description	When an application or service is running, the Dynamic Orchestrator shall determine if a deployment change should be performed when there is a violation of an application requirement or Service Level Objective (SLO) and send a signal to the ADS-ranker to trigger a change in the deployment to try to satisfy the requirements or SLOs.				
Additional Information	The Triple Monitoring detects this violation and sends an alert to the Dynamic Orchestrator to start this process.				

Table 12 - Requirement (2) for Dynamic Orchestrator

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-03	Stakeholder	PERF	Application Engineer, Data Engineer	MAN
Name	Decision Efficiency				
Description	The orchestrator shall consider different re-deployment mechanisms such as scaling processing nodes, changing configuration of VMs – e.g. vRAM, vCPU -, data placement and more and decide what mechanism has the highest probability of improving the requirements or SLOs satisfaction according to the system and application status.				
Additional Information	The complete list of mechanisms is still being under consideration.				

Table 13 - Requirement (3) for Dynamic Orchestrator

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-04	System	FUNC	Application Engineer, Data Engineer	MAN
Name	Resources Limits				
Description	The orchestrator shall be able to receive a trigger from the ADS-Ranker when a deployment parameter, such as the number of replicas, the number of vCPUs or the assigned cluster memory, cannot be further increased or decreased (i.e. this resource has reached its maximum or minimum possible value) and use this information in its own decisions.				
Additional Information	The complete list of deployment parameters might vary according to the application/service and its actual deployment. This information should be available in the Playbook or other resource.				

Table 14 - Requirement (4) for Dynamic Orchestrator

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-05	Stakeholder	FUNC	Application Engineer, Data Engineer	DES
Name	Orchestration for Improvements				
Description	When an application or service is running, the orchestrator shall detect changes in the system status or inputs (e.g. less new events per minute) and trigger a change in the deployment that results in lower costs (e.g. to use less replicas) without compromising the application functioning.				
Additional Information	N/A				

Table 15 - Requirement (5) for Dynamic Orchestrator

	Id	Level of detail	Type	Actor	Priority
	REQ-DO-06	System	FUNC	Application Engineer	MAN
Name	Management of Multiple Objectives				
Description	Because of the different SLOs/requirements applications can have, the DO shall consider different objectives such as optimizing application performance, use of resources and reliability. It might be desired that multiple objectives are optimized at the same time and some of them might also be opposed to each other, e.g. application performance might be maximized by using more resources which affects the resources optimization				
Additional Information	N/A				

Table 16 - Requirement (6) for Dynamic Orchestrator

6.2. Design Specifications

To comply with the requirements stated in section 6.1., we propose to implement the DO's logic by developing a Reinforcement Learning (RL) approach. We have chosen RL because it offers a formal framework in which we can formulate the dynamic orchestration problem and responds to the above requirements:

1. It offers dynamic and adaptable decisions during runtime, learning from its own experience and tailoring its decisions according to the environment (in our case the BigDataStack platform and the application managed) (REQ-DO-02)
2. It can consider multiple actions (in our case re-deployment mechanisms) and learn what action should be taken according to the state of the environment. Furthermore, it learns the characteristics of each different application by experience, to identify the effective actions for each different application (REQ-DO-03 and REQ-DO-04)
3. It can manage and optimize multiple objectives through a proper design of its reward function [38] (REQ-DO-05 and REQ-DO-06)

In the general setting of RL, an agent learns how to control an unknown environment by interacting with it, in order to achieve a certain goal. To control the environment, the agent can perform a set of actions that may alter the state of this environment. For each action performed, the agent observes the change in the environment's state and a numerical signal, usually called reward, that indicates if the action performed moved the agent closer or further to the completion of its goal.

In the RL framework, the agent is the system's manager, and the system and its execution context are seen as the environment; the agent needs to find the best configuration by modifying the configuration parameters, seen as actions (see Figure 6). Existing studies [39, 40, 41, 44] show that RL can lead to a good performance for the configuration of different systems, after learning from many iterations doing the same task. However, the need of extensive experience is a problem, because the RL agent should already start with a reasonable performance. In fact, every action the agent takes in this case has a high cost of

redeploying the application or service, affecting the current service as well as the computing resources in the platform.

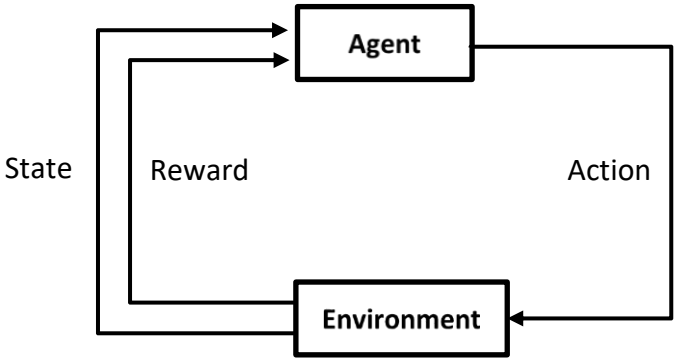


Figure 6 – Reinforcement learning feedback loop with the environment.

Because of this, we are developing a new approach called ***Tutor4RL***. Tutor4RL takes as input domain knowledge guidelines that are used to constraint, explore and learn from the environment in which the agent is deployed, while learning from its own experience the best actions to achieve its goal in different states.

We modify the RL framework by adding a component we call the Tutor. The tutor possesses external knowledge and helps the agent to improve its decisions, especially in the initial phase of learning when the agent is inexperienced. In each step, the tutor takes as input the state of the environment and outputs the action to take, in a similar way to the agent's policy. However, the tutor is implemented as a series of programmable functions that can be defined by domain experts and interacts with the agent during the training phase. We call these functions knowledge functions and they can be of two types:

- Constrain functions: are programmable functions that constrain the selection of actions in a given state, "disabling" certain options that must not be taken by the agent. For example, if the developer of the application has decided a maximum budget for the application, even the application load is high and this could be fixed by adding more resources to the deployment, this should not be done if the budget of the user has already reached its maximum.
- Guide functions: are programmable functions that express domain heuristics that the agent will use to guide its decisions, especially in moments of high uncertainty, e.g. start of the learning process or when an unseen state is given. Each guide function takes the current RL state and reward as the inputs and then outputs a vector to represent the weight of each preferred action according to the encoded domain knowledge. For example, a developer could create a guide function that detects the number of current users for an application and if the number is higher than a certain threshold, more resources might be deployed for the application.

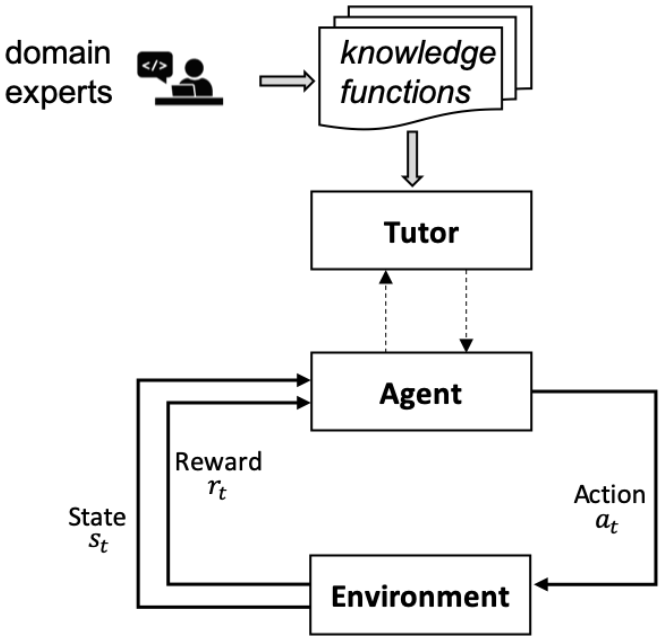


Figure 7 – High level vision of Tutor4RL.

The benefit coming from using Tutor4RL is twofold:

- During training, the tutor enables a reasonable performance, opposed of the unreliable performance from an inexperienced agent, while generating experience for the agent's training. Furthermore, the experience generated by the tutor is important because it provides examples of good behaviour, as it already uses domain knowledge for its decisions.
- The knowledge of the tutor does not need to be perfect or extensive. The tutor might have partial knowledge about the environment, i.e. know what should be done in certain cases only; or might not have a perfectly accurate knowledge about what actions should be taken for a given state. Instead, the tutor provides some "rules of thumb" the agent can follow during training, and based on experience, the agent can improve upon the decisions of the tutor, achieving a higher reward than it.

The main functioning of Tutor4RL is as follows:

1. Application developer (i.e., the domain expert) defines guide and constrain functions These functions encode domain knowledge of the developer that guide and constrain the RF agent during its initial stage. This is important for new applications and/or a new system execution context, where traditional RL would need to explore the state space randomly and thereby negatively impact QoS of the application. If the application has been deployed before, Tutor4RL can use the historical data from that previous deployment and encodes it as a guide function.
2. The Triple Monitoring Engine and QoS Evaluation informs the Interpreter about the current system metrics and the SLO violations, respectively.
3. These metrics are taken as input by the agent and the tutor and both output a vector with valuations for each action.

The RL Agent selects an action, from its policy or from the suggestions provided by the tutor, that should be executed by the ADS-Ranker and sent to it.

6.2.1. Adaptable Distributed Storage Interplay

The Adaptable Distributed Storage (as described in 4.2) will not rely on the Dynamic Orchestrator or the Ranking & Deployment to scale in/out its resources; rather, because of the larger number of metrics available internally, it integrates its own Elasticity Manager subcomponent that is responsible for taking this kind of decisions for the storage layer. As a result, the storage can be re-configured automatically, moving data regions across its current nodes and scale in or out to be adapted under diverse workloads. As these redeployments are being triggered separately, the Dynamic Orchestrator should be aware of those, and postpone any redeployment action on the application level until the reconfiguration of the storage is finished, and the system is balanced.

Therefore, the Dynamic Orchestrator needs to consider there is a second dynamic adaptation mechanism acting at the storage layer level. This second adaptation component (i.e., Elasticity Manager) will inform the Dynamic Orchestrator component regarding reconfigurations of the data storage layer; in fact, this has been specified as a requirement imposed on the Adaptable Distributed Storage (see REQ-ADS-06) by the Dynamic Orchestrator. More specifically, the Adaptable Distributed Storage will notify information regarding pending redeployments of the storage, when the process of data reconfiguration starts and finishes, along with the current deployment of this layer.

In our setting, the Adaptable Distributed Storage logic is seen as a Guide function, so it is used by the agent to improve its performance. This information helps the DO to determine in what cases the Adaptable Distributed Storage should be scaled up or down, first by observing the behaviour of the already implementing logic, and then repeating and potentially, improving these decisions thanks to having a broader picture of the application and system status.

6.3. Implementation and Integration Highlights

We have completed an initial design and implementation of the DO, which has been completed to provide the following overall functionality:

1. The Triple Monitoring Engine (TME) & QoS Evaluation (QoSE) informs the Interpreter about the current system metrics and the SLO violations, respectively.
2. The Interpreter converts these metrics and violations in states and rewards:
    a. The states represent the system status in a discrete space.
    b. The rewards indicate the Reinforcement Learning Agent if an executed action was “good” or “bad” in terms of requirements and SLOs compliance (e.g. if the requirements and SLO violations disappeared after the execution of an action).
3. The Interpreter sends the current state of the system to the RL (Reinforcement Learning) Agent and according to this, the RL Agent selects an action, from its policy or from the suggestions provided by the tutor, that should be executed by the ADS-Ranking:
    a. The actions are type of changes in the deployment such as change the number of replicas, change the number of vCPUs or change the vRAM assigned—note these are just some of the changes that are being considered, the full list of deployment changes still needs to be determined.

b. One of the actions is to keep the current deployment.

4. Once an action has been executed, the interpreter receives the new metrics and SLO violations, calculates the reward and sends it to the RL Agent.
5. The RL Agent updates its state-action ranking (Q-values).

Note that so far, we have not implemented Tutor4RL in this early version because the development of Tutor4RL has come as a result of our experimentation with this early version of the DO. The implementation of Tutor4RL as part of the DO will be addressed in our next step.

6.4. Experimentation Outcomes

As described above, we have implemented an early version of the DO using Tabular Q-learning and tested it in simulations of a streaming application in which the load of the application increases (see [44]) for a detailed description and evaluation of this prototype). This streaming application can find lost children based on the processing of camera data. It can be split in two components: (1) an offline module, which is trained with pictures of the child in a server and (2) an online module, a face detection and matching service that is deployed in several devices and is in charge of finding the child (see **Error! Reference source not found.**).

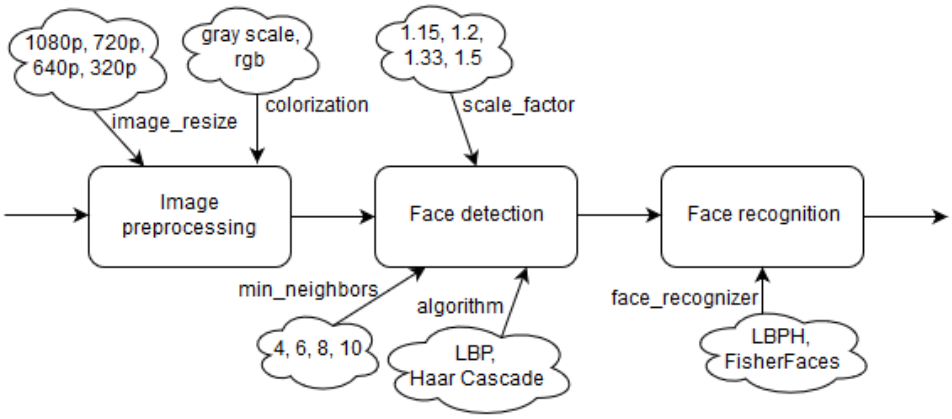


Figure 8 – Example of streaming analytics application.

We have shown that RL can be used efficiently (up to 25% better precision than a state-of-the-art heuristics) to dynamically orchestrate such a data processing pipeline like the ones in BigDataStack. However, we noticed two issues with applying traditional RL:

i. Bad performance during the "training phase" of the RF agent, and
ii. Missing constrains to avoid clearly wrong actions.

Both issues are very relevant to BigDataStack: BigDataStack applications need to be ready from the start and the DO should ideally avoid completely wrong actions. We started to address both issues with Tutor4RL.

Tutor4RL adds two features to traditional RL: Guide Functions and Constrain Functions. These functions enable the user to give some initial knowledge to the RL agent to direct its initial exploration.

We implemented a prototype of Tutor4RL with standard RL libraries in order to provide a fair comparison of it against other heavily used RL algorithms. Specifically, we have modified the library Keras-RL to implement a tutored Deep Q-Network (DQN) agent.

An important question in our model is when the tutor should decide for the agent and vice-versa. In a similar way to how Epsilon greedy exploration works, we defined *Tau* as the threshold parameter for the agent to control when it will use the suggested actions from the tutor instead of using its own. The initial value of *Tau* is a parameter of our model and the best value to initialize it depends on the use in which Tutor4RL is used. This parameter is linearly reduced while the agent gathers more experience and learns to take better decisions.

To test Tutor4RL, we have used the library OpenAI *gym* [42], which provides several environments ready to be used with RL. As we are testing a DQN agent, we decided to use the Atari game Breakout [43] which is a complex use case in which we can observe how the agent performs in cases in which reward is sparse and episodes are long in time steps. This is a different use case than the one we are addressing in BigDataStack, but we have chosen it because it is heavily used in the RL literature, so it lets us compare Tutor4RL with the state-of-the-art in a straightforward manner.

In Breakout, the state of the environment in each time step is the video games' frame in pixels. The actions are four: no operation, fire (which throws the ball to start the game), left and right. The reward is the points achieved in the game, given each time a brick is broken.

We implemented a simple guide function that encapsulates some basic knowledge about the game: the function takes as input each frame, searches for the ball and the position of the bar, and moves the bar to the left if the ball is to the left of the bar or to the right if the ball is on that side. If the ball is not seen, then the action chosen is "fire" to start the game.

We have compared the functioning of Tutor4RL by also training a plain DQN agent for the same use case. The results can be seen in the plot below:

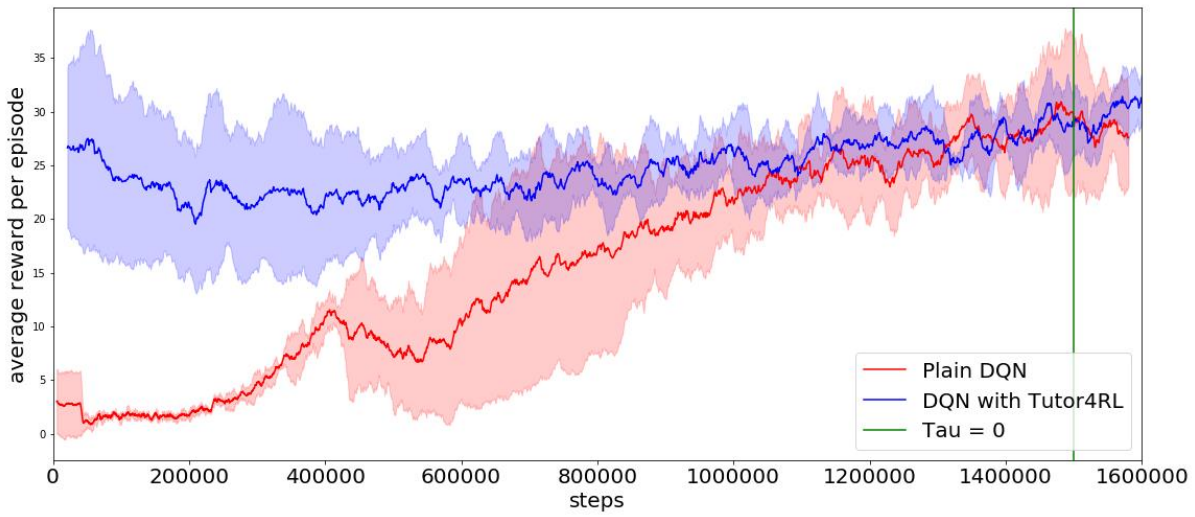


Figure 9 – Comparison of performance between Tutor4RL and a plain DQN agent.

As it is possible to see, from the initial steps the DQN agent with Tutor4RL manages to achieve a reasonably high reward while the plain DQN agent performs very poorly, because

of its inexperience. As the agents perform more steps, the plain DQN agent catches up, but it's not until step 1 million that it manages to achieve a similar reward to the tutored DQN agent. *Tau* is decreased in every step, starting with a value of 1 and reaching 0 in step 1.5 million. It is important to note that after this step, the tutor is not used anymore but the agent keeps up with its high reward.

6.5. Next Steps

In the future, we plan to improve our early version of the DO by using deep RL and including Tutor4RL. We also plan on researching what guide functions can be derived from existing techniques of system's management/configuration, as well as what actions should be disable in certain cases and expressing it as constrain functions.

We are also going to research ways to efficiently combine the policy's output with the guide and the constrain functions' outputs. In addition, we plan to implement all the possible re-deployment actions that the DO will be able to provide during runtime.

As for tests, we plan on carrying out several tests to determine:

i. How reliable is our approach of Tutor4RL for configuration of systems?

ii. What is the performance that the DO can achieve after acquiring experience with BigDataStack applications, testing it against our use cases?

The set of system and application metrics that should define the state of our RL environment, finding the right balance between giving the DO a meaningful view of the application and system status and not creating a too large state space.

7. ADS Ranking & Deploy

The role of the ranking and deployment module of Big Data Stack is to decide how to deploy the user's application and then operationalize that deployment via a container orchestration platform (e.g. Kubernetes). Ranking and deployment is part of the application deployment back-bone that enables a user to get their application running on a hardware cluster. Prior to ranking and deployment, the user will have defined in a conceptual manner what their application is comprised of and how the different services within that application interact. This conceptual definition will have been expanded into multiple candidate deployment pattern (CDP) playbooks representing different ways that the application/services can be mapped onto compute resources for deployment. Finally, these CDP Playbooks will have been benchmarked, providing estimated resource usage and quality of service information for each, creating Dimensioned Deployment (DD) Playbooks. Ranking and deployment takes these DD Playbooks and associated benchmarking information as input.

As its name suggests, ranking and deployment is split into two distinct components, namely: ADS (Application and Data Services) Ranking and ADS (Application and Data Services) Deployment. ADS Ranking is responsible for taking the different DD Playbooks and associated benchmarking information, and deciding which DD Playbook is the most suitable based on the user requirements and preferences. This has two uses within BigDataStack, namely: to determine what compute resources to request for a user's application when first deploying it; and to re-estimate compute resource needs in cases where a current deployment is predicted to miss one or more Service-Level Objectives. Meanwhile, ADS Deployment is responsible for taking the selected DD Playbook and using the configuration information contained within, to operationalize deployment of the user's application on the cloud infrastructure.

The initial design and implementation details for ADS Ranking and ADS Deployment were described in the previous version of this deliverable, i.e. D3.1. In particular, refer to D3.1 for component requirements specification and design, in addition to information about the early prototypes and a discussion on how we might evaluate the quality of the outcomes of ADS Ranking in particular. Note that over the last 11 months there were two minor terminology changes. First, in D3.1 the central state storage service was referred to as the Central Decision Tracker, it is now referred to as the Global Decision Tracker (GDT). Second, we previously did not distinguish between CDP Playbooks and those same playbooks after benchmarking information was added, which we now distinguish these as CDP (Candidate Deployment Pattern) Playbooks and DD (Dimensioned Deployment) Playbooks, respectively. In contrast, in this section we report on updates to the design and implementation of the two components, as well as provide an experimental evaluation of the current version of ADS Ranking, that have occurred between M12 and M23.

7.1. Requirements

To facilitate the understanding of the design as well as the challenges addressed by this component, the requirements related to this component have been brought from D2.3 and included into this section. Please note the following requirement tables are compiled together with the rest of requirements of BigDataStack in D2.3, and that they are included in here for the reader's convenience.

This section contains the requirements for both the ADS Ranking and ADS Deployment components, denotated as REQ-ADSR-XX and REQ-ADSD-XX, respectively.

	Id	Level of detail	Type	Actor	Priority
	REQ-ADSR-01	System	FUNC	Application Dimensioning Workbench	MAN
Name	Ingest Candidate Deployment Playbooks and Benchmarking Information				
Description	The Application Dimensioning Workbench sends a series of candidate deployment patterns (CDP) playbooks and benchmarking information to the ADS Ranking component. ADS Ranking needs to collect all these patterns for subsequent scoring/ranking based on the user requirements and preferences.				
Additional Information	Ingestion occurs via a common publisher/subscriber platform (RabbitMQ).				

Table 17 - Requirement (1) for ADS Ranking

	Id	Level of detail	Type	Actor	Priority
	REQ-ADSR-02	System	FUNC	Dynamic Orchestrator, Application Dimensioning Workbench	MAN
Name	Deployment Suitability Feature Extraction				
Description	Once a series of candidate deployment pattern playbooks and associated benchmarking information has been received, the next step is to determine how each pattern is predicted to perform based on the benchmarking information. In effect, this involves defining a series of functions that relate individual or groups of user requirements to the predicted performances produced by benchmarking. The output of this step is a vector representation for each CDP playbook, representing how that playbook is predicted to perform under different user requirements.				
Additional Information	Features produced here are dependent on the capabilities of the benchmarking system and the amount of information the user provides in terms of requirements and preferences.				

Table 18 - Requirement (2) for ADS Ranking

	Id	Level of detail	Type	Actor	Priority
	REQ-ADSR-03	System	FUNC	Dynamic Orchestrator, Application Dimensioning Workbench	MAN
Name	CDP Playbook Scoring (Heuristic)				

Description	Given a vector representation for a CDP Playbook, we next need to map this vector into a single score, representing how suitable that playbook will be overall (such that we can compare different CDP Playbooks). This involves combining the different elements within the vector (that each represent some aspect of pattern suitability, such as cost, or predicted compute wastage). The first version of this component will use a hand-tuned linear combination.
Additional Information	N/A

Table 19 - Requirement (3) for ADS Ranking

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSR-04	System	FUNC	Dynamic Orchestrator, Application Dimensioning Workbench	DES
Name	CDP Playbook Scoring (Supervised)				
Description	Given a vector representation for a CDP Playbook, we next need to map this vector into a single score, representing how suitable that playbook will be overall (such that we can compare different CDP Playbooks). This involves combining the different elements within the vector (that each represent some aspect of pattern suitability, such as cost, or predicted compute wastage). The second version of this component will learn how to combine the elements based on logging information from past deployments. Models may be non-linear in nature.				
Additional Information	Depends on REQ-ADSR-06.				

Table 20 - Requirement (4) for ADS Ranking

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSR-05	System	FUNC	Dynamic Orchestrator, Application Dimensioning Workbench	MAN
Name	CDP Playbook Selection				
Description	Once all candidate deployment patterns have been scored, the final step is to select one of those patterns to pass to ADS Deployment. In many cases this will simply involve selecting the highest scoring pattern. However, the user may have the option to select an alternative configuration at this stage.				
Additional Information	N/A				

Table 21 - Requirement (5) for ADS Ranking

	Id	Level of detail	Type	Actor	Priority
	REQ-ADSR-06	System	FUNC	Dynamic Orchestrator, Application Dimensioning Workbench	DES
Name	Supervised Model Training				
Description	To support REQ-ADSR-04, a supervised scoring model is needed. To react to changes in the deployment environment over time, this model needs to be frequently updated based on new information from current deployments. This model needs to be trained based on logging data being collected by the Triple Monitoring Framework.				
Additional Information	Requires logging information produced by the Triple Monitoring Framework and stored in the Central Decision Tracker.				

Table 22 - Requirement (6) for ADS Ranking

	Id	Level of detail	Type	Actor	Priority
	REQ-ADSR-07	System	FUNC	Dynamic Orchestrator	MAN
Name	CDP Playbook Re-Scoring				
Description	It is envisaged that in (rare) scenarios, an ongoing application deployment will fail to meet the user's quality of service requirements. For instance, this might occur due to assumptions on data input volumes being violated. In this case, we may not be able to solve this issue without fully redeploying the user application with different resources. To support such re-deployment activities, ADS Ranking supports a re-scoring function, where a previous set of CDP playbooks for a user's application can be re-scored based on updated preferences provided by the Dynamic Orchestrator, as well as data about how the previous deployment performed (and failed).				
Additional Information	N/A				

Table 23 - Requirement (7) for ADS Ranking

	Id	Level of detail	Type	Actor	Priority
	REQ-ADSR-08	System	FUNC	ADS Ranking	DES
Name	Deployment Dataset Generation				
Description	To support REQ-ADSR-06 and hence REQ-ADSR-04, significant volumes of logging data from past deployments are needed to enable effective model creation. To this end, a framework and methodology for generating this				

	data is needed. Such logging data can be produced through either benchmarking, live deployment of the end-user applications and via simulated application deployment.
Additional Information	Data storage for this task is handled by the Triple Monitoring Framework and Central Decision Tracker. Data generation is supported by deployments by the application dimensioning workbench and other dedicated deployment applications.

Table 24 - Requirement (8) for ADS Ranking

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSD-01	Stakeholder	FUNC	ADS Ranking	MAN
Name	Performance Measurability				
Description	Each environment should be measurable according to a set of characteristics, that is, Key Performance Indicators (KPIs).				
Additional Information	The KPIs considered must include: - vCPUs - Memory				

Table 25 - Requirement (1) for ADS Deploy

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSD-02	Stakeholder	FUNC	Application Engineer, Data Engineer	MAN
Name	Standards-based Playbook				
Description	The description of the environments and deployments (i.e., playbooks) will follow a specification language that is intuitive and as close (similar) as possible to well-known and widely-used schemas to describe software application deployments in cloud infrastructures, such as Docker Compose or Kubernetes Deployment.				
Additional Information	N/A				

Table 26 - Requirement (2) for ADS Deploy

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-ADSD-03	System	FUNC	Application Engineer, Data Engineer	MAN
Name	Standard deployment information				
Description	When communicating with other components, as described in Section 7.2, these components will use the playbook standard defined in REQ-RD-02.				
Additional Information	N/A				

Table 27 - Requirement (3) for ADS Deploy

	Id	Level of detail	Type	Actor	Priority
	REQ-ADSD-04	System	FUNC	ADS Ranking	MAN
Name	Application Scoring System				
Description	The ranking system evaluates each environment's deployment, which keeps track of the most suitable configuration for each application. When trying a deployment configuration for a new application, this ranking will be used to select the most suitable one.				
Additional Information	The evaluation needs to be performed following the measurements defined in REQ-RD-01.				

Table 28 - Requirement (4) for ADS Deploy

	Id	Level of detail	Type	Actor	Priority
	REQ-ADSD-05	Software	FUNC	Cluster Management	MAN
Name	Compatibility with Kubernetes				
Description	Since the technology used to run and orchestrate the applications is based on Kubernetes (OKD[10]). Thus, the ADS-Deployment component is required to be compatible with Kubernetes.				
Additional Information	The ADS-Deploy component should translate from the playbook standard defined in REQ-RD-01 into Kubernetes primitives.				

Table 29 - Requirement (5) for ADS Deploy

	Id	Level of detail	Type	Actor	Priority
	REQ-ADSD-06	System	PERF	ADS Ranking	MAN
Name	Synchronous communication				
Description	The communication with and within ADS Ranking and ADS Deploy must be done through an API REST.				
Additional Information	N/A				

Table 30 - Requirement (6) for ADS Deploy

7.2. Design Specifications

The design for this component (originally specified in Section 7 of D3.1) has remained valid for year 2 for the most part. The following sections describe the aspects of the design that have been updated during Y2.

[10] OKD - https://www.okd.io/

7.2.1. Connection with the Visualization Service

In the original design for ADS Ranking, it was envisaged that once the best deployment was identified, that choice would be immediately sent to ADS Deployment to operationalize application deployment. However, through discussions with the development team responsible for the user-facing platform, we realized that in some scenarios the user may wish to at least approve the recommended deployment, if not even further customise it based on their knowledge of the application. Hence, a design change was made to ADS Ranking, de-coupling it from ADS Deployment.

Under the new design, when a new DD Playbook is selected by ADS Ranking, it is now published to the Visualisation service as well as the Global Decision Tracker, in contrast to sending the selected pattern directly to ADS Deployment. The Visualisation service now visualises the DD playbook contents to the user, where they can either approve the configuration (which sends the DD Playbook on to ADS Ranking as before) or abort the deployment. This visualisation is shown in Figure 10.

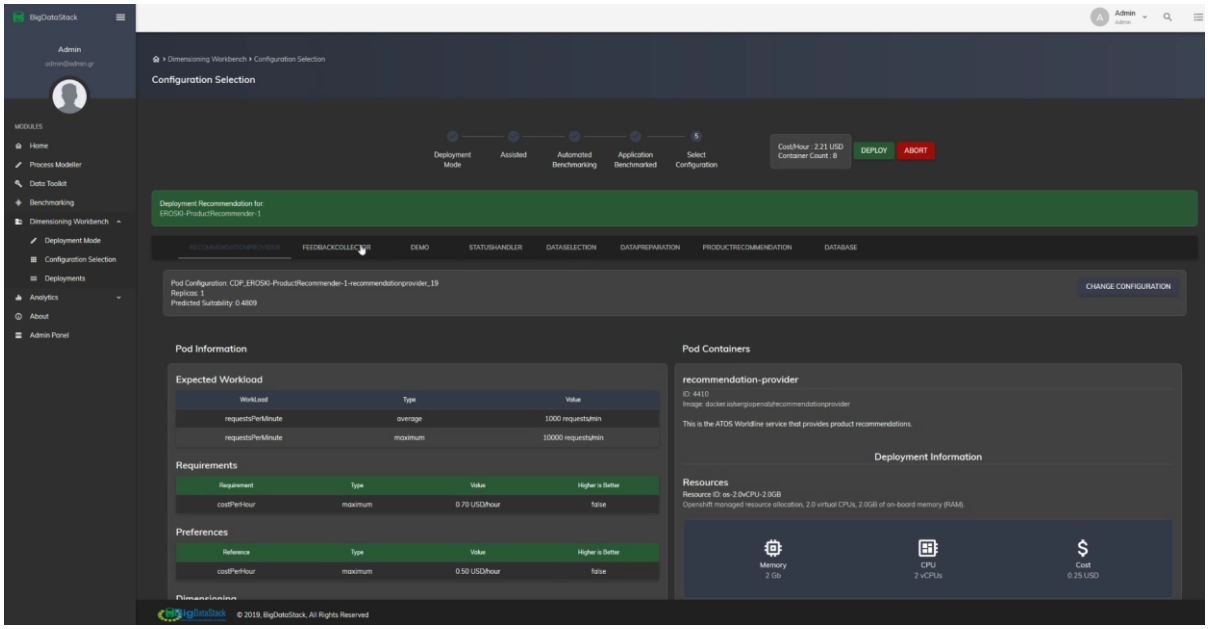


Figure 10 – CDP Playbook Visualisation and Approval Screen

7.3. Experimentation Outcomes

Given that ADS Ranking has reached its Tier 1 implementation that provides the base set of functionalities, we need to evaluate its performance at identifying effective and efficient deployment configurations. As its name suggests, ADS Ranking is a ranking service at its core, i.e. it ranks a set of items provided to it, which are DD Playbooks in our case. Some of those DD Playbooks will be more suitable than others. By suitability, we refer to whether the user's requirements and preferences will be met or exceeded, if we use that DD Playbook to deploy the user's application. Hence, we can measure how effective ADS ranking is for an application by evaluating to what extent the top-ranked DD Playbooks are suitable. By evaluating the effectiveness of ADS Ranking at deploying different types of application, we can determine the overall effectiveness of ADS Ranking as a whole. In this section we describe the experimental framework and setting we use to perform an

evaluation of ADS Ranking[11] in terms of dataset, methodology, metrics and baselines. We then report the performance of ADS Ranking Tier 1 and the baselines under this dataset and metrics.

7.3.1. Dataset

As discussed in D3.1 Section 7.5, the idea of producing an automatic system to estimate what resources are needed to deploy a user application is novel. Hence, there are not readily available standard datasets that we can leverage to evaluate ADS Ranking. Instead, for our initial evaluation we generate a new dataset. In effect, a dataset for this task can be considered to be comprised of five main parts:

- **BigDataStack Playbooks**: The definition of applications that we are going to deploy onto the cluster infrastructure. Each BigDataStack Playbook describes the services within a user's application, along with the quality of service factors that the user cares about in terms of hard requirements and softer preferences. However, it does not state how that application should be deployed in terms of resources to be allocated to the different services within that application.
- **Workload**: The workload for an application represents the amount of work that the application needs to do. For a real-time streaming application, this might represent the stream of records or requests that need to be processed. Meanwhile for batch operations, this would be the dataset or database that needs to be processed or queried.
- **CDP Playbooks**: For a BigDataStack Playbook that describes a single application, we also need a series of Candidate Deployment Pattern (CDP) Playbooks that describe the different ways that we might deploy that application on the cluster infrastructure in terms of resources (CPU, GPU, memory, per service). These CDP Playbooks are combined with benchmark performances (discussed below) to form the items that ADS Ranking scores and ranks (Dimensioned Deployment Playbooks).
- **Benchmark Performances**: As part of the ranking process, ADS Ranking utilizes predicted performance estimates produced by the Benchmarking (ADW Core) component of the Application Dimensioning Workbench. In effect, for each CDP Playbook, Benchmarking provides a series of indicators (features) about how well the application is expected to perform if deployed using the resources described within those CDP Playbooks. The combination of a CDP Playbook and this benchmark data forms what we refer to as a Dimensioned Deployment (DD) Playbook. A DD Playbook represents a single candidate deployment of a user's application, but also has all of the information that ADS-Ranking needs to predict its suitability.
- **Ground-truth Performances**: To evaluate to what extent each DD Playbook is, in fact suitable (rather than is predicted to be suitable), we need to have ground truth information about how the user's application would actually perform on the cluster

[11] Note that as the Openshift Application Simulator Adaptor (OASA) and its plugins that are being developed as part of WP6 T5.1 becomes more fully-featured, this will also provide further datasets that we can use to evaluate ADS Ranking. However, at the time of writing, the first version of this service has only just been completed, and hence is not used here.

infrastructure if deployed using those DD Playbooks. Note that this is different to what the Benchmark Performances provide, as those are only (predictive) estimates and are subject to error.

To produce our initial dataset, we first created 24 real-time stream processing applications. Each of these applications have different processing properties, such as start-up time, per-record processing time, memory usage, maximum throughput and more. We then defined three quality of service levels, which we refer to as medium, high and extreme, where each quality of service level specifies the response time bounds and cost for the application that are acceptable for different classes of user, as follows:

- Medium QoS:
  - Requirement: Response Time less than 200ms, Cost less than $1.9/hour
  - Preference: Response Time less than 100ms, Cost less than $0.7/hour
- High QoS:
  - Requirement: Response Time less than 150ms, Cost less than $1.9/hour
  - Preference: Response Time less than 70ms, Cost less than $0.7/hour
- Extreme QoS:
  - Requirement: Response Time less than 70ms, Cost less than $1.9/hour
  - Preference: Response Time less than 50ms, Cost less than $0.7/hour

Next, we generated one BigDataStack Playbook for each unique application and quality of service pair, resulting in 72 BigDataStack Playbooks (24 applications x 3 QoS levels). For this first dataset, we use only a single stream processing workload, where the average input rate is 300 requests per second, with a peak input rate of 500 requests per second. We refer to the combination of one of the BigDataStack Playbooks with this workload as an experimental scenario.

For each of the generated BigDataStack Playbooks, we then submitted them to the ADW Pattern Generation component deployed on our local testbed, which in turn produced CDP Playbooks for each. Based on the underlying available hardware, each BigDataStack Playbook has 35 possible deployment configurations, hence 35 CDP Playbooks are generated per experimental scenario, creating a total of 2,520 CDP Playbooks (72 scenarios x 35 deployment configurations). At this point, we deployed each of the 2,520 CDP Playbooks in turn, collecting resource usage and quality of service information. More precisely, we tracked average and peak CPU and memory usage, along with average and peak response time. In this way, we collected our ground truth performances. There was no competing for resources during these tests and so performances should be comparable between scenarios.

At the time of this experiment, the WP5 benchmarking component did not yet support our 24 applications, hence we could not directly use it to obtain the benchmark performances we need for the dataset. Instead, to collect our benchmarking performances, we submitted each CDP Playbook to a local Benchmarking Simulation service that we developed, which simply takes the true ground truth performances and generates benchmark performances from them, with a randomised degree of performance error (+/- 20%) added to represent

imperfect benchmarking. Combining each of the 2,520 CDP Playbooks with their imperfect benchmarking information, we create the 2,520 DD Playbooks that we will have ADS-Ranking rank. In this way, we now have a full dataset that we can use to evaluate ADS Ranking.

7.3.2. Metrics

For each of the 72 experimental scenarios, ADS Ranking will output a ranking of the associated 35 DD Playbooks. However, to determine how effective each of these rankings are, we need a means to determine the suitability of each DD Playbook within the rankings. During dataset creation described above, we have two pieces of information to aid in this task. First, we have the quality of service requirements and preferences set by the user. Second, our ground truth performances tell us how well each DD Playbook actually performed. Hence, we need a mapping function that takes these two pieces of information and produces a suitability score, where a higher score indicates that the user's requirements and preferences were better met (while also minimising cost). Hence, we use a simple scoring function that produces a suitability score between 0 and 3, where 0 indicates that the DD Playbook was unsuitable and 3 indicates that all requirements and preferences were met. Scoring is performed as follows:

- If either response time or cost exceeds the user requirement, the DD Playbook receives a score of 0.
- If the user requirements are met, but none of the user preferences are met, the DD Playbook receives a score of 1.
- If the user requirements are met, and either (but not both) of the user preferences are met, the DD Playbook receives a score of 2.
- If all requirements and preferences are met, then the DD Playbook receives a score of 3.

We use this function to produce a suitability score/label for each of the 2,520 DD playbooks.

Once the DD Playbooks have been scored, we need to use these scores to evaluate the performance of ADS Ranking as a whole. To do so, we use standard ranking metrics from the information retrieval literature. In particular, we report:

- *Precision@5*: This evaluates whether the top ranked DD Playbook was suitable (had a score equal to or greater than 1) for the user's application
- *Mean Average Precision (MAP)*: Average precision (at a particular rank) is the proportion of suitable (has a score equal to or greater than 1) DD Playbooks down to that rank. MAP is average precision calculated at the maximum rank (35 in this case) over multiple application deployments. [13]
- NDCG@5: Discounted Cumulative Gain (DCG) is a measure of the usefulness, or gain, of an item based on its position in a ranking. Total gain is accumulated starting from the top of the result list (ranking) and moving downwards to a set rank (@N). Gain of each result is discounted at lower ranks and can incorporate different (suitability) grades. Hence, unlike the above two metrics, this metric considers whether the preferences were met in addition to the requirements. NDCG is DCG normalized

across (in our case) different application deployments to account for some deployments being easier to find suitable patterns for than others. [9]

7.3.3. Baselines

Using the above dataset and metrics, we can score ADS Ranking in terms of its effectiveness. However, such a score in isolation can be misleading, as it does not provide us information about how difficult the task is. Hence, we also need reference baselines to compare against, providing us context. As this is a new task, there are no standard baselines. Hence, we propose two new baselines here, representing simple strategies that a human might employ when selecting a DD Playbook:

- **RankByCost**: This baseline simply ranks each DD Playbook by its deployment cost on the cluster hardware, where the cheapest deployment is ranked first. In particular, cost is calculated as the sum of the cost of the requested resources across the services defined in the DD Playbook, where a mapping between resources and a US dollar cost from a commercial cloud provider (Amazon Web Services EC2) is used.
- **MidTierFirst**: This second baseline represents a user selecting resources that are in the middle of the available range, as they don't know what they need. To represent this, we manually ordered the available DD Playbooks by requested resources, placing those using mid-tier hardware first, followed by high-tier hardware, and finally putting the lowest-tier hardware at the bottom of the ranking.

7.3.4. ADS Ranking Performance Results

In this section we report the performance of the ADS Ranking component against the baselines summarized above. Figure 11 reports the performance of ADS Ranking Tier 1 in terms of Precision@5, MAP and NDCG@5, * indicates a statistically significant increase in performance over the MidTierFirst baseline (paired t-test, $p<0.05$). As we can see from Figure 11, ADS-Ranking (Tier 1) is significantly better at recommending deployment configurations than the baselines tested (e.g. 0.5582 vs. 0.2793 NDCG@5). Moreover, the increase in performance is larger under Precision@5 and MAP (that only consider the user requirements) than under NDCG@5 (which factors in requirements and preferences), indicating that ADS Ranking is much better at meeting at least the minimal user requirements. On the other hand, current average performance of ADS Ranking appears to be around 0.55, indicating that there is still significant scope to improve ranking performance. Indeed, in year three we will be investigating on how to achieve further increases in performance using learning to rank models in ADS Ranking Tier 2.

Approaches	Precision@5	MAP	NDCG@5
RankByCost	0.0111	0.1407	0.2793
MidTierFirst	0.1778	0.2260	0.3532
ADS Ranking Tier 1	**0.5500***	**0.5204***	**0.5582***

Figure 11 – ADS Ranking Performance

7.4. Implementation and Integration Highlights

The following sections describe relevant implementation and integration work carried out during Y2.

7.4.1. Re-Implementation to Decrease Latency

The original Tier 0 implementation of the ADS Ranking component was created in a modular fashion using the Apache Spark framework and its Spark Streaming module. The core idea underpinning this design decision was two-fold. First, it provided a convenient means to compartmentalize functionality into data transformation operations which were independent and easily scalable to deal with high volume request loads. This is important, as complex applications comprised of many services may have thousands of potential deployment options (represented as DD Playbooks) that need to be considered. Second, Apache Spark already provides built-in machine learning capabilities that could be leveraged to support model learning within ADS Ranking (later in Tier 2).

However, after the first implementation (Tier 0) of ADS Ranking was complete, we identified an issue with this design choice, namely a lack of responsiveness when connected to the BigDataStack User Interface component. At its core, Apache Spark is a batch-oriented platform, designed to process groups of data at one time. The Spark Streaming module that we use enables pseudo-real-time computation by reducing the batch sizes to typically only 10's of items (DD Playbooks) at a time, referred to as micro-batches. However, while playbook computation was fast, we observed multiple seconds of wasted time between these micro-batches, due to the set-up and shut-down of each micro-batch.

As a result, we refactored the initial Apache Spark implementation using a different platform (Apache Flink). This platform is a similar open source project to Apache Spark, in that it provides a compute framework for JVM-based languages, but was designed from the ground-up for real-time computation. As a result of this change, we managed to eliminate the additional latencies that were introduced by Spark's micro-batching, reducing the delay in ADS Ranking processing time by around 4 seconds on average during our initial testing.

7.4.2. ADS Ranking Tier 1 Implementation

As per the initial design and development plan summarised in D3.1 Section 7.6, during the second period covered by this current deliverable, the second iteration of ADS Ranking (Tier 1) was successfully implemented. This second version of the ADS Ranking component integrates directly with the first iteration of the application dimensioning workbench to obtain benchmarking features. The Tier 1 implementation also includes the first implementation of the re-ranking functionality (REQ-ADSR-07).

Post M18, development efforts have been shifted to the creation of the underlying training datasets needed to train the Tier 2 implementation of ADS-Ranking. Further details regarding this can be found in D5.2 Section 8.

7.4.3. ADS Deploy Implementation

Following the initial design, the ADS-Deploy component has been implemented in a two tiers structure.

The first tier considered releasing a prototype which integrated with the existing technologies referenced in REQ-ADSD-05. This prototype received a DD Playbook, which followed the structure defined in D2.5) using a RESTful interface. Then, the component translates the DD Playbook into a Kubernetes' compatible JSON file, and sends this file to the infrastructure manager. It utilizes the OKD's managed API REST to ensure that the deployment is correctly communicated. In BigDataStack, a deployment is communicated to OKD using:

- DeploymentConfig: Including the information on the different pods and containers.
- Services: To manage the access point to the application deployed on BigDataStack.
- Routes: Providing the hostname information to access the pods from the outside world.

For further information on the design of the ADS-Deploy prototype, please refer to D2.5.

In a second tier, the ADS-Deploy component has been integrated with the rest of the system. This integration has continued being done using the RESTful API, which is reached by the ADS-Ranking component whenever it becomes necessary. The need for a REST API has been an architectural decision, due to the synchronous and very dependent nature of both ADS-Deploy and ADS-Ranking components.

7.5. Next Steps

It is currently envisaged that there will be two further releases of the ADS Ranking component and one release of the ADS Deployment component during BigDataStack, integrating more advanced functionality:

- ADS Ranking
    - *Tier 2*: This version will transition from using current heuristic scoring of CDP Patterns to the first version of our proposed learning to rank approach.
    - *Tier 3*: This version will include our second iteration of the learning to rank approach with support for reinforcement learning from live data collected by the Triple Monitoring Framework.
- ADS Deploy
    - *Tier 3*: This version of the ADS Deploy will support enhanced integration with the Global Decision Tracker for monitoring application deployment state in cases where immediate deployment is not possible.

8. Triple Monitoring & QoS Evaluation

The triple monitoring component collects and stores several metrics regarding the performance of a deployment at an application, data service and resource cluster level. These metrics are used to dynamically adapt the environment and ensure the best QoS (Quality of Service) to the user. When a user requests a service from BigDataStack, a minimum QoS is agreed between the user and the system. At runtime, certain metrics or Key Performance Indicators (KPI) are collected by the Triple Monitoring Engine and evaluated against the agreed Service-Level Objectives (SLOs) by the QoS Evaluator.

8.1. Requirements

To facilitate the understanding of the design as well as the challenges addressed by this component, the requirements related to this component have been brought from D2.3 and literally included into this section. Please note the following requirement tables are compiled together with the rest of requirements of BigDataStack in D2.3, and that they are included in here for the reader's convenience.

	Id	Level of detail	Type	Actor	Priority
	REQ-TME-01	System	FUNC	Application Engineer, Data Engineer	MAN
Name	Metrics pusher				
Description	The metric pusher retrieves KPI data, cleans them and ingests them into the monitoring collector (*Prometheus)*.				
Additional Information	The metrics pusher is used when the exporter approach is impossible to apply—since Prometheus exporters require an HTTP server to publish metrics for the monitoring collector, components that lack this service need an alternative. Besides, this solution will be very useful for getting application specific metrics. This component is a REST-API and Prometheus Exporter which receives KPI data over HTTP in JSON format, then format them and ingest them into Prometheus.				

Table 31 - Requirement (1) for Triple Monitoring Engine

	Id	Level of detail	Type	Actor	Priority
	REQ-TME-02	System	FUNC	QoS Evaluation	DES
Name	RESTful API for accessing the collected monitoring metrics				
Description	The metrics are accessible through a RESTful API.				
Additional Information	This component translates client's requests to Prometheus request compatible. Grafana[12] will be used for visualization.				

Table 32 - Requirement (2) for Triple Monitoring Engine

[12] Grafana. https://grafana.com/

	Id	Level of detail	Type	Actor	Priority
	REQ-TME-03	System	FUNC	QoS Evaluation, Dynamic Orchestrator	MAN
Name	Metrics publication				
Description	Measurements stored in the metrics repository must be periodically published through a publisher/subscriber mechanism. The publication of measurements must start and stop following the request made by the QoS Evaluation, as this component is also responsible for managing the life cycle of the quality monitoring and evaluation tasks (see REQ-QOS-07).				
Additional Information	The monitoring metrics getter is implemented using RabbitMQ[13].				

Table 33 - Requirement (3) for Triple Monitoring Engine

	Id	Level of detail	Type	Actor	Priority
	REQ-TME-04	Software	FUNC	Application Engineer, Data Engineer	DES
Name	Spark compatible				
Description	The triple monitoring engine monitors the performance of Apache Spark[14], which is used in the BigDataStack project as an analytics engine for Big Data, and thus needs to be compatible with this technology.				
Additional Information	Monitoring Spark is done using the Spark measure project, which can be embedded in a Spark application to allow the collection of some metrics after each SQL execution. Those metrics are sent to *push gateway* to be exported to Prometheus.				

Table 34 - Requirement (4) for Triple Monitoring Engine

	Id	Level of detail	Type	Actor	Priority
	REQ-TME-05	Software	FUNC	Application Engineer, Data Engineer	DES
Name	LeanXcale compatibility				
Description	LeanXcale database[15] already uses Prometheus for its monitoring subsystem. However, the integration relies on deployments that require the manual reconfiguration for the Prometheus in cases of scaling actions. Thus, it should be extended to consider automatic re-deployments driven by an elasticity action and automatically reconfigure the integration with				

[13] RabbitMQ. https://www.rabbitmq.com/
[14] Apache Spark. https://spark.apache.org/
[15] LeanXcale. https://www.leanxcale.com/

	the Prometheus monitoring system. In these scenarios, LeanXcale should reconfigure its integration with the existing Prometheus deployment on the run-time and provide monitoring information for the new nodes.
Additional Information	N/A

Table 35 - Requirement (5) for Triple Monitoring Engine

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TME-06	Software	FUNC	Application Engineer, Data Engineer	DES
Name	OKD compatibility				
Description	The Triple Monitoring Engine (TPE) monitors the performance of OpenShift OKD[16], which is the baseline technology used in the orchestration of containers. Therefore, the TME should be compatible with this technology.				
Additional Information	N/A				

Table 36 - Requirement (6) for Triple Monitoring Engine

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TME-07	Software	FUNC	Application Engineer, Data Engineer	DES
Name	CEP compatibility				
Description	The Triple Monitoring Engine (TME) monitors the performance of the UMP CEP (Complex Event Processing), which is used in the BigDataStack project as a streaming engine for processing data in real-time. Therefore, the TME needs to be compatible with this technology. The integration with TME is done by federating Prometheus instances, that is, connecting the CEP' Prometheus-based monitoring system into DECENTER Prometheus-based central monitoring system.				
Additional Information	The CEP exposes several monitoring metrics that are exported to Prometheus.				

Table 37 - Requirement (7) for Triple Monitoring Engine

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TME-08	Software	FUNC	Application Engineer, Data Engineer	DES

[16] Openshift OKD (Origin Kubernetes Distribution). https://www.okd.io/

Name	Minio compatibility
Description	The Triple Monitoring Engine (TME) monitors the performance of Minio[17], which is used for object storage in the system. Therefore, the TME needs to be compatible with this technology.
Additional Information	N/A

Table 38 - Requirement (8) for Triple Monitoring Engine

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TME-09	Software	FUNC	Application Engineer, Data Engineer	DES
Name	OpenStack Networking Services compatibility				
Description	The Triple Monitoring Engine (TME) should monitor the performance of the internal network connecting the different containers inside a deployed application. BigDataStack networking resources and services are provided through a solution stack combining OpenShift, Kuryr and Neutron, that is, the networking services from OpenStack (see requirements REQ-CM-02, REQ-CM-03 and REQ-CM-04). This means that networking is ultimately provided by OpenStack and hence the need for integrating of TME with it to have networking monitored.				
Additional Information	Cluster Management (OpenShift) and Information-Driven Networking components interoperate to provide reliable networking to the applications and services deployed on the BigDataStack platform.				

Table 39 - Requirement (9) for Triple Monitoring Engine.

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TME-10	Software	FUNC	Application Engineer, Data Engineer	MAN
Name	Persistently store the monitoring metrics				
Description	The Triple Monitoring Engine (TME) should use a database for persistently storing monitoring metrics and is connected to Prometheus via PrometheusBeat.				
Additional Information	Metrics saved persistently will be used for historical reason.				

Table 40 - Requirement (10) for Triple Monitoring Engine

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TME-11	Software	FUNC	Application	ENH

[17] Minio Private Cloud Storage- https://www.minio.io/

				Engineer, Data Engineer	
Name	Spark Monitoring Pushgateway				
Description	This component is used to gather metrics from Spark and ingest them into the metrics collector.				
Additional Information	The connection between this component and the applications will use HTTP.				

Table 41 - Requirement (11) for Triple Monitoring Engine

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TME-12	Software	FUNC	Application Engineer, Data Engineer	ENH
Name	Metrics visualization				
Description	The metrics must be shown to the end-user via a graphical interface. Grafana is used for metrics' visualization.				
Additional Information	Grafana[18] is configured for receiving metrics from two sources (Prometheus, InfluxDB).				

Table 42 - Requirement (12) for Triple Monitoring Engine

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-TME-13	System	FUNC	QoS Evaluation	DES
Name	Metrics aggregation				
Description	The metrics publication (through publisher/subscriber pattern) process (see REQ-TME-03) should be also in charge of aggregating measurements in periods so the aggregated metrics can be then used by the QoS Evaluation to work in base of confidence levels and time intervals (see REQ-QOS-06).				
Additional Information	The aggregation function to use will be quantiles/percentiles, depending on the nomenclature we use (see REQ-QOS-06 for more details).				

Table 43 - Requirement (13) for Triple Monitoring Engine

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-QOS-01	System	FUNC	Developer	MAN
Name	Regular recording of QoS metrics				
Description	When a user's application is deployed, the Triple Monitoring Engine and QoS Evaluation monitors that application, tracking statistical information				

[18] Grafana - https://grafana.com/

	about its operation and associated QoS data, including network, data storage, virtualization layers, etc. This data is needed to support the learning of ranking models by ADS-Ranking service (part of Application and Service Deployment; see REQ-ADSR-03) and regularly saved in a centralised data store for later access.
Additional Information	N/A

Table 44 - Requirement (1) for QoS Evaluation

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-QOS-02	System	FUNC	Developer	MAN
Name	QoS violation alert				
Description	If the system does not respect the agreed QoS, an alert is raised.				
Additional Information	This alert is used internally to evaluate the performance of an environment, relating to REQ-RD-004.				

Table 45 - Requirement (2) for QoS Evaluation

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-QOS-03	System	FUNC	Developer	DES
Name	QoS violation monitoring				
Description	QoS violations are also monitored and shown to the user/admin.				
Additional Information	N/A				

Table 46 - Requirement (3) for QoS Evaluation

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-QOS-04	System	PERF	Dynamic Orchestrator	DES
Name	Asynchronous rich notification of QoS violations				
Description	QoS violations should be notified by means of a publisher/subscriber mechanism, together with the id of the metrics to which the SLO applies.				
Additional Information	The main consumer of the SLA violations notifications is the Dynamic Orchestrator.				

Table 47 - Requirement (4) for QoS Evaluation

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-QOS-05	System	PERF	Dynamic Orchestrator	MAN
Name	Ability to detect spike violations of QoS				

Description	The QoS Evaluation must consider all data points (i.e. measurements) stored in the monitoring system's time series database when computing SLO violations to cover spikes that may be missed by any sampling method otherwise.
Additional Information	N/A

Table 48 - Requirement (5) for QoS Evaluation

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-QOS-06	System	PERF	Dynamic Orchestrator	MAN
Name	Evaluate aggregated behaviour with a certain level of confidence				
Description	The QoS Evaluation must consider the aggregated measurements during a given time window to determine the compliance of the SLO for the most part or a given period or time window, to avoid notifying violations due to outliers or sporadic spikes in the measurements. We define "for the most part" as the level of confidence we can have in the evaluation of the SLO.				
Additional Information	There exist different ways in which we can "assess" a group of data points or measurements to determine whether they comply with the objective "for the most part". The most common way is to aggregate data points in groups of *n* and determine whether the entire group complies with the objective, and using aggregation functions such as quantiles/percentiles. Thus, the SLO evaluation need to follow the following norm: - ***Metric*** < ***objective*** for ***percentage*** of measurements collected in ***time window*** For example, that: - *Response time* < *900ms* for *99%* measurements collected in *10min* This percentage can be calculated as the percentile 99th or 0.99 quantile (also known as 99% quantile), depending on the nomenclature we use.				

Table 49 - Requirement (6) for QoS Evaluation

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-QOS-07	System	FUNC	Dynamic Orchestrator	MAN
Name	Management of the quality evaluation tasks				
Description	The QoS Evaluation must be responsible for managing the life cycle of quality monitoring and evaluation tasks for the applications and services deployed in the BigDataStack platform. These tasks must be initiated and stopped following the lifecycle of the application and service deployments, i.e., to be active only when the corresponding deployment is running.				
Additional Information	N/A				

Table 50 - Requirement (7) for QoS Evaluation

8.2. Design Specifications

Figure 12 shows the high-level architecture of the Triple Monitoring Engine (TME) and QoS Evaluation components, following the requirements defined in Section 8.1. As it is shown, the metrics collector is in a central place, receiving information from the compatible technologies exploiting Prometheus' federation feature (REQ-TM-8, REQ-TM-9 and REQ-TM-10) and scraping all endpoint conformably to its configuration file. Since Prometheus' retention period[19] is limited, Prometheus Beat requests metrics each interval (1 second by default) then ingests these metrics into RabbitMQ and Logstash. Metrics sent to Logstash are saved to Elasticsearch for ad-hoc access and historical purposes. Those who are published to RabbitMQ are redirected to queues based on the subscriptions list handled by the manager.

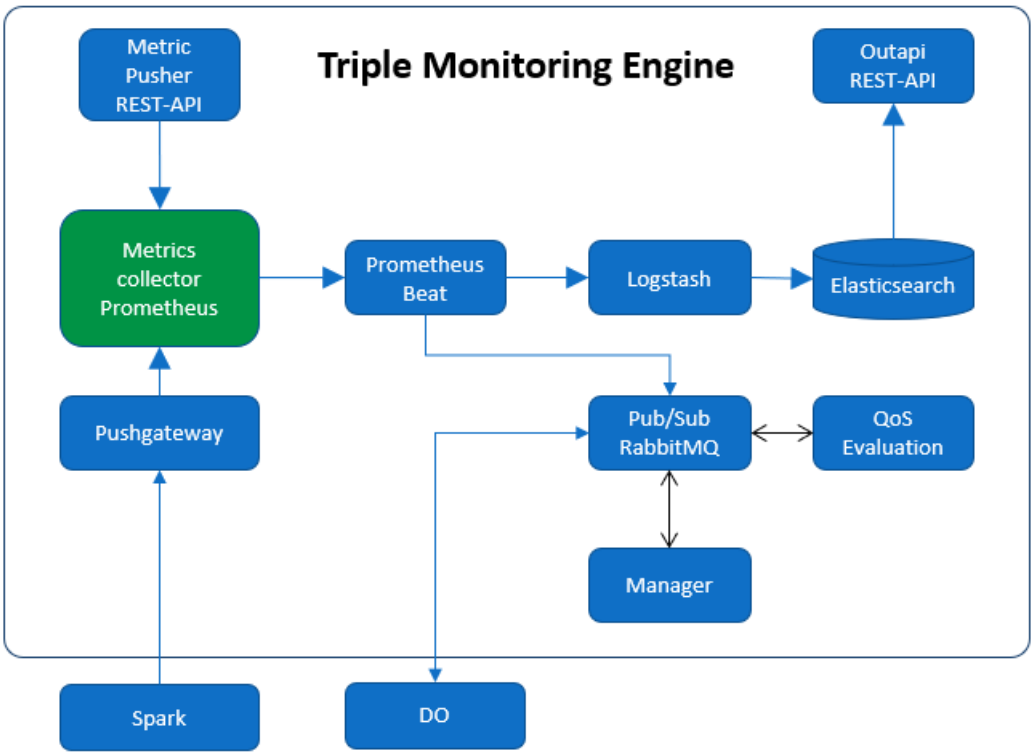


Figure 12 – Triple Monitoring Engine & QoS Evaluation – conceptual view.

The QoS (Quality of Service) Evaluation guarantees the compliance of a given KPI (Key Performance Indicator) with an SLO (Service-Level Objective) for the most part of a given period or time window. We define "for the most part" as **the level of confidence** we can have in the evaluation of the SLO. There exist different ways in which we can "assess" a group of data points or measurements to determine whether they comply with the objective "for the most part". One way is to aggregate data points in groups of *n* and determines whether the group as a whole complies with the objective. There are different aggregation functions we can use: from quantiles/percentiles to mean (average) and

[19] Prometheus retention period, https://prometheus.io/docs/prometheus/latest/storage/

median; we chose the former percentile[2]. In other words, that, metric's value is lower or higher than the objective for the percentage of measurements collected in the time window. The TME provides through the so-called Manager a functionality dedicated to the specific needs of the QoS Evaluation; the real-time stream of values of a given percentile of a bucket of measurements collected during a specific time window. Notice a 95th percentile will be the value which is greater than or equal to 95% of values collected in that bucket of measurements.

If a condition is not met, then the QoS Evaluation component raises an alert to the system, which is sent to the manager, through the Pub/Sub mechanism. Figure 12 shows the interaction between the Triple Monitoring Engine (based on Prometheus) for KPI monitoring and RabbitMQ for publication of metrics. Prometheus focuses on collecting metrics. RabbitMQ is used for managing the messages between components; one of the most important messages are those publishing metrics from Prometheus to the Manager and the QoS Evaluation. Subscribers, such as the QoS Evaluation, register in the Pub/Sub service and are notified every time that there is new data collected by Prometheus.

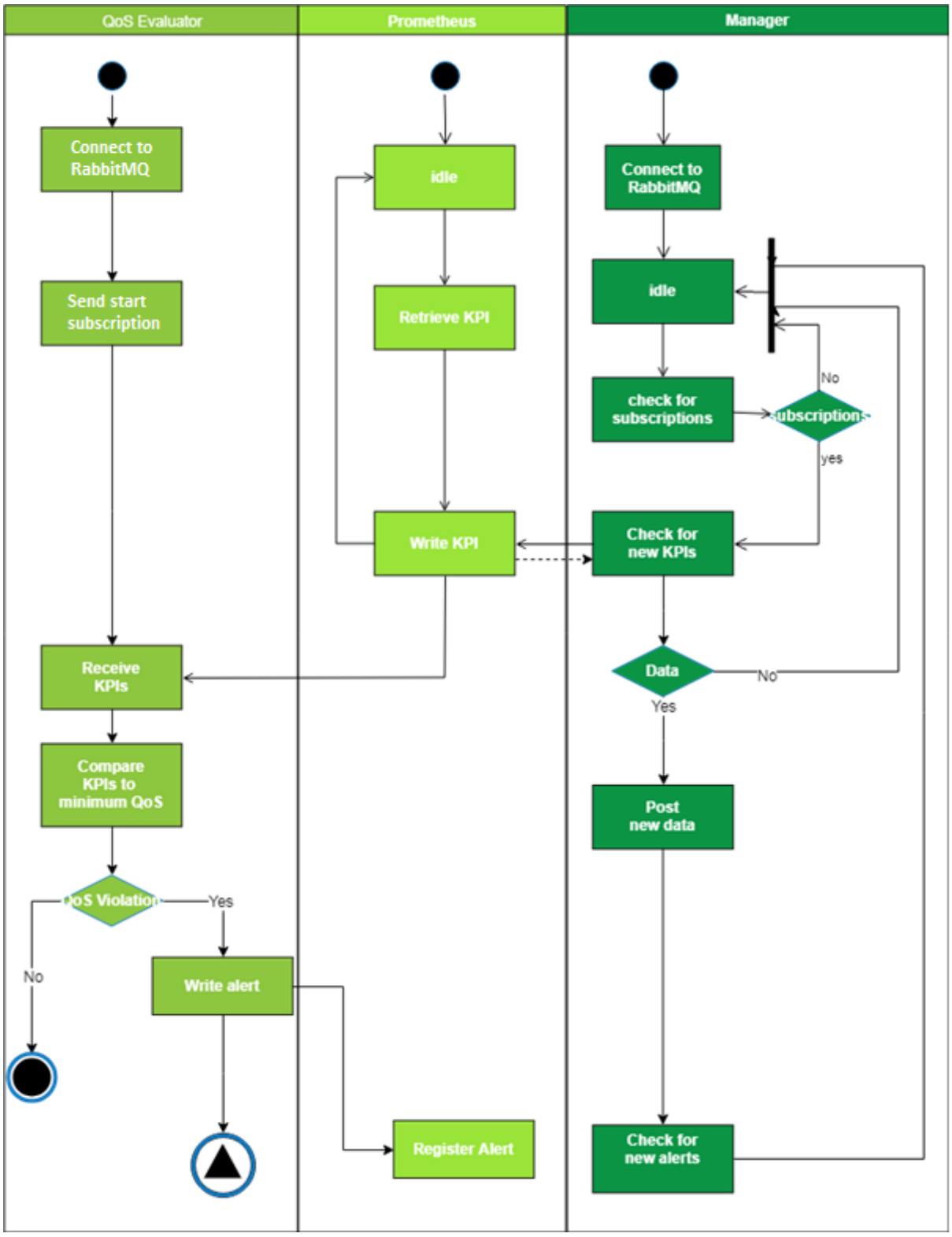


Figure 13 – Interaction between monitoring and QoS Evaluation components.

Figure 13 shows the interaction between the TME and QoS Evaluation, and the rest of the system. As described before, the Triple Monitoring Engine can provide aggregated information on the performance of the system and compare it against the expected QoS. The information on performance is delivered through the pub/sub system, and it starts providing it after a petition from the QoS evaluator. Following this comparison, the QoS

Evaluator component triggers and alert every time that the minimum agreed QoS is not respected. This alert is intercepted by the Dynamic Orchestrator component, which decides if it is necessary to re-deploy the monitored application.

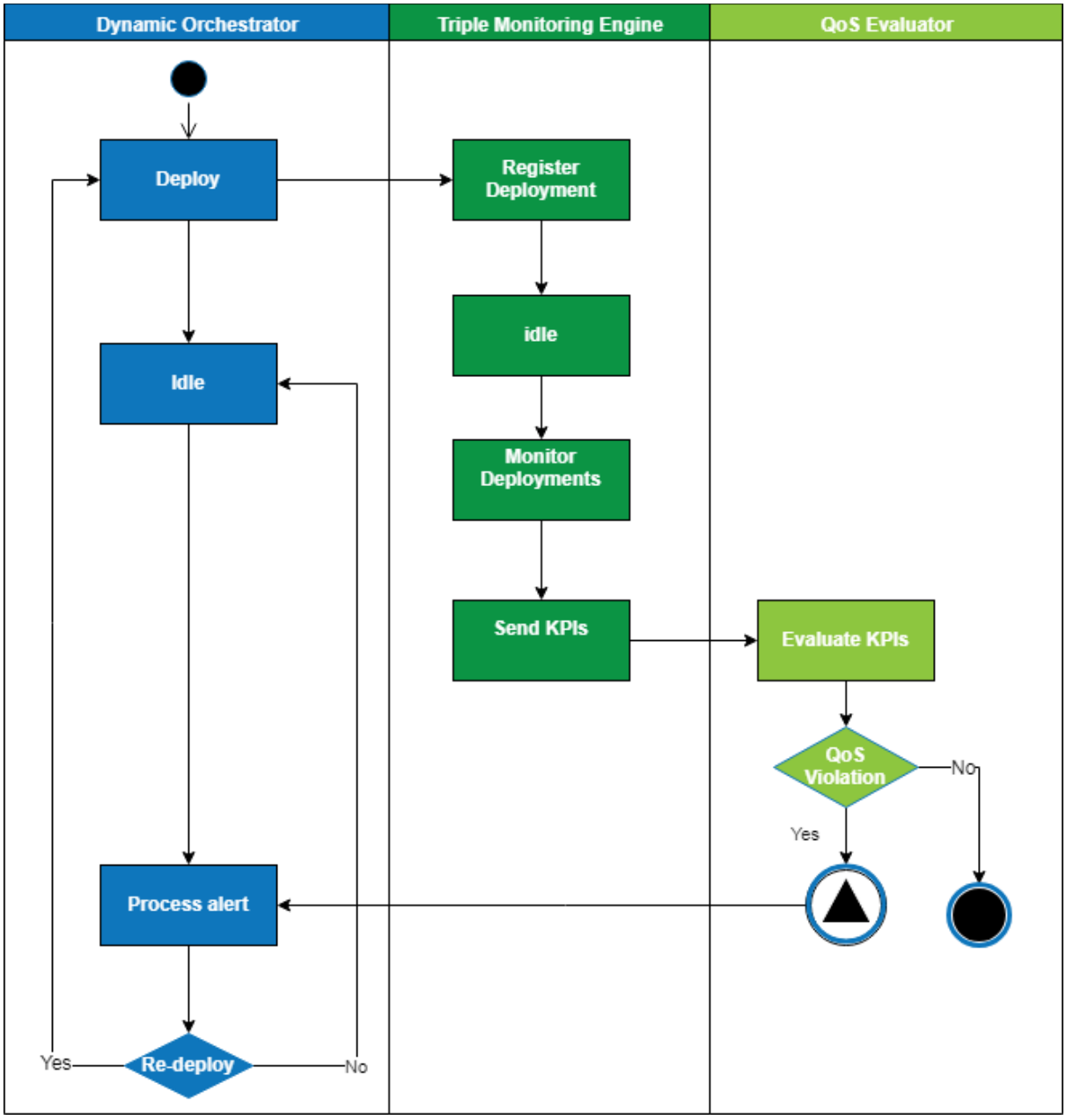


Figure 14 – Interaction between Triple Monitoring Engine, QoS Evaluation and ADS Deploy components.

8.3. Experimentation Outcomes

No individual or specific experiments are conducted for this component; the Triple Monitoring engine and QoS Evaluation (QoSE) play a supportive role to the components bringing the intelligence to the DDIM capability: the ADS Ranking & Deploy and the Dynamic Orchestrator (DO). Therefore, for experiments where the TME & QoSE participate are engaged please go to Sections 5 and 6.

8.4. Implementation and Integration Highlights

The following sections describe relevant implementation and integration work carried out during Y2.

8.4.1. QoS Evaluation Confidence Levels

The QoS Evaluation need to request the Manager to start aggregating and computing a certain percentile on a specific metric (KPI). In response, the Manager publishes the percentile to the queue the QoS Evaluation is waiting for. This request needs to contain the name of the queue to publish the percentile in, the name of the request and a list where each element is an object composed by the name of metrics, the percentage, the name of the application producing the corresponding metric, the interval and the time window (time span). This request has the following format:

```
{
"request":"qos_start",
"queue":"qos", "metrics":[
  {
    "application":"tester",
    "metric":"scrape_duration_seconds",
    "interval":10,"percentage":90}
  ]
}
```

As explained in the design specifications section, this integration between the QoS Evaluation and the Manager is made fully synchronous through the RabbitMQ messaging (publisher/subscriber) system.

The manager creates a bucket based on the interval of time specified in the request, then compute the percentile considering the percentage.

This percentage can be calculated as the percentile 99th or 0.99 quantile (also known as 99% quantile), depending on the nomenclature we want to use. The formula is the following:

```
index = (percentage)*(size+1)
```

where “size” is the amount of observation in the given time window.

Algorithm of percentile computation

```
Input: list_of_value, percentage
list_of_value.sort()
```

```
size = get_size_of_list(list_of_value)

index = percentage * (size + 1)

If index == size then

      index = size – 1

return list_of_value[index]
```

Output:


```
{

"application":"tester",

"metric":"scrape_duration_seconds",

"percentile":"0.016867146",

"request":"qos"

}
```


This computation will be periodically performed until the manager receives a request to stop.

The integrations with LeanXcale database, UPM CEP and Apache Spark to collect metrics from those systems were fully explained in D3.1, Sections 8.2.1, 8.2.2 and 8.2.3, respectively.

8.5. Next Steps

The following features in the roadmap of this component are:

- Triple Monitoring Engine (TME)
    - Collect metrics from the resources cluster manager (Openshift/Kubernetes) through the Prometheus federation mechanism.
- QoS Evaluation (QoSE)
    - Management of the evaluation lifecycle so the main consumer of the QoSE services can create, start, pause, resume and delete QoS evaluation jobs.

9. Information-Driven Networking

The Information-Driven Networking mechanisms provide a set of functionalities for traffic engineering and network management in the cloud infrastructure by taking into consideration that the services / applications, while running, have requirements which should be fulfilled in real-time and close to real-time settings and need to be discoverable, accessible and exposed on the network safely. These requirements concern the efficient Domain Naming System (DNS) resolution for services and pods among different compute nodes, services prioritization, the satisfaction of time critical constraints and security aspects.

All these properties are controlled by means of labels and pods selectors which are defined and have as effect to enforce specific network policies. This is required in order to make the service / application accessible from any node within the private cluster and through the Internet. The RedHat OpenShift built on top of the Kubernetes Container Management System supports a networking model which assumes that pods can communicate with other pods, regardless of which host they land on. Every time a pod is launched, a cluster-private-IP address is assigned in order not to explicitly create links between pods or map container ports to host ports. This means that containers within a pod can all reach each other's ports on localhost, and all pods in a cluster can see each other without Network Address Translation (NAT). This is a network engineering novelty which is realized by integrating Kuryr into the OpenShift and enables to avoid the double encapsulation problem due to using two (2) different overlays (OpenStack SDN and OpenShift SDN on top). It has been presented in more details in the section presenting the Cluster Management (cf. Section 5). All the microservices are packaged with a sidecar that intercepts incoming and outgoing calls for the services, providing the hooks needed to externally manage and control routing, telemetry collection, and policy enforcement for the whole application. This ensures advanced performance and security in the cloud infrastructure and is achieved by defining rules. These rules specify the connections that are allowed or not allowed to specific services or specific nodes in the cloud infrastructure of the BigDataStack project and support a set of features including (dis)enabling the enforcement of specific policies, rate limits to dynamically adjust the traffic to a service, control headers, routing and denials, white/black listing.

The outcome of the Information-Driven Networking mechanisms will be to translate these requirements into networking primitives that achieve the desired dissemination, regulatory compliance and sharing of the information in the BigDataStack environment.

9.1. Requirements

To facilitate the understanding of the design as well as the challenges addressed by this component, the requirements related to this component have been brought from D2.3 and literally included into this section. Please note the following requirement tables are compiled together with the rest of requirements of BigDataStack in D2.3, and that they are included in here for the reader's convenience.

	Id	Level of detail	Type	Actor	Priority
	REQ-IDN-01	System	FUNC	Data Scientist	MAN

Name	Information-Driven Networking based on type of data
Description	The Information-Driven Networking mechanisms enforce a set of policies by specifying the rules of how two or more components can communicate (send/receive data) with each other according to the available resources.
Additional Information	A different policy is enforced based on different incoming data requirements, following the type of processing requirements (stream, micro-batch, batch) and the type of data (structured, semi-structured, unstructured).

Table 51 - Requirement (1) for Information-Driven Networking

	Id	**Level of detail**	**Type**	**Actor**	**Priority**
	REQ-IDN-02	Software	FUNC	Data Scientist	MAN
Name	Information-Driven Networking based on application requirements				
Description	The Information-Driven Networking mechanisms enforce a set of policies by specifying the rules of how to handle applications with different requirements according to the available resources. For instance, an application with analytics requiring real-time data processing may impose time-critical constraints on the handling, operation and transformation of data. To support online analytics and decision making in time-critical conditions specific network policies need to be applied to deliver the results within predefined time constraints.				
Additional Information	The Data Scientist can set an “allow/deny access” policy regarding the set of applications and their requirements (real-time, close to real-time needs) accessing the backend services of the BigDataStack environment to prioritize/isolate the set of ingress/egress workloads that are enabled/dis-based on their IP & Port in order to achieve efficient services interaction.				

Table 52 - Requirement (2) for Information-Driven Networking

9.2. Design Specifications

Through the Information-Driven Networking component the Data Scientist declares her intend to be realized by the underlying system to translate either the data or the application or the security requirements into specific networking primitives that achieve the desired Service-Level Objective (SLO). This objective may refer to various kinds of traffic – streams, batches and micro batches – get the isolation/priority of availability and bandwidth that are needed to serve the network users effectively. With the convergence of all data and services in the same network, the Information-Driven Networking will manage traffic according to the network utilisation, the applications requirements and the communication latency without compromising the functionality of the network. Using policy statements, either the Network Administrators or the Data Scientists can specify which kinds of service / pod need to be given priority, at what times and on what part of their communication protocol (TCP, HTTP, etc.).

As all the mandatory building blocks of BigDataStack are containerized, a pod representing the basic building block in Kubernetes, encapsulates an application container (or multiple containers). Therefore, a set of labels and selectors need to be defined to assign key/value pairs to pods and set up the expressions that combine these labels in order to identify the traffic from/to individual containers, virtual machines and hosts that it needs to handle before it is routed/delivered to its destination. Then, the network policy definition includes a pod selector and the rules that apply to all the pods that meet the selector criteria. These rules are applicable to egress and ingress resources establishing connections to the pods, refer to labels with specific IPs or IP ranges and can permit or restrict communication to specific ports or allow/deny access to/from specific namespaces. For instance, there may be various namespaces serving different needs such as client and UIs services/applications. To configure network policies enforcement, specific services (frontend, backend) need to be exposed to specific namespaces (client, UIs). In the following, we present an example of controlling ingress traffic by giving an indicative network policy definition.

```
kubectl create -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-nginx
  namespace: sample-policy-demo
spec:
  podSelector:
    matchLabels:
      run: nginx
  ingress:
    - from:
      - podSelector:
          matchLabels: {}
EOF
```

Figure 15 – An indicative network policy definition for ingress traffic.

To address the challenges of a specific application, its requirements and the respective policies enforcement, a set of mechanisms operating at the services layer are expected to set up the appropriate attributes to understand the virtual hosts, URLs and other HTTP headers. This functionality implements the policy enforcement endpoint inside the pod as sidecar container in the same network namespace. This approach is highly flexible and HTTP aware and facilitates to apply policies in support of **operational goals**, such as service routing, retries, circuit-breaking, etc.

Containers networking is realised by *Networking as a Service* (through Neutron in OpenStack) and easily deployed containers (through Magnum either as Virtual Machines or Physical Machines). The idea is to bridge networking functionalities supported by Neutron

for Containers Use Cases using abstraction mechanisms (exploiting functionalities of Istio[20] and/or Kuryr[21], presented in section 5). The outcome is to deliver Neutron networking and services to Docker containerised services.

The Information-Driven Networking mechanisms also operate at the network layer. The latter gives the advantage of being universal. Our focus is to address the challenges arising from the diverse **data types** (i.e., stream, micro-batch, batch) to enforce policies to DNS, storage services (i.e., scalable storage of LeanXscale, Object Store, etc.), real-time streaming, and a plethora of other services that do not use HTTP. This functionality implements the policy at the host node outside the network namespace of the guest pods. The workloads in the BigDataStack environment can communicate without IP encapsulation or network address translation for bare metal performance, which enables easier troubleshooting, and better interoperability. In settings that require an overlay, the Information-Driven Networking mechanisms will work with tunnelling. This approach is universal, highly efficient, and isolated from the pods and facilitates to apply policies in support of **security and data privacy goals**. In the following, we present an example of controlling communications to HTTP GET requests by giving an indicative network policy definition which consists of three policy objects.

```
# Restricting customer's communications to HTTP GET requests.
kind: SampleNetworkPolicy
metadata:
  name: customer_app
spec:
  selector: app == 'customer_app'
  ingress:
    - action: Allow
      http:
        methods: ["GET"]
  egress:
    - action: Allow
---------------------------------------------------------------------------------
# The customer_app is the consumer of this service.
# Restricting incoming connections to customer_app.
kind: SampleNetworkPolicy
metadata:
  name: summary
spec:
  selector: app == 'summary'
```

[20] Istio. https://istio.io/
[21] Kuryr. https://wiki.openstack.org/wiki/Kuryr

```
  ingress:
    - action: Allow
      source:
        serviceAccounts:
          names: ["customer_app"]
  egress:
    - action: Allow
---------------------------------------------------------------------------------
# Restricting access to LXS.
# Only the summary microservice has direct access to LXS database.
kind: SampleNetworkPolicy
metadata:
  name: LXS_db
spec:
  selector: app == 'LXS_db'
  ingress:
    - action: Allow
      source:
        serviceAccounts:
          names: ["summary"]
  egress:
    - action: Allow
```

Figure 16 – An indicative network policy definition for controlling HTTP GET requests.

In the following figure, we present the high-level functionalities of the Information-Driven Networking tool in a UML Components diagram.

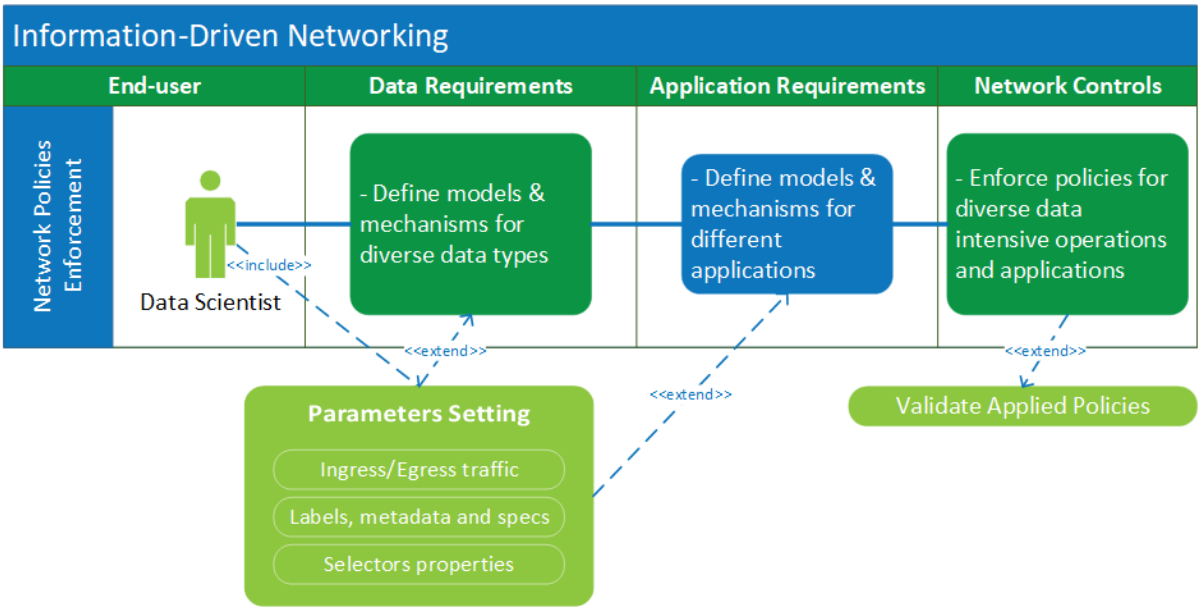


Figure 17 – Information-Driven Networking UML.

9.3. Experimentation Outcomes

The **Data Scientist** uses the **Information-Driven Networking (IDN)** tool, to define metadata and means of communication to apply tailored controls to data intensive operations and applications related with data intensive tasks according to specific requirements, by also including:

- The identification of the end-to-end application objectives in terms of specifying KPIs and criteria for optimal networking management and engineering (i.e. throughput, packet loss, jitter);
- The definition of the constraints arising from the type of data to be processed (data transfer, liveness, readiness among services) and the requirements of the application (time criticality, security, privacy);
- The validation of the applied network controls by evaluating that the policies have been correctly enforced and that resources are distributed among consumer services/applications as requested.

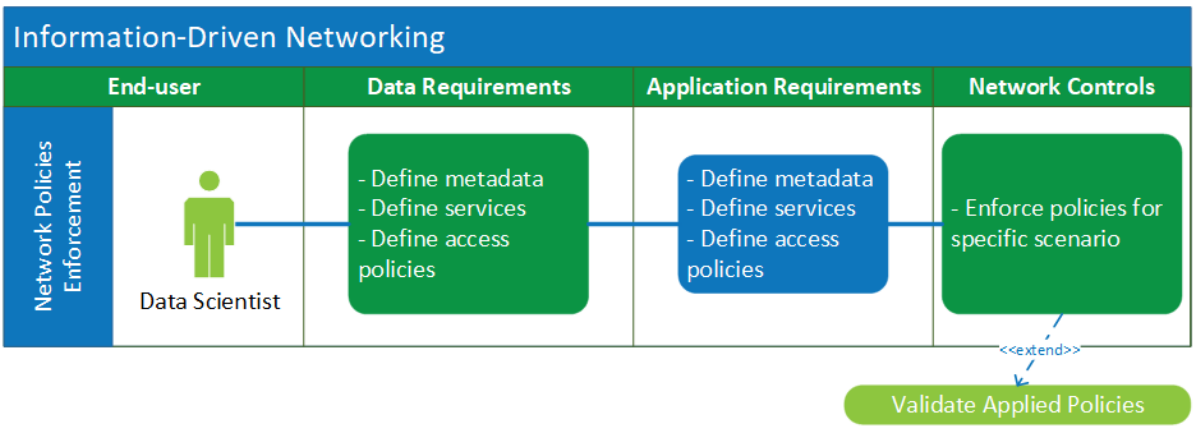


Figure 18 – Mapping of Information-Driven Networking tool with BDS Use Cases.

The IDN plays a supportive role to the components bringing the intelligence to the Data-Driven infrastructure Management: the ADS Ranking & Deploy and the Dynamic Orchestrator (DO). Furthermore, due to the late start of the development of this component (according with the original plan), the IDN has not been engaged yet in the experimentation carried out for those two components.

9.4. Implementation and Integration Highlights

The Information-Driven Networking component combines the Kubernetes Network Policies[22] to handle Ingress and Egress traffic in the cloud infrastructure at the Network and the Transport Level with the Istio[23] open source service mesh that transparently layers the services / pods at the Service Layer according to the OSI Model. The service mesh is used to describe the network of containerized microservices that interact in the BigDataStack environment. As the project progresses and the service mesh grows in size and complexity, it requires efficiency in service discovery, load balancing, failure recovery, metrics, and monitoring.

In this direction, we deploy special sidecar proxies throughout the BigDataStack environment which intercept all network communication between microservices. The key capabilities include: i) the efficient traffic management, incorporating the rules configuration and traffic routing which controls the traffic flows and API calls between services / pods; and ii) the secure traffic management, providing the underlying secure communication channel between services, which manages authentication, authorization, and encryption.

9.5. Next Steps

In the proceeding time period, we will work in the direction to enrich the Information-Driven Networking mechanisms and their experimentation with variable scenarios by enforcing different policies in diverse application requirements and for multiple constraints imposed by the data types, the time flexibility and the security constraints.

To achieve this, we will define a set of scenarios including simple cases (i.e. dropping all the traffic) and complex cases (i.e. limiting traffic to an application).

The simple case includes to isolate traffic to a service from other pods. This includes pods coming by the BigDataStack environment and external services as depicted in Figure 19.

[22] https://kubernetes.io/docs/concepts/services-networking/network-policies/
[23] https://istio.io

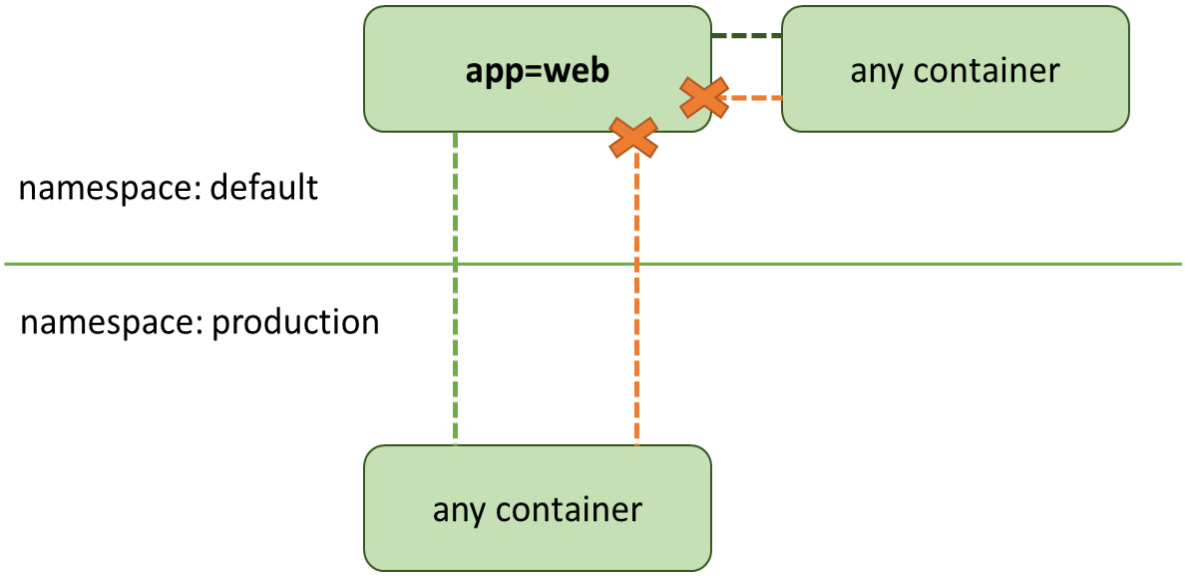


Figure 19. Traffic Isolation from internal and external pods.

The complex case includes the traffic restriction to a service by allowing connections only through microservices that need to use it, as depicted in Figure 20.

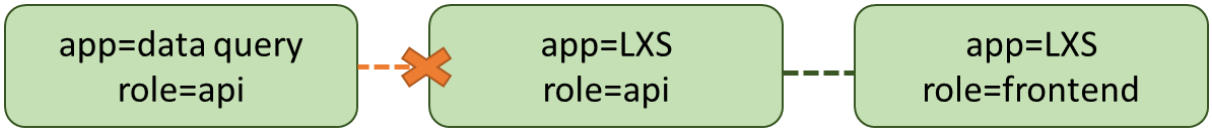


Figure 20 – Grant access to services that use the pod.

Additionally, the use case of DANAOS will serve as a ground truth scenario to assess the enforcement of time constrained policies as it has a set of real-time data processing requirements. At the same time, the use case of Connected Consumer of ATOS WRL and personalised Insurance Services of GFT will serve as a ground truth scenario to assess the enforcement of security policies which preserve the privacy of the end user.

10. Global Experimentation Outcomes

To support the development of the Data-Driven Infrastructure Management (DDIM) services within BigDataStack, it is important to understand the impact of different configurable parameters (such as CPU, Memory or application-specific factors) inherent to the applications that the platform may need to deploy. This is because DDIM services such as ADS-Ranking and the Dynamic Orchestrator need to make decisions about how to best set those configurable parameters, such that QoS (Quality of Service) objectives are met, and at minimal cost. In an ideal world, there would be enough time to perform a detailed analysis of each possible application type that might be deployed, quantifying the impact of each configurable. However, the universe of applications that can be deployed is so large that this is not a realistic goal. Indeed, *DockerHub*, the largest public repository of containerized applications currently holds over 2.8 million images. Instead, we opt to perform a deep-dive analysis of the most complex of the BigDataStack use cases from a deployment perspective (Connected Consumer), with the aim of identifying the main configurable parameters that impact QoS, as well as broader lessons. By doing so, the observations that we make can then be used to enhance the design and implementation of the DDIM services themselves. Additionally, through our analysis, new ways to improve QoS for the use-case may also be identified.

10.1. QoS Analysis in CC Use-Case

The Connected Consumer (CC) use case is a particularly suitable candidate for analysis, as it spans both intensive batch-style computation (during model learning), in addition to real-time streaming computation (serving recommendations to users). It also needs to tackle challenges with variable data rates for the real-time setting, as request volume will vary over time and can be impacted by external events (e.g. flash sales). As such, this use-case utilises all the DDIM services, including ADS-Ranking, ADS-Deployment, Triple Monitoring, the Dynamic Orchestrator, and more. Furthermore, as the aim of this use-case is to produce state-of-the-art supervised models (including GPU-based deep learned models), it is representative of a wide range of current applications that BigDataStack may need to deploy and manage.

The following sections are dedicated to our deep dive analysis of this use-case, including experiments examining how different deployments perform on cluster infrastructures in terms of QoS. Moreover, as a result of this analysis, we also identified an avenue to enhance QoS for this use-case over the state-of-the-art, which we also describe and report performance results for here.[24] We structure this section as follows. In Section 10.2, we provide a short literature review into product recommendation and more specifically, grocery product recommendation. Section 10.3 describes our methodology for evaluating QoS for the use-case, in terms of datasets, metrics and data pre-processing performed. In Section 10.4 we analyse the QoS of different grocery recommendation algorithms across three datasets, representing the current-state-of-the-art. Section 10.5 describes the weakness of these models and introduces a new model that we propose called VBCAR to tackle these deficiencies. In Section 10.6 we compare the performance of VBCAR against the

[24] We also recently published this enhancement, and as such we include the full research paper in Appendix A.

state-of-the-art and discuss the trade-offs in terms of QoS that it brings. Finally, in Section 10.7 we provide an investigation into key configurables of state-of-the-art grocery recommendation models that impact QoS, namely: embedding size, hidden layer size and the data sample size.

10.2. Current Product Recommendation Systems

Grocery recommender systems are increasingly used by supermarkets and online shopping platforms to assist their users in selecting and choosing products to purchase, as is the case for the EROSKI use-case. The most significant difference between the grocery recommendation task and other recommendation tasks, such as video recommendation [16] and movie rating prediction [17], is that the basket contextual information is both very common and important in grocery shopping scenarios. Most of the existing commonly used techniques for recommender systems, for example, the matrix factorization (MF)-based methods and the neural network (NN)-based methods, can be directly applied to solve the grocery recommendation task.

A number of MF-based recommender systems have been proposed, which mainly focused on modelling interactions between users and items as explicit feedback about those items (i.e. rating scores) from users [16,17,18]. For example, a basic MF algorithm [17] encodes the user-item explicit feedback as a rating matrix and predicts the rating scores of unseen items for users by completing the matrix. Many sophisticated matrix factorization techniques, such as Time Singular Value Decomposition [18], implicit Factorization Machines (FM) [19] and context-aware FM [20], have been proposed to address both classical item recommendation as well as more advanced scenarios, such as time-aware [18], implicit feedback [19] and context-aware recommendation [20]. In the scenario of grocery recommendation with implicit feedback, it is common practice to assign positive rating values to those user-item pairs that have purchase records and negative rating values for those that have zero purchases on record. These rating values can then be used to train different recommendation models, including the aforementioned MF-based models. Furthermore, the MF method that uses Bayesian personalized ranking optimisation (MF-BPR) loss [16] has been shown to be effective for grocery recommendation. However, this and similar matrix completion-based methods [16,17,21] have a significant limitation, namely that they are unable to incorporate basket information (i.e. information about sets of items bought together), limiting their effectiveness.

Recently, some prior works have started to apply neural networks to learn latent factors between users and items, with the aim of better capturing non-linear relationships among these interactions. For instance, the Neural Collaborative Filtering (NeuCF) [21] model is a general framework that integrates deep learning into matrix factorization approaches using implicit feedback. The attention mechanism was introduced by Chen et. al [22] to integrate item- and component-level implicit feedback in multimedia recommendation. Additionally, a lot of recurrent neural network (RNN) models have been applied to recommender systems to better capture contextual information. For example, Manotumruksa et al. [23] applied two gating RNNs, i.e. a Contextual Attention Gate (CAG) and Time- and Spatial-based Gates (TSG), incorporating both time and geographical information for (venue) recommendation. Meanwhile, Li et al. proposed a collaborative variational auto-encoder [24] that learns deep latent representations from content data in an unsupervised manner, while also learning

implicit relationships between items and users from both content and ratings. More recently, the Graph Neural Networks (GNNs) [25,26] that jointly learn network embeddings integrating node information and topological structure from graph data, have been applied into recommender systems to jointly model the user-user social graph and the user-item graph for item recommendation [27 ,28].

Considering specifically the grocery product recommendation domain, state-of-the-art neural network-based approaches have been recently proposed to learn latent representations that incorporate the basket information to enhance the performance of grocery recommendation [29,30,31,32]. Most notably, Triple2vec [26,30] is an effective model that learns latent representations capturing the basket context by maximizing the likelihood of reconstructing sampled triples, where each triple is constructed by the previously observed two items co-occurring within the same basket of one user. This approach is currently the state-of-the-art in grocery recommendation approaches and is the natural choice for deployment for the EROSKI use-case. As such, we use Triple2vec in addition to other more traditional MF approaches in this study.

10.3. Experimental Methodology

In this section we summarize our experimental setup for the grocery recommendation use-case, in terms of datasets, metrics, baselines and any pre-processing techniques that we apply.

Baselines: Among the existing product recommendation methods discussed above, we choose three state-of-the-art methods, namely the MF-BPR [17], NeuCF [21] and Triple2vec [30], for evaluation.

Datasets: We conduct experiments on three real-world grocery transaction datasets, namely Instacart[25], Tafeng and Dunnhumby[26], which are public benchmark datasets in the research community of grocery recommender systems. Due to the very large number of interactions within the Instacart dataset, we use a 10% random sample in terms of users and items to reduce training time. The statistics of these datasets are shown in Figure 21. Of the three datasets, two contain item side information. In particular, the Instacart dataset contains three types of item side information, where two types (‘aisle_id' and ‘department_id') are categorical data and (‘product name') is textual data. Meanwhile, the Dunnhumby dataset contains four types of item side information, where two types (Manufacturer and Department) are regarded as categorical data and two types (Commodity Description and Sub Commodity Description) are textual data. The Tafeng dataset does not have side information.

[25] http://www.instacart.com/datasets/grocery-shopping-2017
[26] http://www.dunnhumby.com/careers/engineering/sourcefiles

Percentage	#Users	#Items	#Orders	#Interactions
Tafeng	9,238	7,973	77,202	464,118
Dunnhumby	2,497	61,600	113, 831	1,048,575
Instacart (10%)	11,868	3,144	132,397	302,244
Instacart (100%)	119,098	32,243	2,741,332	29,598,689

Figure 21 – Grocery recommendation datasets.

Pre-processing: All three datasets are subject to filtering to remove rare items and users since they will not have enough associated interactions, in a similar way to previous works [26,33]. In particular, any user that has purchased less than 30 items and/or has less than 10 baskets, is filtered out. Furthermore, any item that was purchased less than 20 times is removed. For model evaluation, we split all the baskets for each of the three datasets into training (80%) and test (20%) sub-sets based on time order. Depending on the model type, we then may further split or sample the (80%) training sub-set.

Metrics: We treat product recommendation as an item ranking task (for each user). We report two main metrics, namely NDCG@k and Recall@k, which are all standard ranking evaluation metrics and widely used in recommender systems [16,17,18], as the metrics for evaluating the effectiveness. We also evaluate the efficiency of each model based on the running time, the consumption of memory, CPU and GPU. Recommendation algorithm deployment is performed on a local cluster powered by GeForce RTX 2080Ti GPUs. To provide cost estimates for each algorithm per dataset we calculate the cost for equivalent hardware in commercial clouds. Indeed, based on current cloud pricing (Amazon Web Services, EC2), we estimate that the cost for an equivalent machine in the cloud would be around $0.75 per hour.

10.4. Product Recommendation QoS

We initially assess the QoS (effectiveness and efficiency) for state-of-the-art recommendation methods (i.e. MF+BPR, NeuCF and Triple2vec) from the literature. Figure 22 shows the results of this comparison in terms of the next basket recommendation task on the Dunnhumby, Tafeng and Instacart (10%) datasets. Note that we exclude some baselines for the larger Instacart dataset due to excessive training time. We observe that the Triple2vec model shows consistently better performance on NDCG@5 and Recall@5 metrics in all our three datasets, compared with the MF+BPR and NeuCF models. In terms of the efficiency, MF+BPR needs less running time for obtaining the recommendation result in both the Dunnhumby and Tafeng datasets, while Triple2vec needs less running time in the Instacart dataset. By comparing the prices and the recommendation performance, we suggest that Triple2vec is the best algorithm in handling the grocery recommendation task since no other baselines can outperform it in both effectiveness and efficiency.

	Method	Effectiveness		Efficiency		
		NDCG@5	Recall@5	Time	Memory	Price
Dunnh	**MF+BPR**	0.5790	0.0728	**1052s**	**2.710GB**	**$0.75**
	NeuCF	0.5052	0.0571	27683s	6.076GB	$6
	Triple2vec	**0.6195**	**0.0874**	5349s	6.973GB	$1.5
Tafeng	**MF+BPR**	0.3548	0.1587	**284s**	2.535GB	**$0.75**
	NeuCF	0.2692	0.1121	15410s	6.076GB	$3.75
	Triple2vec	**0.4489**	**0.2056**	521s	**0.823GB**	**$0.75**
Instacart	**MF+BPR**	0.5438	0.5622	1733s	3.381GB	**$0.75**
	NeuCF	0.3083	0.3113	88455s	6.076GB	$18.75
	Triple2vec	**0.6536**	**0.6581**	**1262s**	**1.498GB**	**$0.75**

Figure 22 – Grocery recommendation QoS for current state-of-the-art algorithms.

10.5. Enhancing QoS (Proposing VBCAR)

The previous experiment provides us a strong conclusion on what recommendation algorithm should be deployed for this use-case, i.e. Triple2vec. However, that algorithm has a notable limitation, it is not able to capture side information (e.g. product categories or descriptions), which are likely to be useful when recommending groceries. Hence, in order to enhance QoS for grocery recommendation, we make modifications on the Triple2vec model in terms of two aspects to improve performance. First, we extend the Triple2vec model to the Bayesian situation and make the model to be able to learn distributional representations. Second, we integrate the model with item side information to better capture the item similarities among the learned embeddings.

In particular, given a grocery purchase history represented as:

$$\mathcal{S} = \{\langle u, i, o\rangle \mid u \in \mathcal{U}, i \in \mathcal{I}, o \in \mathcal{O}\}$$

with

$$\mathcal{O} = \{o_1, o_2, \cdots, o_L\}, \mathcal{U} = \{u_1, u_2, \cdots, u_N\}\ \mathcal{I} = \{i_1, i_2, \cdots, i_M\}$$

being the sets of baskets, users and items respectively, Triple2vec and similar models first sample a large number of triples

$$\mathcal{T} = \{\langle u, i, j\rangle \mid \langle u, i, o\rangle \in \mathcal{S}, \langle u, j, o\rangle \in \mathcal{S}\}$$

reflecting two items purchased by the same user in the same basket from historical grocery shopping baskets. Triple2vec aims to learn the latent embedding for users and items to predict the occurrence probability of these triples. Following the basic idea of word2vec [34], one can treat these sampled triples as context windows and learn latent embeddings by optimizing the likelihood of the triple samples:

$$\mathcal{L}_{Triple2vec} = \sum_{(i,j,u)\in\mathcal{T}} (\log P(i|j,u) + \log P(j|i,u) + \log P(u|i,j)),$$

which leads to the training objective of Triple2vec [30]. Here *P(i | j, u)*, *P(j | i, u)* and *P(u | i, j)* are the *softmax* formulations predicting the occurrence probability of a context entity from the embeddings of two target entities, e.g.

$$P(i \mid j,u) = \frac{\exp\left(\mathbf{z}_i^{iT}(\mathbf{z}_j^i+\mathbf{z}_u^u)\right)}{\sum_{i'}\exp\left(\mathbf{z}_{i'}^{iT}(\mathbf{z}_j^i+\mathbf{z}_u^u)\right)},$$

where

$$\mathbf{z}_u^u \in \mathbf{Z}^u \text{ and } \mathbf{z}_i^i, \mathbf{z}_j^i \in \mathbf{Z}^i$$

are the latent representations of user *u* and items *i* and *j*. This skip-gram-based loss objective is commonly trained with the negative sampling trick that uses the Noise Contrastive Estimation (NCE) to approximate the *softmax* function [30].

We propose a new model, Variational Bayesian Context-Aware Representation (VBCAR) that extends Triple2vec by assuming that both $\mathbf{Z}^u$ and $\mathbf{Z}^i$ are random variables. Instead of optimizing the likelihood, VBCAR introduces a variational evidence lower bound of the triple samples and maximizes this lower bound by the amortized inference [35,36]:

$$\mathcal{L}_{VBCAR} = \mathbb{E}_{q_\phi(\mathbf{Z}^u,\mathbf{Z}^i)}\left[\mathcal{L}_{Triple2vec}\right] - KL\left(q_\phi\left(\mathbf{Z}^u,\mathbf{Z}^i\right) \| p(\mathbf{Z}^u,\mathbf{Z}^i)\right),$$

where KL is the Kullback-Leibler divergence and $q_\phi(\mathbf{Z}^u,\mathbf{Z}^i)$ is the variational distribution of the embedding, which can be factorized as Gaussian distributions of the mean-field form. These two variational distributions are independent and can be inferred by two different encoder networks with the item key representations of users and items as input respectively [26]:

$$q_{\phi_1}\left(\mathbf{Z}^u \mid \mathbf{F}^u\right) = \mathcal{N}\left(\boldsymbol{\mu}^u, \boldsymbol{\sigma}^{u2}\mathbf{I}\right), q_{\phi_2}\left(\mathbf{Z}^i \mid \mathbf{F}^i\right) = \mathcal{N}\left(\boldsymbol{\mu}^i, \boldsymbol{\sigma}^{i2}\mathbf{I}\right),$$

where $\boldsymbol{\mu}^u$, $\boldsymbol{\sigma}^{u2}$, $\boldsymbol{\mu}^i$ and $\boldsymbol{\sigma}^{i2}$ are parameters of their embedding distributions inferred by the fully connected layers (FC). Since encoding items and users using a one-hot identity representation is computationally expensive for large datasets, our model uses randomly generated keys[27] for items and users. For ease of reference, we denote these key-based representations of each item and user as the Item Key and User Key representations, respectively. Figure 23 (a) provides an overview of our VBCAR model.

[27] A key is a random initialization from the standard normal distribution to represent the input feature.

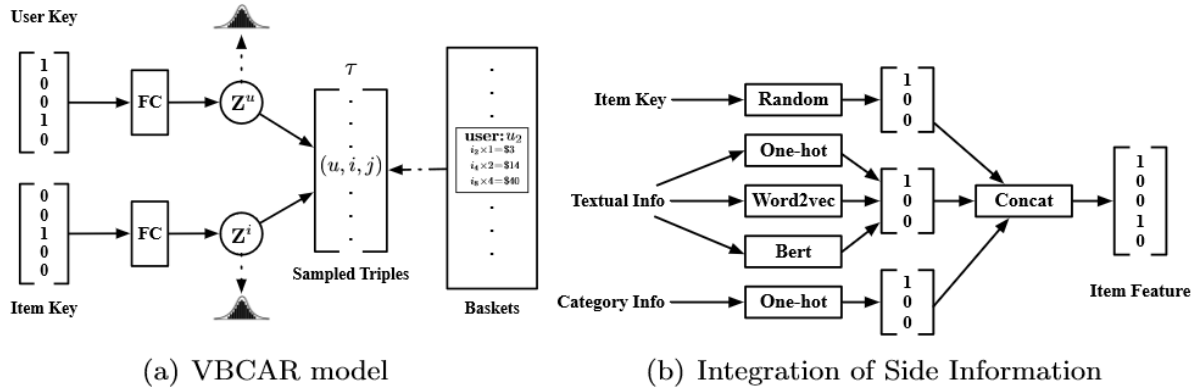


Figure 23 – Representation Learning for VBCAR with Item Side information.

Besides the Item Key representation of items, we also adopt three feature extraction methods for pre-processing the item description/name into vectoral representations. Figure 23 (b) shows the integration process of the side information. The three feature extraction techniques are summarized below:

- **One-hot (One):** Since encoding the full item description into one-hot category representations would be prohibitively expensive due to the large vocabulary size, we instead resort to encoding only the high frequency words from the training triples into one-hot representations.
- **Word2vec (W2c) [34]:** Word2vec is one of the most popular methods to learn word embeddings using neural networks. We use the Google pre-trained word2vec model[28] to obtain embeddings for all the words appearing in the item descriptions. We use the mean vector of these word embeddings to construct the product description embedding.
- **BERT** [37]: We also use the Bidirectional Encoder Representations from Transformers (BERT), a popular pre-trained language model for encoding words and sentences into embeddings. We use the DistilBert[29] from HuggingFace, which is a smaller and faster architecture based on BERT.

In our later experiments, we also combine (concatenate) different item representations together to evaluate the impact each has on product recommendation performance.

10.6. VBCAR and VBCAR-S Performance

We assess the QoS provided by VBCAR and our VBCAR-S model by comparing with Triple2vec. Figure 24 shows the results of this comparison in terms of the next basket recommendation task on our datasets.

[28] http://code.google.com/archive/p/word2vec/
[29] http://github.com/huggingface/pytorch-transformers

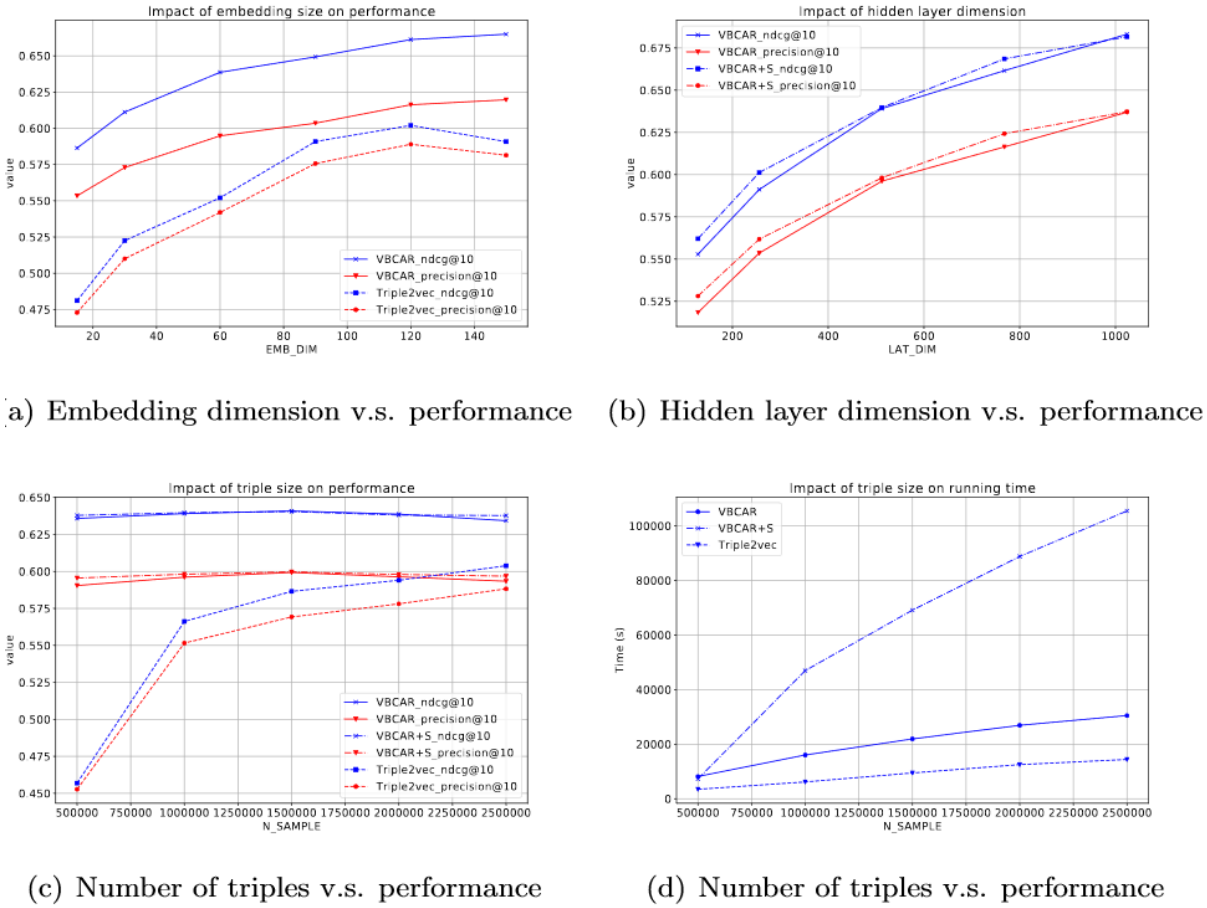
	Method	Effectiveness		Efficiency		
		NDCG@5	Recall@5	Time	Memory	Price
Dunnh	**Triple2vec**	0.6195	0.0874	5349s	6.973GB	$1.5
	VBCAR	0.7405	0.1446	15770s	**3.499GB**	$3.75
	VBCAR+S	**0.7576**†	**0.1544**†	47106s	3.535GB	$10.5
Instacarflafeng	**Triple2vec**	0.4489	**0.2056**	521s	0.823GB	**$0.75**
	VBCAR	**0.4643**†	0.1940	41525s	3.224GB	$10.5
	Triple2vec	0.6536	0.6581	1262s	1.498GB	$0.75
	VBCAR	0.7405	0.6796	49852s	4.584GB	$10.5
	VBCAR+S	**0.7576**†	**0.6959**†	66533s	4.619GB	$14.25

Figure 24 – Quality of our new VBCAR model in comparison to Triple2vec.

We can observe from Figure 24 that both the VBCAR and VBCAR-S models show better performances than Triple2vec in almost all the metrics and all the datasets, which validates the effectiveness of these models. Furthermore, our VBCAR-S model when combining the item keys with encoded product name/commodity descriptions obtained the best performance over all evaluation metrics and datasets. Moreover, the performance improvements observed for VBCAR-S over the baseline methods are statistically significant (paired t-test $p < 0.05$) in both datasets. This shows that incorporating item side information can enhance the performance of state-of-the-art grocery recommendation. However, it is notable that these gains in recommendation performance comes at a cost. The time needed to train the VBCAR models is over 10x-15x that of Triple2vec, due to the need to extract and train based on the additional features. Hence, if time or cost are critical considerations for the application owner, it may be advisable to still use the less effective Triple2vec models.

10.7. Analysis of Use-Case-Specific Factors

Finally, we analyze the impact of key configurables of Triple2vec and VBCAR, namely the embedding size, the hidden layer size and the number of sampled triples on both the effectiveness and efficiency. Figure 25 summarizes the experimental results.



(a) Embedding dimension v.s. performance (b) Hidden layer dimension v.s. performance

(c) Number of triples v.s. performance (d) Number of triples v.s. performance

Figure 25 – Analysis of grocery recommendation configurables.

i. **The Effect of Embedding Size:** We report the performance of both the VBCAR and Triple2vec models in Figure 25 (a) by varying the embedding size. We can observe that increasing the embedding size substantially enhances the performance of recommendation (in terms of NDCG@10 and Precision@10) for both models. Indeed, the highest effectiveness observed for VBCAR used an embedding size of 150, while the highest effectiveness of Triple2vec was observed when using an embedding size is 120.

ii. **The Effect of Hidden Layer Size:** As both VBCAR and VBCAR+S models infer embeddings of nodes and attributes by MLP with the fixed dimensional hidden layers, we analyse the performance of both the VBCAR and VBCAR+S models in Figure 25 (b) by testing different values for hidden layer size. We can observe that increasing the hidden layer size substantially enhances the performance of recommendation (in terms of NDCG@10 and Precision@10) for both the VBCAR model and VBCAR+S model.

iii. **The Effect of Triple Sample Size:** Figure 25 (c-d) reports the effect of the number of triples sampled for training on the recommendation performance and running time. As the number of triples increase from 0.5 million to 2.5 million, we can clearly see that the performance of Triple2vec improves, but the changes of the NDCG@10 and Precision@10 performance in both the VBCAR and VBCAR+S model are not nearly as large. In particular, both the VBCAR and VBCAR+S model trained with only 0.5 million

sampled triples can outperform Triple2vec trained with 2.5 million sampled triples, which means our VBCAR model with limited sample size as input can learn more expressive representations, result in better improvement compared with Triple2vec. In Figure 25 (d), we can observe that the running time of all the three models increases as the number of triples increases, Triple2vec is the most efficient model under all the triple size settings, while in contrast the VBCAR+S model is the most expensive to train. This result can be explained by the fact that VBCAR has more parameters (hidden layers) needed to be learned than Triple2vec. Meanwhile, VBCAR+S also has additional input features derived from the item side information which need to be encoded.

In general, we can see from these results that each of these three configurables, which are particular to this type of recommendation algorithm can have a large and significant impact on grocery recommendation QoS.

10.8. Study Summary and Lessons Learned

In this section we have analysed the EROSKI grocery recommendation use-case, with the aim of identifying the main configurables that impact QoS, as well as broader lessons. We first analysed the literature to find the current state-of-the-art approaches in grocery recommendation and evaluated those models over three datasets to determine the QoS that they provide and at what cost. From these results, we conclude that the Triple2vec model is both the most effective, and the most efficient for use on large datasets, and hence is the recommended choice for deployment for this use-case.

However, we also noted that this model has a significant limitation, in that it is not able to encode side information, such as product category or descriptions. As such, we also proposed two new models, namely VBCAR and VBCAR+S, which aim at tackling this issue. Through analysis of this new model, we show that significantly improved recommendation accuracy is possible, although this comes at a notable additional cost in training time and hence monetary cost when using cloud infrastructures.

Finally, we analysed the main configurables that are specific to Triple2vec and VBCAR(+S). Through this analysis, we demonstrated that the configurables Embedding size, Hidden Layer Size and Triple Sample Size have a large and significant impact on grocery recommendation QoS. Considering this outcome from the perspective of the DDIM services, this shows that simply considering generic configurables such as CPU and memory allocation for an application will not be enough to predict application QoS. Hence, we recommend that models such as those used by ADS-Ranking and the Dynamic Orchestrator should consider application-specific configurables as well during their operation where possible.

11. References

[1] Network Policies in Kubernetes. Available Online: https://kubernetes.io/docs/concepts/services-networking/network-policies/

[2] Project Calico. Available Online: https://www.projectcalico.org/

[3] Istio. Available Online: https://istio.io/

[4] de Vaulx, Frederic J., Eric D. Simmon, and Robert B. Bohn (2018). "Cloud computing service metrics description." Special Publication (NIST SP)-500-307. 2018.

[5] William Voorsluys, James Broberg, Srikumar Venugopal, Rajkumar Buyya, Martin Gilje Jaatun, Gansen Zhao, Chunming Rong (2009). "Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation", Cloud Computing, Springer Berlin Heidelberg, 2009, P 254-265

[6] D. Guyon, A. Orgerie, C. Morin and D. Agarwal (2017). "How Much Energy Can Green HPC Cloud Users Save?" in 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), St. Petersburg, 2017, pp. 416-420.

[7] Gulisano, V., Jimenez-Peris, R., Patino-Martinez, M., Soriente, C., & Valduriez, P. (2012). "Streamcloud: An elastic and scalable data streaming system." IEEE Transactions on Parallel and Distributed Systems, pp. 2351-2365.

[8] H. Rui et al. (2014). "Enabling cost-aware and adaptive elasticity of multi-tier cloud applications." Future Generation Computer Systems, pp. 82-98.

[9] Kalervo and Jaana. (2002). "Cumulated gain-based evaluation of IR techniques." ACM Transactions on Information Systems (TOIS), pp. 422--446.

[10] L. Tie-Yan. (2009). "Learning to rank for information retrieval." Foundations and Trends in Information Retrieval, pp. 225-331.

[11] M. Ferdman et al. (2012). "Clearing the clouds: a study of emerging scale-out workloads on modern hardware." ACM SIGPLAN Notices, pp. 37-48. ACM.

[12] Raschke, R. (2010). "Process-based view of agility: The value contribution of IT and the effects on process outcomes." International Journal of Accounting Information Systems, 11(4), pp. 297-313.

[13] Salton and McGill. (1986). "Introduction to modern information retrieval." McGraw-Hill, Inc.

[14] Sergey and Christian. (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint.

[15] Z. Jia et al. (2013). "Characterizing data analysis workloads in data centers." IEEE International Symposium on Workload Characterization (IISWC), pp. 66-76. IEEE.

[16] Andriy Mnih and Ruslan R Salakhutdinov. Probabilistic matrix factorization. In Advances in Neural Information Processing Systems, pages 1257–1264, 2008.

[17] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In Proceedings of the 25th conference on uncertainty in artificial intelligence, pages 452–461, 2009.

[18] Yehuda Koren. Collaborative filtering with temporal dynamics. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 447–456, 2009.

[19] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In 2008 Eighth IEEE International Conference on Data Mining, pages 263–272, 2008.

[20] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. Fast context-aware recommendations with factorization machines. In Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval, pages 635–644, 2011.

[21] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In Proceedings of the 26th International Conference on World Wide Web, pages 173–182, 2017.

[22] Jingyuan Chen, Hanwang Zhang, Xiangnan He, Liqiang Nie, Wei Liu, and Tat-Seng Chua. Attentive collaborative filtering: Multimedia recommendation with item-and component-level attention. In Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval, pages 335–344, 2017.

[23] Jarana Manotumruksa, Craig Macdonald, and Iadh Ounis. A contextual attention recurrent architecture for context-aware venue recommendation. In Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, pages 555–564, 2018.

[24] Xiaopeng Li and James She. Collaborative variational autoencoder for recommender systems. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 305–314, 2017.

[25] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In International Conference on Learning Representations, 2017.

[26] Zaiqiao Meng, Shangsong Liang, Hongyan Bao, and Xiangliang Zhang. Co-embedding attributed networks. In Proceedings of the 12th ACM International Conference on Web Search and Data Mining, pages 393–401, 2019.

[27] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In The World Wide Web Conference, pages 417–426, 2019.

[28] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2019.

[29] Duc Trong Le, Hady W Lauw, and Yuan Fang. Basket-sensitive personalized item recommendation. In Proceedings of the 26th International Joint Conference on Artifical Intelligence, pages 2060–2066, 2017.

[30] Mengting Wan, Di Wang, Jie Liu, Paul Bennett, and Julian McAuley. Representing and recommending shopping baskets with complementarity compatibility and loyalty. In

Proceedings of the 27th ACM International Conference on Information and Knowledge Management, pages 1133–1142, 2018.

[31] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. E-commerce in your inbox: Product recommendations at scale. In Proceedings of the 21st ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 1809–1818, 2015.

[32] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. Variational bayesian context-aware representation for grocery recommendation. In Workshop on Context-Aware Recommender Systems, 2019.

[33] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In Proceedings of the 19th International Conference on World Wide Web, pages 811–820, 2010.

[34] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems, pages 3111–3119, 2013.

[35] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. Journal of the American Statistical Association, 112(518):859–877, 2017.

[36] Rui Shu, Hung H Bui, Shengjia Zhao, Mykel J Kochenderfer, and Stefano Ermon. Amortized inference regularization. In Advances in Neural Information Processing Systems, pages 4393–4402, 2018.

[37] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv:1810.04805, 2018.

[38] Mossalam, H., Assael, Y. M., Roijers, D. M., & Whiteson, S. (2016). Multi-objective deep reinforcement learning. *arXiv preprint arXiv:1610.02707*.

[39] Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016, November). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (pp. 50-56). ACM.

[40] Mao, H., Netravali, R., & Alizadeh, M. (2017, August). Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (pp. 197-210). ACM.

[41] Bu, X., Rao, J., & Xu, C. Z. (2009, June). A reinforcement learning approach to online web systems auto-configuration. In *2009 29th IEEE International Conference on Distributed Computing Systems* (pp. 2-11). IEEE.

[42] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

[43] Breakout-v0 – Openai gym. https://gym.openai.com/envs/Breakout-v0/

[44] Fadel Argerich, M., Cheng, B., & Fürst, J. (2019). Reinforcement Learning based Orchestration for Elastic Services. *arXiv preprint arXiv:1904.12676*.