A Big Data Quality Preprocessing and Domain Analysis Provisioner Framework using Cloud Infrastructures

Dirk Hölscher*, Timo Bayer*, Philipp Ruf*, Christoph Reich* and Frank Gut†
*Institute for Cloud Computing and IT Security, Furtwangen University of Applied Science, 78120 Furtwangen, Germany
Email: {dirk.hoelscher, timo.bayer, philipp.ruf, christoph.reich}@hs-furtwangen.de
† Daimler AG, 70565 Vaihingen, Germany
Email: frank.gut@daimler.com

***Abstract*—Big data is a new economic driver for many advanced technology domains, such as autonomous driving, reusable rockets or cancer research. Generating knowledge from large amounts of data in such domains, will become increasingly important. Management and processing is done in powerful big data infrastructures located in the cloud. Diversifying requirements for the different big data domains require new uniform and adaptable architectural patterns that can be implemented and changed without much effort. This paper introduces a generic domain independent cloud big data framework, focussing on simplifying data preprocessing tasks and the deployment of data analysis environments by using adaptable and easy to configure domain specific components.**

Keywords–Domain Analysis Provisioner; Big Data Frameworks; Reusable Architectural Patterns; Cloud Computing

I. INTRODUCTION

Requirements for big data infrastructures strongly differ depending on the domain and individual/statutory provisions. As rapidly as data volumes change, business objectives will change based on newly discovered knowledge. The fast-growing need for integrating new data to analyze an increasing amount of data in new analysis domains and for provisioning new analysis methods/approaches for big data analysis requires fast and simple adjustable big data infrastructures that easily cope with changing requirements.

Limiting the administrative effort by using reusable components based on common and interchangeable technologies would also enable small and middle-class businesses to do big data analysis. Due to the continuously increasing data volumes more and more resources are needed for fast and reliable analysis task execution. Relocating traditional big data infrastructures into the cloud using virtualized environments, benefiting from the cloud's essential characteristics (resource pooling or rapid elasticity) will help reduce costs, by providing less complex and reusable frameworks. Big Data infrastructures can be operated as a private cloud (self-hosted service) or in a multi-tenant public cloud environment, depending on company's preferences and guidelines. Encouraging businesses of all sizes to use and analyse their data, non-abstract and easy adaptable infrastructure patterns are needed.

The main contribution of this paper introduces a generic data analysis framework, which can be adapted to fulfil different domain and data analyst's requirements. The to be introduced infrastructure consists of two main components, the preprocessing framework and the domain analysis provisioner framework. The preprocessing framework is responsible for transferring, storing, persisting and processing data provided by various data sources, whereas the domain analysis provisioner framework provides Platform-as-a-Service(PaaS) environments providing necessary analytical tools.

The paper is organised as follows: In Section II, related work will be presented. Section III describes components and functionality of the preprocessing framework. In section IV, the domain analysis provisioner framework, providing machine-learning and analytic environments is described. The next section elaborates important characteristics of the proposed framework followed by section VI, which summarizes and concludes the paper.

II. RELATED WORK AND STATE OF THE ART

In the following subsections, we present related work about big data architectural patterns and broker-based systems.

A. Architecture Patterns

In the field of big data architectures, there are several best practice design patterns that can be used.

The goal of the *Lambda* architecture pattern (Marz and Warren [1]), is to redefine classic data computation using modern big data technologies, while focusing on high scalability, realtime and immutability data processing. While scalability and realtime capabilities are achieved by using modern big data technology, achieving immutability requires a particular design. They described an architecture, in which input data is computed twice using batch processing as well as stream processing while the corresponding results are stored separately. This approach enables to recompute the processing results in the case of possible malfunctions. The disadvantages of this design pattern are (i) the necessity to develop and maintain redundant program logic within the two processing models and (ii) the resulting storage overhead.

To address these problems another design pattern, called *Kappa* architecture, was introduced by Kreps in [2]. The *Kappa* architecture focuses on the capability to recompute the processing results in case of possible malfunctions but eliminates the necessity to manage redundant program logic. To achieve this, the *Kappa* architecture utilizes the messaging system *Apache Kafka* to integrate data streams into the processing engine. Due to the capability of *Apache Kafka* to persist messages for a given time period, it is possible to recompute the data streams in case malfunctions were detected. The infrastructure mentioned in this paper is strongly based on the principles of these two architectural patterns.

The National Institute of Standards and Technology (NIST) defined a general reference architecture for big data applications. As described in [3], the proposed architecture comprises

 ISBN: 978-1-61208-631-6

the five functional components *System Orchestrator, Data Provider, Big Data Application Provider, Big Data Framework Provider* and *Data Consumer*, which describe common functionalities big data applications should have. Based on the NIST's big data Reference architecture, we built a generic framework utilizing the components defined by NIST.

Rahmen et al. describe in [4] a platform to process and analyze healthcare information. The platform mainly relays on predefined functionalities and hides technology-specific implementations. The work introduces centralized approaches to manage the platform and deploy customized applications. Our platform, considers several of the introduced approaches while trying to fill the gap of domain-independent platforms. The purpose of the Cask Data App Platform [5], is to speed up the development of data analysis tasks. Therefore, several layers of abstraction are provided, such as an easy to use API and container based runtime environments to run and deploy analysis tasks. They provide graphical interfaces to quickly create ad hoc analysis tasks. Due to the limitations, of their imperative approach, we introduce a declarative solution to achieve maximum flexibility for analysis tasks.

B. Broker Systems

There is a wide variety of cloud based broker systems supporting the intermediation and aggregation of services offered by heterogeneous Cloud Service Providers (CSP) containing potential complex mechanisms related to any subdomain. A cloud broker is the basis of our domain analysis provisioner framework.

Roy et. al. show a Quality of Service (QoS) enhanced virtual resource broker [6] that allow different CSPs to register their resources at the broker by declaring, e.g. non-functional properties of their services. A broker client may request a virtual resource with a certain amount of assured quality by utilising the systems inter-CSP manager. The work on hand delimits itself from the QoS brokerage and the dedicated cost- and billing system, but focusses on the request analyzer as well as on the resource allocation manager.

Another economic driven cloud service facilitator was presented by Kim et. al. in [7]. The Virtual Machine (VM) reservation based cloud service broker considers the executed applications inside the leased resources, which overlaps with the scope of the work on hand. As workload increases, the *VM reservation module* starts to conduct demand prediction and reservation planning for new resources.

In order to develop efficient scheduling strategies of jobs in federated cloud environments, Pacini et. al. suggested an Ant Colony Optimization (ACO)-based approach, implemented as a three-layered broker in [8]. An underlying layer of the broker calculates the most suitable datacenter for a particular job, depending on the results of a parameter sweep experiment (e.g., simulations with repeatedly changing input parameters).

III. A REUSABLE BIG DATA INFRASTRUCTURE FOR PROCESSING MASSIVE DATASETS

The interpretation of massive datasets called big data analysis offers a promising potential for various industries and research fields. Due to the characteristic properties of these datasets (see [9]) as their volume, variability, and velocity, the development of such an application poses a particular challenge. In order to reduce the emerging development costs for adapting the data analysis task caused by different analysis domains, it is important to identify general-purpose concepts. In the following section, we introduce a configurable, expandable and reusable big data infrastructure to address these problems.

A. Framework Requirements

The main objective of the described framework is to abstract universally valid concepts, such as collecting and provisioning input data and orchestrating them in an architectural pattern, which is easily adaptable for domain-specific use cases. To address these challenges, we defined modular layers, which can be easily applied individually or combined to build up larger data analysis tasks. As depicted in Figure 1 these layers consist of various parts that can be divided into infrastructural system and domain-specific components. The infrastructure components are designed and implemented according to common big data application functionalities (see [10]). To tackle the big variety of scenarios the approach is focusing on domain-specific components. The functionality of these components depends on the provided data structures, data sources, and required processing tasks. Therefore, the framework comprises generic implementations to abstract the usage of technologies, as well as providing easy to use and flexible interfaces including communication functionalities for the combined infrastructure. Besides the identification and concatenation of suitable technologies and universally valid functionalities, another aspect comprises the definition of consistent interfaces and the logical separation of application components from collecting raw data until their final usage. The following subsections describe the responsibilities of the defined layers and the involved components.

B. Information Layer

Selecting and integrating relevant data sources for data collection is the foundation of every big data application. Large amounts of data sources, such as smart devices, wearables or sensors in manufacturing facilities, result in the necessity to involve multiple data sources and correlate the gathered data. Therefore, the *Information Layer* can be considered as a logical representation of various data sources.

C. Messaging and Distribution Layer

The main goal of the *Messaging and Distribution Layer* (see Figure 1) is to integrate the data sources and provide generated data to the succeeding infrastructure components. A domain-specific *Connector* has to be extended with specific collection functionalities to integrate data sources. For example, this may be used to establish a connection to a remote data store, understand the dataset format and much more. Depending on the application the data sources can vary from static datasets to realtime data collected by sensors.

Integrating Static Datasets: While dealing with batch data the connector will initially store new data within the *FileCache*. The *FileCache* is realized as a distributed Network File System (NFS) using the Parallel NFS (pNFS) standard providing fast and scalable functionality to store data within the infrastructure for further distribution. After uploading the data into the *FileCache* the *StorageConnector* is informed about the occurrence of new data. The responsibility of this component comprises uploading data from *FileCache* to a distributed storage engine

 ISBN: 978-1-61208-631-6

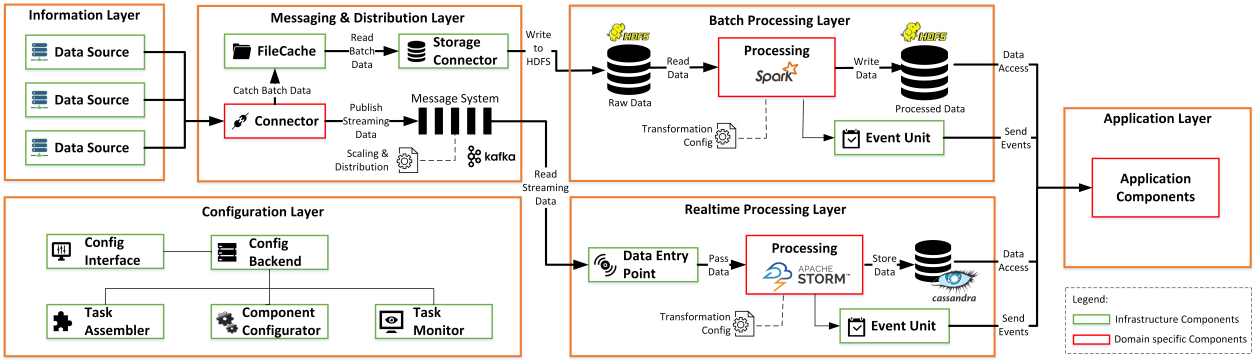

Figure 1. Architecture of the big data preprocessing framework

realized as a Hadoop Distributed File System (HDFS) cluster. Depending on the size of data, uploading to a HDFS cluster can be very time consuming, due to the necessity of splitting data into several blocks, which will be replicated and distributed in the storage cluster. In order to be able to manage massive datasets in a timely manner, this functionality can be highly parallelized using multiple instances of the *StorageConnector*. Using the *Messaging and Distribution Layer*, processing this kind of data only requires the definition of how to read data within the *Connector*. To speed up the development of such a component a base class was implemented providing functionalities for the *FileCache* and data persistence.

Integrating Realtime Data: Dealing with streaming data requires a different behaviour from the *Messaging and Distribution Layer*. In this case, the *Connector* has to continuously read small datasets and distribute it to the succeeding infrastructure components. The base class for implementing a *Connector* provides the functionality to distribute data within the infrastructure. To ensure a scalable and reliable data distribution the messaging system *Apache Kafka* has been chosen. Therefore, the base class acts as a *Producer* and provides the functionality to publish new data to multiple broker instances, which subsequently can be processed in the *Realtime Processing Layer*.

D. Batch Processing Layer

While the *Messaging and Distribution Layer* can be used independently, most big data scenarios require further data investigation (e.g, quality check, improvement, etc.) by a domain-specific *Apache Spark* job. Using *Apache Spark* for this kind of processing benefits from high scalability, fault tolerance and in-memory processing capabilities, which are required for processing large data sets efficiently. Implementing such a task strongly depends on the provided data structures and therefore, predefining general valid functionalities is impossible. Therefore, our approach focuses on integrating the preprocessing tasks within the infrastructure. Our approach provides base implementations for storage engine connections and for the integration of result postprocessing. The results of such a task can be divided into two categories: a) Generating new data (e.g., aggregations or enhanced data records). To store and provide the results for further usage, the base implementation allows to store new data into the HDFS cluster. b) Storing data tags that can be found looking for specific data patterns through data correlation. The *EventUnit* can be used within processing jobs to inform succeeding components about new results (data tags) or to directly exchange the achieved results. The *EventUnit* uses the messaging system *Apache Kafka* to enable a nearly linear scalability and can be configured as a *Producer* or a *Consumer*.

E. Realtime Processing Layer

The realtime layer enables the processing of streamed data individually or by combining it with layer described in previous sections to cover more complex processing scenarios (e.g., creating a *Lambda* architecture). The collected data for the *Realtime Processing Layer* is provided by the messaging system *Apache Kafka* and integrated by the *DataEntryPoint* component. This component acts as a *Apache Kafka Consumer* and can be configured for the given data structures. The processing task can be developed by implementing a domain specific *Apache Strom Topology* with nearly linear scalability, fault tolerance and realtime capabilities, representing important requirements in realtime scenarios. Furthermore, a general-purpose data quality module, validating input data against user configured thresholds for missing values, outliers or min/max values was implemented. Data composition is described using *XSD* schemes defining data structures and constraints for a given dataset. Processing realtime data mainly relies on the discovery of patterns within a given time frame, to generate more significant events for further operations. Furthermore, it is possible to use the previously described *EventUnit* to inform succeeding components about the occurrence of such events. If the processing task involves data manipulation it is possible to store the results within an *Apache Cassandra* cluster.

F. Application Layer

Depending on the given use case, there is a wide range of possible tasks to perform after processing the input data, like building models using machine learning technologies. To tackle the high variability of analysis tasks the *Application Layer* contains domain-specific components, which will access and utilize interfaces provided by the underlying layers. The *EventUnit* abstracts the direct data access to the corresponding storage engine.

G. Configuration Layer

The *Configuration Layer* contains all necessary components to provide centralized infrastructure management, enabling the implementation of different analysis tasks. One

 ISBN: 978-1-61208-631-6

important responsibility of this layer includes a centralized and automated deployment of domain-specific components. To automate the distribution of components and their correct configuration, *DeploymentPackages* are introduced, containing the executables and specific configuration files representing the component's type (e.g., *Connector*, *Application* or *Processing* Component) and their expected parameters.

While the component type defines how the corresponding executables will be distributed and executed within the cluster, parameters can be used to execute multiple instances of the same processing task individually. Starting a processing task relies on an uploaded *DeploymentPackage* using a web interface (see Figure 2). After the *DeploymentPackage* is uploaded,

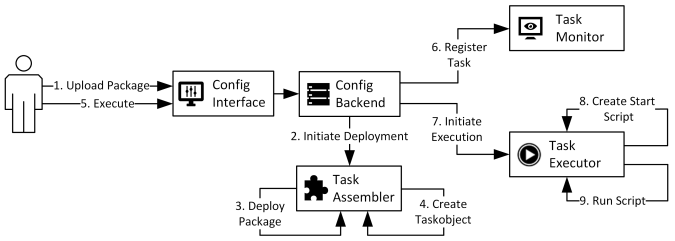

Figure 2. Process to Execute a Processing Task

the involved components are deployed to their assigned nodes by the *Task Assembler* component. Furthermore, this component creates a *Task object*, which is applied to orchestrate the domain-specific components and manage their execution. The *Task object* is also used to dynamically create a form within the user interface to execute a parameterized instance of the processing task. After a configured processing object was established, the *Task Executer* creates and executes a start script, connecting to the corresponding nodes and executing the involved components, while mapping the specified parameters. For the possibility to receive notifications and manage the running components, the *Task object* has to be registered by the *Task Monitor*. The *Configuration Layer* also provides functionalities to dynamically scale the cluster by adding exclusive nodes for each processing task.

IV. A GENERIC PROVISIONER ARCHITECTURE FOR SCHEDULING DATA ANALYSIS ENVIRONMENTS

Using Cloud Management System (CMS) or hypervisor technologies, predefined cloud images enable different isolated processing platforms to be used by a data analyst. Instantiating these non-universal VMs requires an initial configuration with the to be processed data source. These instances implement the proper data processing tasks, using adjusted software, system packages on top of an underlying operating system. A domain analysis provisioner framework using a web dashboard is proposed that accumulates different cloud images with their corresponding data sources and establishes a self-service for users to deploy a personalized data analysis environment.

The raw data of any data source always contains meta data, which has to be translated into meaningful attributes to enable the users to identify and select the desired dataset. Executing an action on these datasets results in a VM containing the required data source and a pre-configuration of analysis or processing tools. Once a validation of data accessibility from the instances point of view is confirmed, the processing tool is executed automatically. Additionally, this proposed configuration enhances usability as well as the convenience level.

A. Architecture Overview

As depicted in Figure 3, the domain analysis provisioner framework architecture consists of multiple dynamic modules, which are partially generated by model driven software development. To separate these structures, the provisioning engineer divides involved subsystems into different domain scopes. The domain analysis provisioner framework scope contains modules regarding the definition of tasks and must adjust itself to the referenced data source. A platform image is always linked to a data source and is responsible for executing the intentions of the domain analysis provisioner framework's scope. Resource provisioning is the central execution of processing platforms and rests upon CMS interaction. Each component may be implemented in a specialized way, maximizing domain specific requirements realization. A *User Interface* displays all data sources and possible actions on different scopes while taking care of accountability requirements. Main focus of the *Task Definition Module* is the connection to the incoming data source technology, using a representation of emerging meta data. A defined task can consist of multiple data sources, requiring a separate connector implementing a `connectToData`, as well as a `getData` function. Using these functions as alternative views of remote data sources supports the engineer of such a system in case metadata is altered frequently. The *Task Management Module* defines basic mechanisms for instantiating and configuring platform images with data sources. For example, *OpenStack* and Amazons Elastic Compute Cloud (EC2) both provides a *user_data* parameter for popular custom commands and system calls executed at instantiation time of an image [11] [12]. With the definition of startup templates, realizing individual and task specific platform preparation is programmatically extended by dynamic metadata of the to be analyzed content. A processing platform is defined by the *Platform Image Definition Module* and must be registered with a CMS. Depending on the deployed software ecosystem and its configuration capabilities, the proper dynamic initiation script provided by the *Platform Instantiation & Configuration* module containing values defined by the metadata representation structure is deployed. Assigning a data source is done by manipulating different configuration files inside the VM. Since the CMS is able to push commands directly to the instances, every other method of registering or gathering data for processing is conceivable. The image ID resulting from the registration process with a CMS is the provisioner's reference for creating platforms inside the virtualized infrastructure. Once a platform is instantiated, the VMs static metadata is persisted inside an internal data store for further usage by the dashboard to visualize processing jobs, as well as manipulations by the *Cloud Resource Management* module. For example, a program executed inside the cloud image produces a log file, whose content may be interpreted by a status server, which returns feedback to the dashboard via REpresentational State Transfer (REST) paths. Thereby, the VMs assigned IP address is consulted from the provisioner's internal data store, requesting its current state at a predefined port and path to display it inside the user interface.

 ISBN: 978-1-61208-631-6

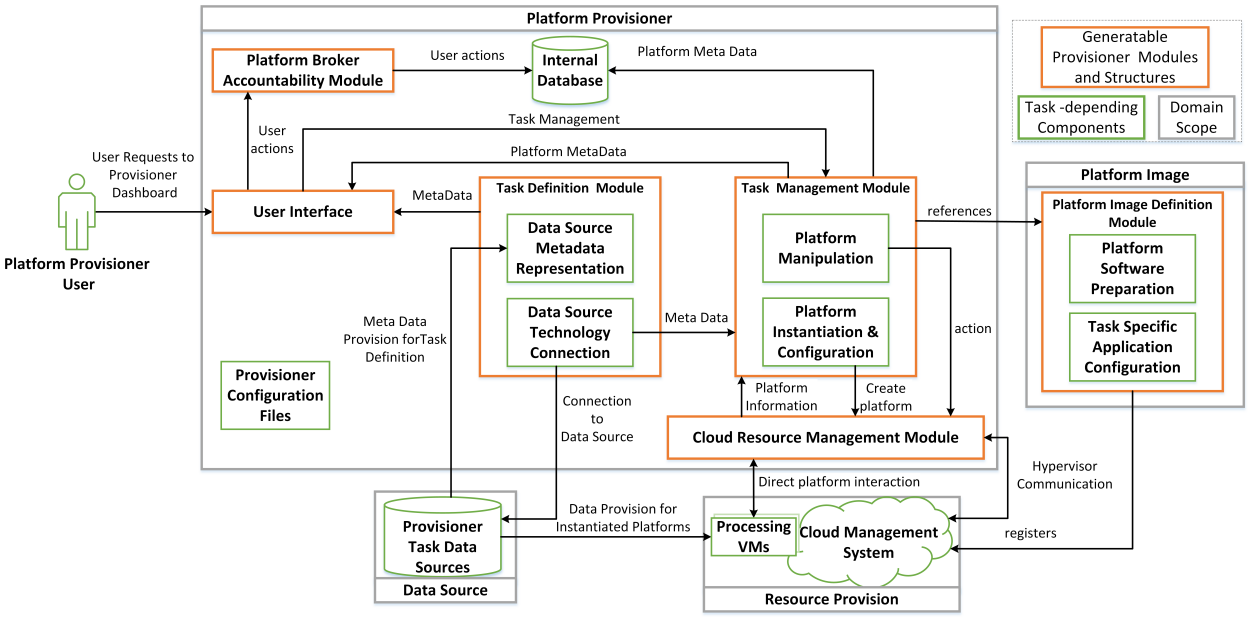

Figure 3. Modules of the generic domain analysis provisioner framework

B. *Language independent module implementation*

For a programming language independent implementation of the modules mentioned before, the Apache Thrift technology has been used. The definition of the modules, common data structures and provided interfaces is achieved using different Thrift Interface Definition Language (IDL) files that are used to compile it to a language specific implementation. The VMs status server may also be implemented as a Thrift service for transparent function calls using similar procedures as REST. Requirements, such as scalability, versioning or physical distribution of a provisioner are viable using Apache Thrift despite the given complexity of the technology when comparing it to language specific distribution technologies, like *OSGi* or native Remote Procedure Call (RPC).

C. *Provisioner module generation*

The previously described relations are rich in complexity and dependencies of modules among themselves. With the implementation of a specific application, based on the architecture shown in Figure 3, there are multiple interfaces and data connectors to define, impacting the cumulative provisioner procedure. With respect to the potential intertwined heterogeneous technologies of each sub-module, the necessary domain knowledge further increases the applications complexity. Therefore, a context-free grammar was engineered, enabling the declaration of these coherences as abstract syntax. A model to code transformation enables the translation of a universal domain analysis provisioner framework application model into technology specific skeletons. The creation of structures containing basic logic related to a specific technology supports the efficiency in provisioner task development. Using the *XText* environment, a *Platform*-Domain Specific Language (DSL) was developed, abstracting the architecture's main features and providing input for the code generator. The resulting structures must be enriched with individual module logic.

The *XTend* framework enables parsing previously generated grammar, as well as the utilization of code creation using templates. Implementing a custom template of the DSL, there are no restrictions for web frameworks or technologies respectively. Executing the required shell scripts, source code and technology dependent components the consequent results are placed in a central Dashboard folder. In case of Thrift usage there will be separate IDL files for each action and connector. Due to the freedom of choice regarding programming languages, a pre-filled skeleton for the service implementation containing common logic makes is not feasible. All client-side service calls are formulated inside the user interface structures and a script for creating proper cloud images is prepared. Path definition for processing applications on the local machine, as well as additional software packages, enables the creation of customized platforms.

V. Framework Evaluation

As elaborated in section I the main challenges for a big data analysis frameworks are reusability, scalability, extensibility and maintainability. These important characteristics are presented in the following section.

Reusability: Relies on the separation between general base framework and domain-specific functionalities. Developing and implementing the framework was done focusing on predefined functionalities while providing an intuitive way to integrate domain-specific components. With this approach and the use of common technologies to transfer, process and store large datasets, the framework is able to reduce the required developing time for a new processing task significantly.

Scalability: Processing large or high frequent datasets in a timely manner, requires high scalability. This is achieved by (i) using technologies, which are able to parallelize processing tasks over a large set of individual nodes and (ii) combining the involved components logically to create independent modules, which can be scaled according to the individual load of a processing task. Therefore, the framework is scalable by using multiple modules of the same type or by executing multiple

 ISBN: 978-1-61208-631-6

instances of specific components within a module.

Extensibility: Another important aspect of a generic framework is to provide a broad extensibility to include additional functionality or to integrate new technologies.Extensibility is needed to integrate additional functionalities or technologies to face the strongly changing requirements of modern Data applications. The framework has been designed carefully with the focus on defining generic interfaces between the layers and technologies. Therefore, replacing existing components, technologies or complete layers without effecting the remaining parts is possible.

Maintainability: Highly scalable big data frameworks require good component management to reduce the effort and costs.To mitigate this problem a centralized management interface that provides uniform functionalities to configure the nodes, technologies and processing tasks was introduced. For example, this includes adding new nodes, stopping processing tasks or centralized logging for debugging purposes.

Technology Usage for Provisioning: The different parts of the proposed domain analysis provisioner architecture can be implemented in nearly every suitable technology. A template for the provisioner core modules was implemented as a python based Django application. The creation of platform images is realized using *virt-builder*, resulting in *qcow2* images for further registration with different CMS environments. Alternatively, a snapshot of an existing and with the required software assembled VM can be used as platform image. This procedure results in an increased disk allocation and possibly additional network latency for deployment, which can be avoided using *CEPH*. Applying this distributed storage technology, a newly created snapshot only requires the difference between its current size and the size of a previously taken snapshot. Furthermore, this Copy On Write (COW) mechanism increases the speed of a VMs instantiation procedure [13]. The *Cloud Resource Management Module* is adjusted for this CMS. To demonstrate a language independent and distributed setup, all actions and connector modules are implemented as Thrift services. This way, the user interface consolidates the different remote procedures as central contact points, allowing changeability of connectors and action implementations at runtime. The extension of an already active domain analysis provisioner framework with additional tasks may depend on the chosen technology like Django.

VI. Conclusion and Future Work

In this paper we introduced a reusable cloud-based big data framework. The defined preprocessing framework handles communication and transfer of data by providing domain independent and easy adaptable interfaces. This underlying structure stores and processes data. With the help of the configuration layer new analysis tasks for incoming data can be created. Using the application layer, machine-learning tools or other applications can be used in standalone mode or the domain analysis provisioner framework can be deployed inside this layer. The introduced domain analysis provisioner framework provides dynamic modules generated by model-driven software development. The platform gives a ready-to-go definition for platform and resource management while providing interfaces to cloud systems. The domain analysis provisioner framework enables the platform engineer to define a domain specific platform by creating an analysis environment with all required tools and direct access to the stored and preprocessed datasets.

As a next step, we will define data quality properties for the data collection process and implement a domain specific data validation chain using rules and neural-networks to determine the quality of collected datasets in critical environments. Benchmarking and improving scalability and security of the proposed platform, as well as developing machine-learning modules to simplify configuration for preprocessing, are another aspect that will be implemented in the future.

Acknowledgement

This work has received funding from INTERREG Upper Rhine (European Regional Development Fund) and the Ministries for Research of Baden-Wuerttemberg, Rheinland-Pfalz and from the Region Grand Est in the framework of the Science Offensive Upper Rhine.

References

[1] N. Marz and J. Warren, Big Data: Entwicklung und Programmierung von Systemen für große Datenmengen und Einsatz der Lambda-Architektur, ser. mitp Professional. MITP Verlags GmbH, 2016.

[2] D. Namiot and M. Sneps-Sneppe, "On internet of things programming models," in Distributed Computer and Communication Networks: 19th International Conference, DCCN 2016, Moscow, Russia, November 21-25, 2016, Revised Selected Papers. Springer International Publishing, 2016, pp. 13–24.

[3] "NIST special publication 1500-6 nist big data interoperability framework: Volume 6, reference architecture," January 2018. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1500-6.pdf

[4] F. Rahman, M. Slepian, and A. Mitra, "A novel big-data processing framwork for healthcare applications: Big-data-healthcare-in-a-box," in 2016 IEEE International Conference on Big Data (Big Data), Dec 2016, pp. 3548–3555.

[5] "Cask Data Application Platform (CDAP)," September 2014, [Online] http://customers.cask.co/rs/882-OYR-915/images/CDAP_101.pdf [retrieved: 03-2018].

[6] D. G. Roy, D. De, M. M. Alam, and S. Chattopadhyay, "Multi-cloud scenario based qos enhancing virtual resource brokering," in 2016 3rd International Conference on Recent Advances in Information Technology (RAIT), March 2016, pp. 576–581.

[7] H. Kim, Y. Ha, Y. Kim, K.-N. Joo, and C.-H. Youn, "A vm reservation-based cloud service broker and its performance evaluation." in CloudComp, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, V. C. M. Leung, R. X. Lai, M. Chen, and J. Wan, Eds., vol. 142. Springer, 2014, pp. 43–52, [Online] http://dblp.uni-trier.de/db/conf/cloudcomp/cloudcomp2014.html [retrieved: 03-2018].

[8] E. Pacini, C. Mateos, and C. G. Garino, "Broker scheduler based on aco for federated cloud-based scientific experiments," in 2016 IEEE Biennial Congress of Argentina (ARGENCON), June 2016, pp. 1–7.

[9] "Cloud security alliance (CSA) big data taxonomy," September 2014, [Online] https://cloudsecurityalliance.org/research/big-data/ [retrieved: 03-2018].

[10] "Nist special publication 1500-3 nist big data interoperability framework: Volume 3, use cases and general requiements," January 2018. [Online]. Available: http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1500-6.pdf

[11] O. Khedher, Mastering OpenStack. Packt Publishing, 2015, ISBN: 9781784395643.

[12] "Amazon EC2 User Guide," 2017, [Online] http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html/ [accessed: 2018-03-16].

[13] W. Kong and Y. Luo, "Multi-level image software assembly technology based on openstack and ceph," in 2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference, May 2016, pp. 307–310.

 ISBN: 978-1-61208-631-6