Measuring Performance Quality Scenarios in Big Data Analytics Applications: A DevOps and Domain-Specific Model Approach

Camilo Castellanos
cc.castellanos87@uniandes.edu.co
Universidad de los Andes
Bogota, Colombia

Carlos A. Varela
cvarela@cs.rpi.edu
Rensselaer Polytechnic Institute
Troy, NY, USA

Dario Correal
dorreal@uniandes.edu.co
Universidad de los Andes
Bogota, Colombia

ABSTRACT

Big data analytics (BDA) applications use advanced analysis algorithms to extract valuable insights from large, fast, and heterogeneous data sources. These complex BDA applications require software design, development, and deployment strategies to deal with volume, velocity, and variety (3vs) while sustaining expected performance levels. BDA software complexity frequently leads to delayed deployments, longer development cycles and challenging performance monitoring. This paper proposes a DevOps and Domain Specific Model (DSM) approach to design, deploy, and monitor performance Quality Scenarios (QS) in BDA applications. This approach uses high-level abstractions to describe deployment strategies and QS enabling performance monitoring. Our experimentation compares the effort of development, deployment and QS monitoring of BDA applications with two use cases of near mid-air collisions (NMAC) detection. The use cases include different performance QS, processing models, and deployment strategies. Our results show shorter (re)deployment cycles and the fulfillment of latency and deadline QS for micro-batch and batch processing.

CCS CONCEPTS

• **Software and its engineering → Software architectures**; *Software performance*; • **Information systems** → *Data mining*; • **Computing methodologies** → *Distributed computing methodologies.*

KEYWORDS

Software architecture, big data analytics, performance quality scenarios, DevOps, domain specific model

ACM Reference Format:
Camilo Castellanos, Carlos A. Varela, and Dario Correal. 2019. Measuring Performance Quality Scenarios in Big Data Analytics Applications: A DevOps and Domain-Specific Model Approach. In *European Conference on Software Architecture (ECSA), September 9–13, 2019, Paris, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Big data analytics (BDA) applications use machine learning (ML) algorithms to extract valuable insights from large, (near) real-time and heterogeneous data. These BDA applications require complex software design, development, and deployment to deal with big data *3V* characteristics (volume, variety, and velocity) to maintain expected performance levels. But the complexity involved in applications development frequently leads to delayed deployments [6] and difficult performance monitoring (e.g., throughput or latency) [12]. Regarding big data 3V characteristics, a BDA solution can be constrained to different performance quality scenarios (QS). For instance, stream analytics applications require low latency, and flexible scalability based on data volume flow. On the other hand, batch processing of heavy workloads over large datasets demand high scalability and fault tolerance to achieve an expected deadline.

In the aviation safety domain, the collision avoidance systems enable aircraft to remain well clear using data collected by onboard and ground sensors. A well clear violation implies loss of separation between airplanes by calculating distance and time thus warning against Near Mid-Air Collisions (NMAC) [11]. The timely detection of NMACs within congested airspace (e.g., airport areas) using streaming and semi-structured sensor data requires data-intensive processing with strong latency constraints.

Within the field of software architecture, little research has been done to specify BDA functional and non-functional requirements using high-level abstractions to deploy, monitor and evolve BDA solutions constrained to performance QS. In this context, ACCORDANT [5] is a Domain-Specific Model approach which allows designing BDA applications using Functional and Deployment viewpoints and QS. A *Viewpoint* is a collection of patterns, templates, and conventions to express different concerns [13]. The QS specify quality attribute requirements for a software artifact to support its design, and quality assessment [3]. Though ACCORDANT metamodel includes a deployment viewpoint, containerization and performance QS monitoring have not been addressed.

This proposal aims to reduce the time of design, deployment, and performance monitoring of BDA applications applied in the avionics domain. We propose an extension of ACCORDANT[5] that includes performance QS and containerization approach to take advantage of portability, scalability, configuration and deployment. We design a domain-specific language (DSL) to describe architectural abstractions of functional, deployment, and QS. These abstractions allow us to generate functional and infrastructure code to measure the application's performance. Our experimentation monitor latency and deadline in two NMAC detection use cases which demand distributed batch and micro-batch processing over different deployment strategies. Our results report improvements in design and (re)deployment times to achieve the expected performance QS. In summary, the contributions of this paper are: i) A metamodel to specify BDA deployments over containers and QS. ii)

ECSA, September 9–13, 2019, Paris, France
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

A DSL to design deployment over containers and QS to accelerate BDA deployment monitoring. iii) An evaluation applied to avionics use cases with different deployment strategies and QS.

The rest of this paper is organized as follows. In Section 2, we present background. Section 3 reviews the related work. Section 4 presents our methodology and proposal overview. Section 5 presents the avionics use cases. Section 6 details the steps followed to validate this proposal. Section 7 reports and discusses the results. Finally, Section 8 summarizes the conclusions and future work.

2 BACKGROUND

2.1 Analytics Portability

Due to the complexity of deploying and operating BDA solutions integrating a myriad of technologies, complex analytics models and distributed infrastructure, some research has been done to tackle such complexity by raising the level of abstraction [5, 8–10]. Due to the wide range of BDA technologies, portability plays a key role to deploy, operate, and evolve BDA applications, and this is where portable standards appear. The Predictive Model Markup Language (PMML)[1] is the defacto standard proposed by the Data Mining Group that enables interoperability of analytics models through neutral-technology XML format. PMML allows specifying a set of ML algorithms and data transformations along with their metadata.

2.2 DevOps and Infrastructure as Code

According to Bass. et. al [4], DevOps is a set of practices aims to reduce the time for implementing from development to production environment, ensuring high quality. Infrastructure as Code (IaC) arises from the necessity to handle the infrastructure setup, evolution, and monitoring in an automated and replicable way through executable specifications. IaC promotes the reduction of cost, time and risk of IT infrastructure provision by offering languages and tools which allow to specify concrete environments (bare-metal servers, virtual machines, operative systems, middleware and configuration resources) and allocate them automatically. In this context, technologies such as Kubernetes[2] offers to decouple application containers from the infrastructure details to deploy, scale and manage container clusters.

2.3 Near Mid-Air Collisions Detection

Given the increasing demand, the airspace utilization density has been growing which reduces the separation between aircraft. This reduction increases the risk of collision, hence avionics' communications and surveillance systems are processing more data, and they have to maintain or improve performance QS in terms of accuracy, response time, and availability. NMAC detection requires sensing aircraft's positions and velocities to calculate distances and times to determine risk levels and maneuvers [11]. The Automatic Dependent Surveillance-Broadcast[3] (ADS-B) is the next generation air transportation technology which operates with satellite tracking rather than radar to monitor air traffic more accurately.

[1]http://dmg.org/pmml/v4-3/GeneralStructure.html
[2]https://kubernetes.io/
[3]https://www.faa.gov/nextgen/programs/adsb/

3 RELATED WORK

Artac et al. [2] propose a model-driven engineering (MDE) approach to create models of data-intensive applications which are automatically transformed into IaC. They use TOSCA and Chef, to support configuration management, service provisioning, and application deployment, but their experimentation does not include performance metrics monitoring of the deployed application. QualiMaster [1, 7] focuses on the processing of online data streams for real-time applications such as the risk analysis of financial markets regarding metrics of time behavior and resource utilization. The aim of QualiMaster is to maximize the throughput of a given processing pipeline. Similarly, our proposal generates software for BDA applications, but taking as input the analytics specification of a predictive model, and the performance metrics to be achieved. Unlike Qualimaster, our proposal is technology-neutral and cross-industry which enables a more widespread application.

Sandhu and Sood [14] propose a global architecture to schedule big data application in geographically distributed cloud data centers based on QoS parameters. These QoS parameters (response time, deadline, etc) along with application features (processing, memory, data input size, and I/O requirements) are given a priori by the users to recommend the appropriate data center and cluster for a specific BDA request. They use a Naïve Bayes classifier to determine the category' probabilities of a BDA request: compute intensive (C), input/output intensive (I), and memory intensive (M). In addition, a map with data centers and infrastructure resources is defined, specifying categories (CIM) to select the most suitable cluster and data center using a neural network model. Previous works analyze performance in already developed BDA software. However, our proposal includes the code generation of software and infrastructure of BDA solutions, and the performance monitoring for each component and connector.

4 A DEVOPS AND DSM APPROACH

Our proposal offers a high-level approach to the DevOps practice, starting from architectural artifacts, instead of source code. Specifically, we propose an extension of ACCORDANT metamodel [5] to deal with infrastructure setup and QS. The ACCORDANT's methodology, depicted in Figure 1, is composed of 7 steps: 1) The business user defines business goals and QS. 2) The data scientist develops analytics models and data transformations. The resulting analytics models are exported as PMML files. 3) Architect design the software architecture using ACCORDANT DSL in terms of *Functional Viewpoint*(FV) and *Deployment Viewpoint*(DV) embedding PMML models in FV to specify software behavior. 4) FV and DV models are interweaved to obtain an integrated model. 5) Generation of software and infrastructure code is done from integrated models. 6) The generated code is executed to provision infrastructure and install the software. 7) QS are monitored in operation.

To enable stakeholders to use the proposed metamodels, we design a Domain Specific Language (DSL) implemented with Xtext[4] framework. This DSL allows us to design both FV and DV models in a textual way. To illustrate how FV and DV models are specified using this DSL, code excerpts of the avionics use cases will be detailed in Section 6.3.

[4]https://www.eclipse.org/Xtext/

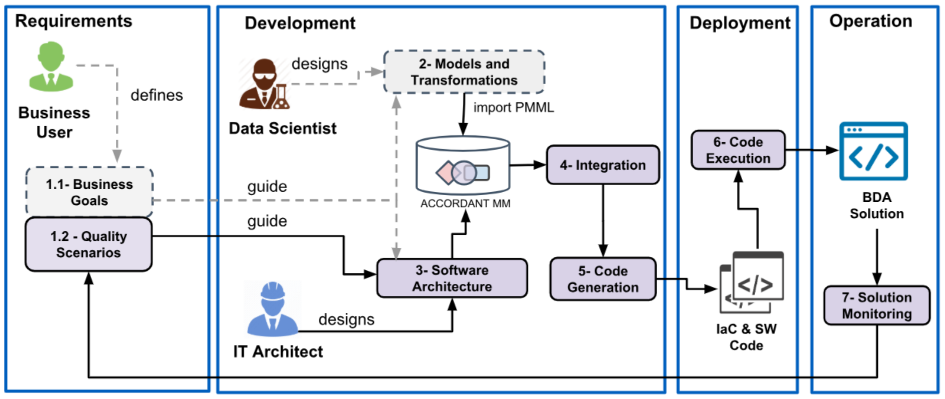


Figure 1: Proposal overview

4.1 Functional Viewpoint (FV)

FV describes the functional components, connectors of the analytics solution and their relationships in a technology-neutral way. Fig. 2 depicts a FV metamodel extract. *Component* metaclasses are specialized in *Ingestors*, *Transformers*, *Estimators* and *Sinks*. *Estimator* and *Transformer* are software component realizations of PMML data model and data transformer respectively, and the PMML file defines the analytics behavior. A *Component* exposes required and provided *Ports*. *Connectors* metaclasses transfer data or control flow among components through an input or output *Roles*. A set of connector types are defined based on the connector's classification proposed by Taylor et al. in [15]: *Stream*, *Event*, *Adaptor*, *Distributor*, *Arbitrator*, and *Procedure Call*.

4.2 Deployment Viewpoint (DV)

DV specifies how software artifacts (components and connectors) are deployed on computation nodes. This proposal extends the DV introduced in [5] by including containerization elements (dotted red lines) and extending QS attributes (dotted blue lines). Fig 3 details the main metamodel elements. DV metamodel comprises *Pod*, *ExposedPort*, and *Deployment* metaclasses to operationalize BDA applications in a specific technology. *Deployment Viewpoint (DV)* specifies *Devices*, *Pods*, *ExposedPorts*, *Services*, and execution environments (*ExecEnvironment*) where the *Artifacts* are deployed. A *Device* is a worker machine (physical or virtual) on which the Pods are deployed. A *Pod* is a group of one or more ExecEnvironment which can share storage and network. An *ExecEnvironment* represents a container with a Docker image, and specific resources requirements (CPU, memory). On this ExecEnvironment, both components and connectors can be installed. A *Deployment* specifies the desired state for a Pod's group and its deployment strategy, including the number of replicas. *Services* and *ExposedPorts* define the policies, addresses, ports, and protocols by which to access to Pods from outside the cluster network. A *QScenario* determines a quality attribute requirement (i.e. latency, availability, scalability, etc) for a specific *Artifact*. Thus, for instance, a QScenario could be defined as "latency <= 3 seconds for an artifact *X*", where artifact *X* corresponds to a software component or connector. An *Artifact* represents functional elements, i.e. components and connectors which are deployed in an ExecEnvironment, thus the mappings between FV and DV are materialized via *component* and **connector** references in the metaclass *Artifact* which point to FV's components. It is noteworthy that a *FV* model can be deployed in different *DV* models, and each DV model can fulfill QScenarios or not.

4.3 Code Generation

Once PMML, FV and DV models are designed and integrated, code generation takes place by means of model-to-text transformations. Code generation is twofold: software, and infrastructure (IaC) code. On the functional code side, each component and connector is assigned to a target technology regarding its attributes specified in the model (processing model, ML algorithm, delivery type, sync type, etc). Such assignment enables us to generate code for target technology constrained to the attributes. For instance, near real-time analytics could require stream or micro-batch processing provided by specific technologies like Apache Storm or Spark respectively. On the IaC side, DV models are transformed to Kubernetes' YAML files to create and configure infrastructure over Kubernetes cluster. YAML files contain Nodes, Pods, Deployments, and Services which are executed through Kubectl tool. In the last step, the performance metrics of the BDA solution are gathered to be compared to initial QS and evaluate the fulfillment of quality requirements.

5 EXPERIMENTATION IN AVIONICS

The experimentation validates if our proposal allows us to design, generate, monitor and evolve BDA solutions regarding performance QS. To do that, we use a case study in aviation safety to detect NMAC on different air space ranges with different deployment models while performance QS are monitored.

NMAC detection comprises a pairwise comparison within flights collection, a 2-combination of a set n, C_n^2), where n is the flight collection's size. Each comparison implies to calculate distance and time based on location, speed and heading to determine the risk level of NMAC assuming constant velocities, headings, and thresholds. A detailed explanation and reasoning of these caculations can be reviewed in [11]. By comparing such metrics calculated for each aircraft pair with thresholds such as time (*TTHR*), horizontal (*DTHR*) and vertical distance (*ZTHR*) is possible to determine the

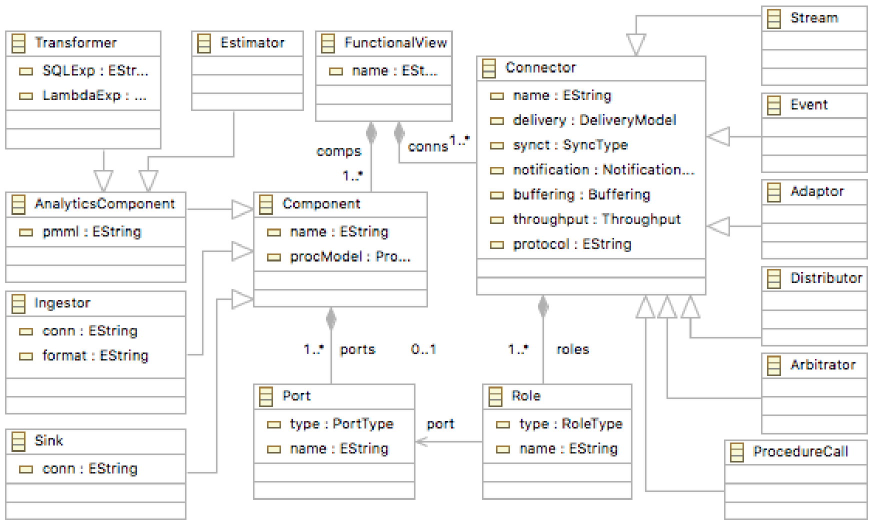


Figure 2: Excerpt of Functional Viewpoint of ACCORDANT metamodel.

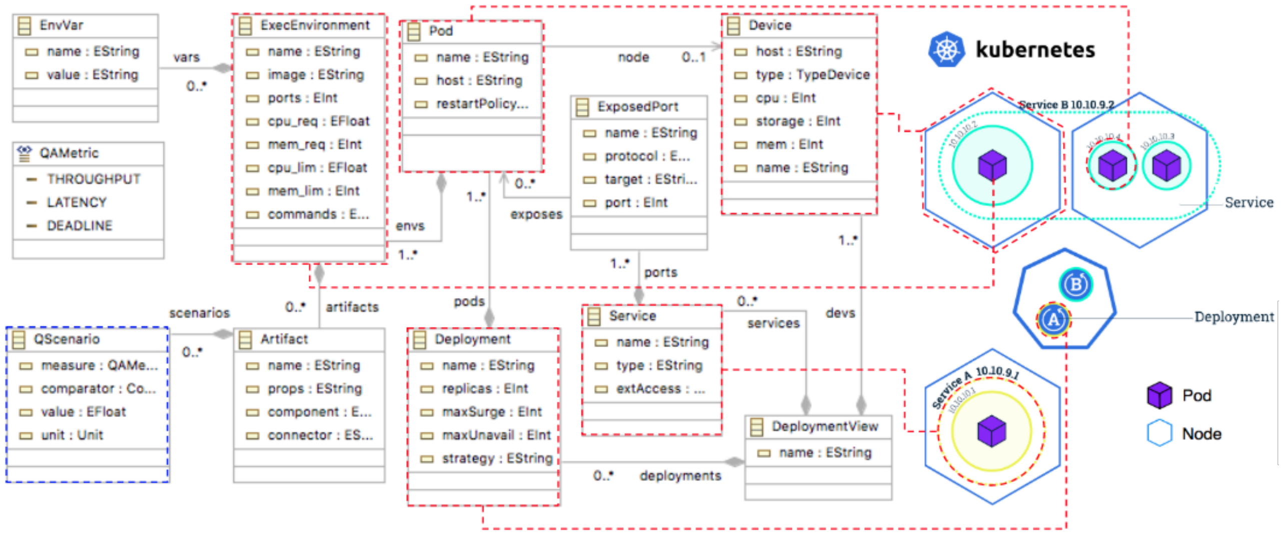


Figure 3: Excerpt of Deployment Viewpoint metamodel

alerting level: *warning* (3), *corrective* (2), *preventive* (1), and *none* (0) as defined by Detection and Avoid Systems.

Our experimentation comprises two use cases, *UC1* and *UC2*, which require different performance QS in batch and micro-batch processing. In UC1, the application computes NMAC alerting levels over a large dataset at rest to offer a consolidated report on a wide range of time. On the other hand, UC2 application consumes ADS-B data every minute to generate near real-time alerts to support avionics operation. The software component diagrams are detailed in Fig. 4. These use cases represent BDA applications since they combine semi-structured, near real-time data sources and analytics models to predict alerting levels. In this experimentation, we have used ADS-B exchange API[5], which generates live position data each minute. Live ADS-B position data are encoded in JSON responses which contain flights, their positions and speeds.

5.1 Development and Deployment Time

We measured the time spent in design, development, infrastructure provisioning, and deployment phases for both use cases with their respective deployment models. We compared our proposal with the traditional approach where each software component is developed from scratch to load PMML files. Connector middleware and technology platforms were installed and configured using Kubernetes.

[5]www.adsbexchange.com

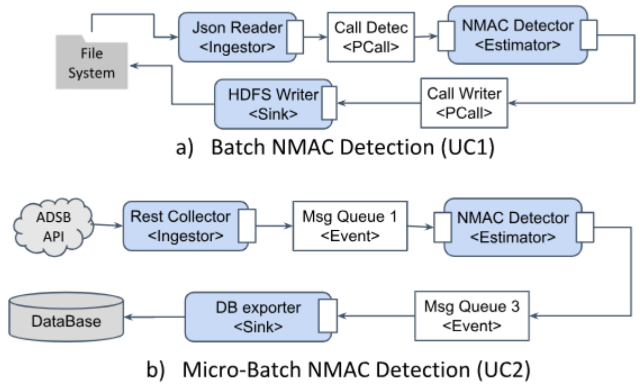


Figure 4: Component diagrams of NMAC Use Cases

The use cases were developed and deployed by two teams, each team was comprised of a developer and a system administrator.

5.2 Use Case 1 (UC1)

In UC1 (see Fig 4a), ADS-B data of eight-hours are stored in a distributed file system to be loaded by JSON Ingestor component. This reader component calls NMAC detector (Estimator) which classifies the alert level. Once alert levels for each flight pair are calculated, they are stored back in the file system. To compare different data size magnitudes, we collected flights' data for three air space ranges: 2 nmi (nautical miles), 20 nmi, and 200 nmi around JFK Airport. These ranges represent different application scopes to attend different demand levels: local, metropolitan, and regional. This use case does not have strong time restrictions due to its heavy workload, therefore, the QS is defined within a wide deadline.

5.3 Use Case 2 (UC2)

In UC2 (see Fig 4b), the Ingestor component consumes data through REST service of ADS-B exchange's API. ADS-B data are pushed in a message queue to be consumed by the NMAC detector component which classifies NMAC alerts. Given the near real-time nature of this application, latency is the critical quality attribute, and we evaluated this QS in two air space ranges: 2 nmi and 200 nmi, which demand different computation resources.

6 METHODOLOGY APPLICATION

We applied the ACCORDANT methodology detailed previously in Fig. 1 to design, develop, deploy and monitor UC1 and UC2.

6.1 Definition of Business Goals and QS

In this step, the business goals were defined for each use case as follows: UC1) Generate NMAC alerting levels for 8-hours ranges around JFK Airport's air space. UC2) Offer a near real-time alerting service which reports NMAC events in delimited airspace.

These business goals involve different constraints, and therefore different QS are specified for each use case. In UC1, the deadline of the predictor component should be less than or equal to 1 hour. On the other hand, in UC2, the latency of the predictor component should be less than or equal to 3 seconds.

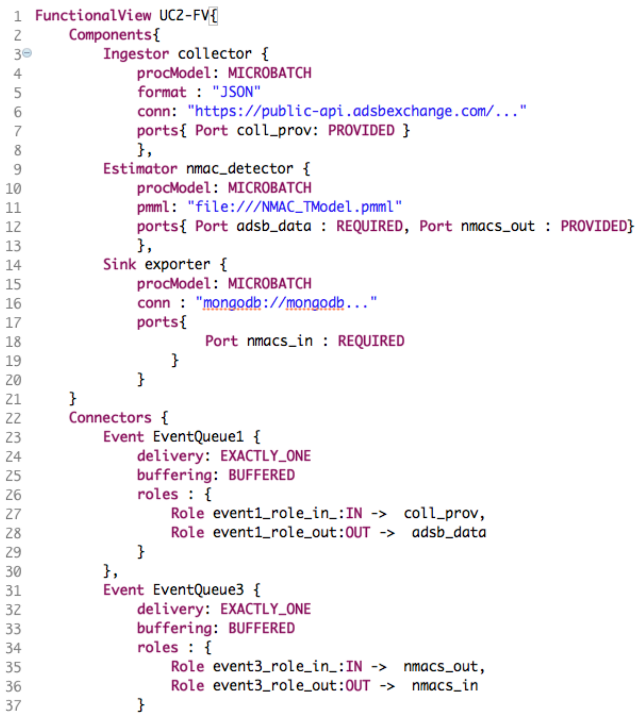

```
1  FunctionalView UC2-FV{
2      Components{
3          Ingestor collector {
4              procModel: MICROBATCH
5              format : "JSON"
6              conn: "https://public-api.adsbexchange.com/..."
7              ports{ Port coll_prov: PROVIDED }
8              },
9          Estimator nmac_detector {
10             procModel: MICROBATCH
11             pmml: "file:///NMAC_TModel.pmml"
12             ports{ Port adsb_data : REQUIRED, Port nmacs_out : PROVIDED}
13             },
14         Sink exporter {
15             procModel: MICROBATCH
16             conn : "mongodb://mongodb..."
17             ports{
18                     Port nmacs_in : REQUIRED
19                 }
20             }
21     }
22     Connectors {
23         Event EventQueue1 {
24             delivery: EXACTLY_ONE
25             buffering: BUFFERED
26             roles : {
27                 Role event1_role_in_:IN ->  coll_prov,
28                 Role event1_role_out:OUT ->  adsb_data
29             }
30         },
31         Event EventQueue3 {
32             delivery: EXACTLY_ONE
33             buffering: BUFFERED
34             roles : {
35                 Role event3_role_in_:IN ->  nmacs_out,
36                 Role event3_role_out:OUT ->  nmacs_in
37             }
```

Figure 5: Excerpt of Functional Specification of Use Case 2 Using the ACCORDANT DSL

6.2 Analytics Model Development

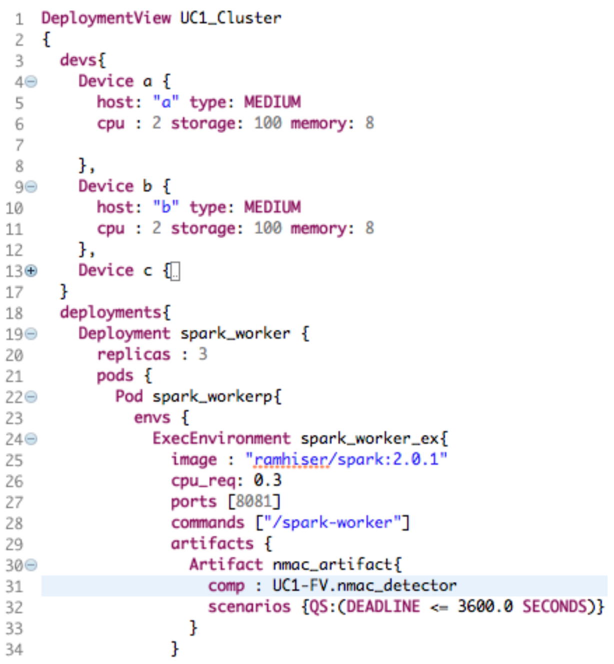
Model training and evaluation are developed outside ACCORDANT, but the resulting model is exported to PMML file to be loaded in FV model. ADS-B dataset (360 live positions data) was collected on December 7th, 2018 from 14:00 to 20:00. We trained and validated a decision tree model after labeling this dataset with the alert level (from 0 to 3) regarding Well Clear Criteria proposed in DAA Phase 1[6]. The independent variables of the analytics model are: *flight1_id*, *flight2_id*, *tcpa*, τ_{mod}, v_z, $|s_z|$, *dcpa*, and the dependent variable is the alerting level (*a*). The model learned the threshold boundaries of each alert level and exhibited a high accuracy (99.987%), so it was exported as PMML file to be referenced by the FV model.

Listing 1 details an excerpt of the decision model in PMML format. At the beginning of XML file, data model's input and output structures are defined in sections *Data Dictionary* and *Mining Schema*, followed by the model specification, in this case, tree model's conditions. A PPML file's extract of the tree model, which assigns the highest alerting level=3, is defined by conditions: τ_{mod} <= 54.3718 (in seconds), *sz_norm* <= 0.0761 (in nmi), τ_{mod} <= 24.6105 (in seconds) and *dcpa* <= 0.6387 (in nmi).

6.3 Functional View Design

FV models were designed using ACCORDANT DSL to specify the component-connector structure for each use case. As an example, Fig. 5 shows an excerpt of UC2 FV where three components (lines 3, 9, and 14) and two connectors (lines 23 and 31) are specified. In

[6]https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20180002420.pdf



```
DeploymentView UC1_Cluster
{
  devs{
    Device a {
      host: "a" type: MEDIUM
      cpu : 2 storage: 100 memory: 8

    },
    Device b {
      host: "b" type: MEDIUM
      cpu : 2 storage: 100 memory: 8
    },
    Device c {…
  }
  deployments{
    Deployment spark_worker {
      replicas : 3
      pods {
        Pod spark_workerp{
          envs {
            ExecEnvironment spark_worker_ex{
              image : "ramhiser/spark:2.0.1"
              cpu_req: 0.3
              ports [8081]
              commands ["/spark-worker"]
              artifacts {
                Artifact nmac_artifact{
                  comp : UC1-FV.nmac_detector
                  scenarios {QS:(DEADLINE <= 3600.0 SECONDS)}
                }
              }
```

Figure 6: Excerpt of Cluster Deployment Specification for Use Case 2 Using the Proposed DSL

addition, *nmac_detector* (lines 9–13) uses micro-batch processing model and has assigned the PMML file exported in Section 6.2. *Nmac_detector* component exposes ports *adsb_data* (provided) and *nmacs_out* (required) to receive ADS-B data and send NMAC results through connectors *EventQue1* and *EventQue3* respectively. Connectors' properties such as delivery and buffering were configured to determine the features of event-oriented connectors.

6.4 Deployment View Design

We designed two DV models *uc1-local* and *uc1-cluster* were to deploy the UC1 FV. The deployment *uc1-local* instantiates estimator component in Python and Sci-kit learn applying serial execution in a single machine with 2.5 GHz Intel Core i5 and 8 GB memory. Deployment *uc1-cluster* takes advantage of distributed processing to increase the throughput on Apache Spark with more computing resources. Hence, *uc1-cluster* deployment model defines a Spark cluster with the master node and worker nodes (three replicas of Kubernetes' Pod). This cluster was deployed using Elastic Container Service for Kubernetes (Amazon EKS) on EC2 instances t2.medium.

An extract of *uc1-cluster* specified in ACCORDANT Deployment DSL is shown in Fig. 6. *Spark_worker::Deployment* has 3 replicas. The *nmac_artifact::Artifact* has associated the component *nmac_detector::Estimator* declared in the functional model UC1. This artifact's code will be generated in Spark (batch processing technology) to expect the inputs defined in the PMML file and predict alert levels. Finally, this *nmac_artifact* is bound to a QS that defines a maximum deadline of 3,600 seconds.

A single *uc2-local* deployment model was defined for UC2 to run in a single machine with 2.5 GHz Intel Core i5 and 8 GB memory. This *uc2-local* model defined a single node-pod with Apache Spark and Kafka, where the estimator and event connectors were installed. This NMAC estimator's QS specifies that latency must be less than or equal to 3 seconds.

Listing 1: Excerpt of Decision Tree Model for Alert Level Prediction in PMML code

```
<PMML xmlns="http://www.dmg.org/PMML-4_3" version="4.3">
...
<DataDictionary>
    <DataField name="a" optype="categorical" dataType="integer">
        <DataField name="a" optype="categorical" dataType="integer">
            <Value value="0"/>
            <Value value="1"/>
            <Value value="2"/>
            <Value value="3"/>
        </DataField>
    </DataField>
    ...
    <DataField name="sz_norm" optype="continuous" dataType="float"/>
</DataDictionary>
<TreeModel functionName="classification" splitCharacteristic="binarySplit">
    <MiningSchema>
        <MiningField name="a" usageType="target"/>
    ...
        <MiningField name="dcpa"/>
    </MiningSchema>
    <Node id="1">
        <True/>
        <Node id="2">
            <SimplePredicate field="double(t_mod)"
            operator="lessOrEqual"
            value="54.3718"/>
            <Node id="3">
                <SimplePredicate field="double(sz_norm)"
                operator="lessOrEqual"
                value="0.0761"/>
                <Node id="4">
                    <SimplePredicate field="double(t_mod)"
                    operator="lessOrEqual"
                    value="24.6105"/>
                    <Node id="5" score="3">
                        <SimplePredicate field="double(dcpa)" operator="lessOrEqual"
                        value="0.6387"/>
                            <ScoreDistribution value="1" recordCount="2.0"/>
                            <ScoreDistribution value="3" recordCount="69.0"/>
                    </Node>
                ...
</TreeModel>
</PMML>
```

6.5 Integration and Code generation

Once FV and DV models are designed and integrated, the code generation produced the YAML files for Kubernetes deployments and services. These YAML files contained provision and configuration policies of Kubernetes cluster. Listing 2 shows an example of generated YAML files. Besides this, software components and connectors are manually associated to specific technologies regarding their constraints. Once these associations are defined, the functional (technology-specific) code can be generated. Listing 3 shows an extract of the generated code for UC2's Estimator which implemented the PMML model in Spark Streaming technology. This implementation defines data input and output from the Data Dictionary and Mining Schema embedded in PMML specifications. The mappings between artifacts and components allow us to include logging code regarding the relevant QS. In the current version, PMML loading and evaluation have been implemented using JPMML API[7].

6.6 Code Execution

Kubernetes code was executed on the AWS cloud using Amazon Elastic Container Service for Kubernetes (Amazon EKS) and Elastic Compute Cloud (EC2). After that, the software code was installed over the EKS cluster to operationalize the end-to-end solution.

[7]https://github.com/jpmml/

6.7 Solution Monitoring

Performance metrics were collected in operation and validated for each use case (UC1 and UC2) against QS defined in Section 6.1. As a result, different deployment configurations (local and cluster) were designed, deployed and monitored.

Listing 2: Generated YAML Code from Deployment Specification for Kubernetes (Extract)

```
kind: Deployment
metadata:
  name: spark-worker
spec:
  replicas: 3
    spec:
      containers:
      - name: spark-worker-ex
        image: ramhiser/spark:2.0.1
        command: [/spark-worker]
        ports:
        - containerPort: 8081
        resources:
          requests:
            cpu: 0.25
```

Listing 3: Generated Java Code of NMAC Estimator Component for Spark Streaming

```
InputStream pmmlFile = new URL("DTModel.pmml")
EvaluatorBuilder b = new LoadingModelEvaluatorBuilder().load(pmmlFile);
Evaluator eval = builder.build();
TransformerBuilder pmmlTransformerBuilder =
new TransformerBuilder(evaluator)
    .withTargetCols().exploded(true);
List<StructField> fields = new ArrayList<StructField>();
fields.add(DataTypes.createStructField("a",DataTypes.IntegerType, true));
...
fields.add(DataTypes.createStructField("sz_norm",DataTypes.FloatType, true));
StructType schema = DataTypes.createStructType(fields);
Transformer pmmlTransformer = pmmlTransformerBuilder.build();
Dataset<Row> inputDs = sparkSession.read().schema(schema).csv("adsb.json");
TransformerBuilder tb = new TransformerBuilder(eval);
Transformer transformer = tb.build();
Dataset<Row> resultDs = transformer.transform(inputDs);
```

7 RESULTS

This section presents and discusses the results obtained during the design, development and operation phases for both use cases (UC1 and UC2) deployed in different DV models (local and cluster) and data ranges (2 nmi, 20 nmi, 200 nmi).

7.1 Development and Deployment Time

Table 1 reports time invested for each BDA development phase, approach, and use case. The traditional approach required less time for both use cases of software design since it is mainly used for documentation and communication purposes, and it does not require many details and formal definitions. However, in our approach, the design requires formal specifications with more detail since DV and FV models are executable first-class citizens, hence the design time invested with this proposal is between 6 and 8 times greater. In contrast, development time was reduced up to 70.8% using our approach because code generation accelerated software development. Similar reductions, from 66.6% to 75%, were reported in infrastructure provision, deployment and re-deployment due to IaC generation. In total, we observed a time reduction of 55.5% (12.5 hours) in development and deployment for UC1, and 57.3% (17.5 hours) for UC2 which facilitated monitoring and assessment of performance metrics for BDA applications. These results are consistent with the metrics presented in [5]. In the following sections, we detail and compare the collected performance metrics for each use case to illustrate the QS monitoring.

Table 1: Design, Development and Deployment Time Invested (in hours)

US	Approach	Design	Dev.	Infrastr.	(Re)Deploy	Total
UC1	Traditional	0.5	18	2	2	22.5
UC2	Traditional	0.5	24	3	3	30.5
UC1	Our	3	6	0.5	0.5	10
UC2	Our	4	7	1	1	13

Table 2: Flights, Pair Comparisons and Alerts for each Collected Data Range

Data Range	Flights	Comparisons	Alerts
2 nmi	3,370	9,932	27
20 nmi	20,999	568,693	1,061
200 nmi	138,590	20,506,061	1,669

7.2 Batch Processing Use Case (UC1)

In UC1, we executed distributed batch processing over three datasets at rest (2, 20 and 200 nmi). We collected the execution time of the Estimator components with two versions of DV models (uc1-local and uc1-cluster), and we compared execution times against the defined deadline. Sizes, flights, and pairwise comparisons for each data range are reported in Table 2. It is noteworthy that pairwise comparisons increase exponentially in respect of data range. In total, this experimentation found in the widest range (200 nmi) 1,669 NMAC alerts classified as follows: 228 first-level, 648 second-level, and 793 third-level. The time range between 15:00 and 16:00 hours observed the highest number of NMAC alerts, located around the main airports of northeast's cities: New York City, Philadelphia, Boston and Washington. We confirmed that the greater number of reported flights on ADS-B service, the more frequency of NMAC alerts were found due to the higher airspace's density.

Regarding the performance metrics monitoring, Fig. 7 details the execution time results of estimator component for *uc1-local* and *uc1-cluster* models previously described in Section 6.4. Deployment *uc1-local* (sequential computation) took 24 seconds for 2 nmi data range; for 20 nmi, 186 seconds; and for 200 nmi 5.588 seconds. On the other hand, *uc1-cluster* (distributed computation) lasted 128 seconds for 2 nmi; 153 seconds for 20 nmi; and 1,619 seconds for 200 nmi. These results showed that while *uc1-local* deployment took less time than *uc1-cluster* for 2 and 200 nmi data sets, *uc1-local* breached the QS of 3,600 seconds (dotted red line in Figure 7). *uc1-local* and *uc1-cluster* reported similar execution times with 20 nmi data range (186 and 153 seconds respectively).

7.3 Micro-Batch Processing Use Case (UC2)

In UC2, we run *uc2-local* deployment, previously described in Section 6.4, and collected latency metrics for an hour with 2 nmi and 20 nmi data ranges. Fig. 8a details flights pairs processed per

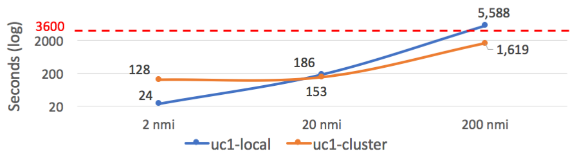


Figure 7: Execution Time in Batch Processing (UC1) for Deployments and Data Ranges

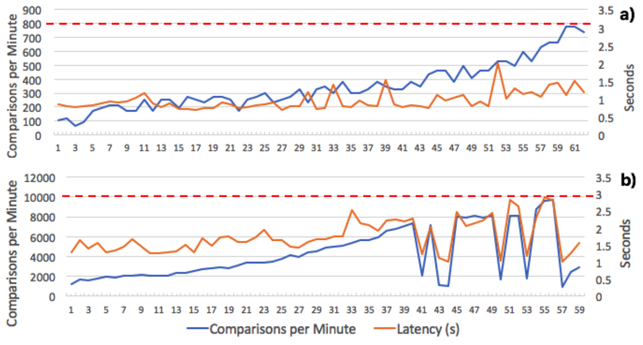


Figure 8: Estimator's Latency in Micro-batch Processing (UC2) in a) 2 nmi, and b) 20 nmi.

minute (left vertical axis) and latency (right vertical axis) for 2 nmi data range. In average, each ABS-B position report contained 26.03 flights which implied 345.2 pairwise comparisons. During the whole processing, latency was significantly lower (between 0.69 and 1.99 seconds) than the performance QS of 3 seconds (in dotted red line) showing a behavior associated with the number of flights.

Fig. 8b depicts flights pairs processed per minute (left vertical axis) and NMAC estimator latency (right vertical axis) for 20 nmi data range. In average, each report contained 88.25 flights which implied 4,217.42 pairwise comparisons. During the data processing, latency remained bellow performance QS (dotted red line) showing a very similar trend when compared to flight pair amount, but with 139 flights (9,591 pair comparisons) latency was closer to the latency limit: 2.91 seconds. Regarding these results, near real-time processing time for a larger dataset with the same *uc2-local* deployment could not fulfill the latency QS, therefore, distributed processing deployment should be required for wider data ranges.

8 CONCLUSIONS

We have presented a DevOps and DSM proposal to design, deploy and monitor BDA solutions. Our results indicated a speeding-up of design, implementation, and (re)deployment of BDA solutions. We obtained time reductions in design, development, and deployment from 55.5% to 57.3% in use cases. This approach advocates for a separation of concerns what facilitated testing different deployment strategies associated with the same functional model.

We executed data processing to evaluate the fulfillment of performance QS specified for two use cases in avionics. Our results highlighted the cases where distributed processing presents better performance than local-sequential processing, and the deployments which incur in QS violations. Some challenges for technology-specific implementations emerge since PMML loading and data transformations are generic through JPMML API, and they are not considered code optimization.

As future work, the performance metrics collected along with FV and DV models could allow us to propose a performance model to predict the expected behavior based on the functional model, deployment model, and target technology to recommend the optimal architecture configuration regarding QS. We are also working on verifying correctness properties over ACCORDANT models such as architectural mismatches. This approach has been used for deploying analytics components and connectors on virtual machines over cloud infrastructure, but different paradigms such as serverless or fog computing could open new challenges and research lines.

ACKNOWLEDGMENTS

This research is supported by Fulbright Colombia and the Center of Excellence and Appropriation in Big Data and Data Analytics (CAOBA), supported by the Ministry of Information Technologies and Telecommunications of the Republic of Colombia (MinTIC) through the Colombian Administrative Department of Science, Technology, and Innovation (COLCIENCIAS) within contract No. FP44842-anexo46-2015. The authors would also like to thank an Amazon Web Services educational research grant.

REFERENCES

[1] Mohammad Alrifai, Holger Eichelberger, Cui Qui, Roman Sizonenko, Stefan Burkhard, and Gregory Chrysos. 2014. *Quality-aware Processing Pipeline Modeling.* Technical Report. QualiMaster Project.
[2] Matej Artac, Tadej Borovsak, Elisabetta Di Nitto, Michele Guerriero, Diego Perez-Palacin, and Damian Andrew Tamburri. 2018. Infrastructure-as-Code for Data-Intensive Architectures: A Model-Driven Development Approach. In *2018 IEEE International Conference on Software Architecture (ICSA).* IEEE, 156–165.
[3] Len Bass, Paul Clements, and Rick Kazman. 2012. *Software Architecture in Practice.* Addison-Wesley.
[4] Len Bass, Ingo Weber, and Liming Zhu. 2015. *DevOps: A Software Architect's Perspective.* Addison-Wesley Professional.
[5] Camilo Castellanos, Dario Correal, and Juliana-Davila Rodriguez. 2018. Executing Architectural Models for Big Data Analytics. In *Software Architecture*, Carlos E. Cuesta, David Garlan, and Jennifer Pérez (Eds.). Springer International Publishing, Cham, 364–371.
[6] Hong-Mei Chen, Rick Kazman, and Serge Haziyev. 2016. Agile Big Data Analytics for Web-Based Systems: An Architecture-Centric Approach. *IEEE Transactions on Big Data* 2, 3 (sep 2016), 234–248. https://doi.org/10.1109/TBDATA.2016.2564982
[7] Holger Eichelberger, Cui Qin, Klaus Schmid, and Claudia Niederée. 2015. Adaptive Application Performance Management for Big Data Stream Processing. In *Symposium on Software Performance.*
[8] M. Gribaudo, M. Iacono, and M. Kiran. 2017. A Performance Modeling Framework for Lambda Architecture Based Applications. *Future Generation Computer Systems* (jul 2017). https://doi.org/10.1016/j.future.2017.07.033
[9] Michele Guerriero, Saeed Tajfar, Damian Tamburri, and Elisabetta Di Nitto. 2016. Towards A Model-Driven Design Tool for Big Data Architectures. In *2nd IWBDSE.*
[10] Yicheng Huang, Xingtu Lan, Xing Chen, and Wenzhong Guo. 2015. Towards Model Based Approach to Hadoop Deployment and Configuration. In *12th WISA.* IEEE, 79–84. https://doi.org/10.1109/WISA.2015.65
[11] César Munoz, Anthony Narkawicz, James Chamberlain, Maria C Consiglio, and Jason M Upchurch. 2014. A Family of Well-Clear Boundary Models for the Integration of UAS in the NAS. In *14th AIAA Aviation Technology, Integration, and Operations Conference.* 2412.
[12] Rajiv Ranjan. 2014. Streaming Big Data Processing in Datacenter Clouds. *IEEE Cloud Computing* (2014), 78—-83. http://mahout.apache.org
[13] Nick. Rozanski and Eoin. Woods. 2005. *Software Systems Architecture : Working with Stakeholders Using viewpoints and Perspectives.* Addison-Wesley. 546 pages.
[14] Rajinder Sandhu, Sandeep K Sood, R Sandhu, and S K Sood. 2015. Scheduling of Big Data Applications on Distributed Cloud Based on QoS parameters. *Cluster Computing* 18 (2015), 817–828. https://doi.org/10.1007/s10586-014-0416-6
[15] Richard N Taylor, Nenad Medvidovic, and Dashofy Eric M. 2010. *Software Architecture: Foundations, theory and practice.* John Wiley and Sons, Inc.