DOI 10.12694/scpe.v17i4.1203
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 17, Number 4, pp. 313–321. http://www.scpe.org © 2016 SCPE

ARCHITECTURE OF A SCALABLE PLATFORM FOR MONITORING MULTIPLE BIG DATA FRAMEWORKS

GABRIEL IUHASZ,[*] DANIEL POP,[†] AND IOAN DRĂGAN[‡]

Abstract. Latest advances in information technology and the widespread growth in different areas are producing large amounts of data. Consequently, in the past decade a large number of distributed platforms for storing and processing large datasets have been proposed. Whether in development or in production, monitoring applications running on these platforms is not an easy task and dedicated tools and platforms were proposed for this scope. In this paper we present a distributed, scalable, highly available platform able to collect, store, query and process monitoring data obtained from multiple Big Data frameworks. We present its architecture and initial results obtained.

Key words: Big Data, monitoring, scalability

AMS subject classifications. 68M14, 62-07

1. Introduction. Big Data technologies have become a more present topic in both academia and industrial worlds. These technologies enable businesses to extract valuable insight from their available data, hence a large number of SMEs are showing increasing interest in using these types of technologies. Distributed frameworks for processing large amounts of data, such as Apache Hadoop [1], Spark [2] [18], or Storm [3] gained in popularity and applications developed on top of them are widely accepted [15]. However, developing software that meets the high-quality standards expected for business-critical Cloud applications remains a challenge for SMEs. In this case model-driven development (MDD) paradigm and popular standards such as UML, MARTE, TOSCA hold strong promises to tackle this challenge [4].

During the development of Big Data applications it is important to monitor the performance of each version of the application, so that software architects and developers can track how their application evolves over time. It is also useful in determining the main factors that impact the quality of the different application versions [3]. Throughout the development stage, running applications tend to be more verbose in terms of logged information so that developers can get information they need, hence data-intensive applications produce large amounts of monitoring data, which need to be collected, pre-processed, stored and made available for high-level queries and visualization.

This paper introduces, for the first time, a scalable, highly available and easy deployable platform for monitoring multiple Big Data frameworks. It currently integrates resource-level metrics, such as CPU, memory, disk or network, together with framework level metrics collected from Apache HDFS, YARN, Spark and Storm. The platform is easily extensible to other Big Data frameworks, or NoSQL / relational database systems.

The paper is structured as follows: next section introduces the design drivers for our platform, presents the overall architecture of the distributed monitoring platform and details its services. Section 3 presents initial validation results obtained against Apache Hadoop, Storm and Cloudera's Oryx frameworks. Section 4 contrasts our approach to similar tools and frameworks. Finally, we conclude with a discussion of ongoing and future work in Section 5.

2. Platform Architecture. This section presents the architecture of the DICE Monitoring platform (DMon). The platform draws its name from the EC-funded DICE project [4][4] The monitoring platforms provides both historical and near real-time performance data for other tools of the DICE ecosystem.

[*]West University of Timisoara and Institute e-Austria Timisoara Romania, (iuhasz.gabriel@e-uvt.ro).
[†]West University of Timisoara and Institute e-Austria Timisoara Romania, (daniel.pop@e-uvt.ro).
[‡]"Victor Babes" University of Medicine and Pharmacy and Institute e-Austria Timisoara Romania, (idragan@ieat.ro).

[1]http://hadoop.apache.org/
[2]http://spark.apache.org/
[3]http://storm.apache.org/
[4]Developing Data-Intensive Cloud Applications with Iterative Quality Enhancements (DICE) aims at providing a toolchain that makes the task of developing Big Data applications less daunting. It provides a set of artifacts (tools, models, methodology) that support software engineers in the process of designing and tuning data intensive applications.

In a nutshell, DMon is designed as a web service that enables the deployment and management of several subcomponents. Each of the subcomponents are responsible for enabling the monitoring of Big Data applications and frameworks. In contrast to other monitoring solutions [1, 2], DMon aims at providing the user with as much data as possible about the current status for the Big Data subcomponents. By doing so a wide range of new technical challenges arise. Due to the fact that DMon is serving near real-time fine grained metrics it must exhibit high availability and easy scalability.

DMon provides a distributed, high availability monitoring service. It is tailor made for Big Data technologies and is easily extendible to collect metrics from a wide range of platforms. Big data service integration/monitoring into monitoring platforms is still an open issue. The ingestion of large amounts of data in a timely manner is also an open question. During production only warning or error level logs and metrics are important. However, during development this level of detail is not sufficient. Consumption and most importantly presentation in a useful manner of collected metrics in near real time represents one of the key problems addressed by our approach. This is the main rationale behind the new distributed monitoring solution for Big Data technologies presented here.

Traditionally, web services have been build using a monolithic architecture where almost all components of a system run in a single process, usually a Java Virtual Machine (JVM). In case of this architecture type there are major advantages when it comes to deployment and networking. On the other hand scaling of such a system is a highly non-trivial task which requires running several instances of the service behind a load-balancer instance.

Although the monolithic approach seems the way to go when deploying monitoring platforms, there are some severe limitations to it. Changes to one component can have an unforeseen impact on seemingly unrelated area of the application. Thus, adding new features to the platform can be potentially really expensive both in terms of time and resources. Secondly, individual components cannot be deployed independently. That is, partial functionality cannot be achieved in those systems. Using such solutions it is likely to partially loose reusability of the entire platform. This issue can be addressed at design time by creating reusable components. In practice we observe that it is not always the case that reusability is a major concern for developers, but rather the focus is on readability of code resulting in some of the cases in side effects like loss of performance.

Taking into consideration both the pluses and the limitations exposed by a monolithic design for a monitoring platform, we decided to use another fundamentally different approach for DMon. That is, we are deploying a widely used approach in the Internet companies [6], the so called **microservice architecture** [13]. By using this architecture we replace the monolithic service with a distributed system of lightweight services. Each of the services are by design independent and narrowly focused. This approach allows us to deploy, upgrade and scale individual services rather than entire monolithic components. Because of the loosely coupled manner of microservices, code reusability is much easier and changes made to individual services do not necessarily require changes in other ones. In the microservice architecture, integration and communication between services should be done either by using HTTP (REST APIs) or using RPC requests. The use of microservice architecture allows as to group related behaviours into separate services, thus enabling us to easily modify the overall system without the hustle for editing of multiple services.

DMon uses REST APIs for communication between different services with request payload encoded as JSON messages. This makes the creation of synchronous or asynchronous messages much easier.

Figure 2.1 shows the overall architecture of the DMon platform. The context in which DMon is developed is relying on the so called lambda architecture [11] . As defined the lambda architecture consists of three layers: *speed*, *batch* and *serving layer*. In order to follow this guidelines DMon is using Elasticsearch as serving layer responsible for loading batch views of the collected monitoring data and enabling other tools/layers to do random reads on it. The speed layer will be used to look at recent data and represent it in a query function. In the case of anomaly detection, it will require the use of unsupervised learning techniques or pre-trained models from the batch layer. The batch layer needs to compute arbitrary functions on large sections of the dataset stored in Elasticsearch. That is, running long-running jobs to train predictive models that than can be instantiated on the speed layer. All trained models will then be stored inside the serving layer and accessed via DMon queries.

Core components of the platform are Elasticsearch, for storing and indexing of collected data, and Logstash

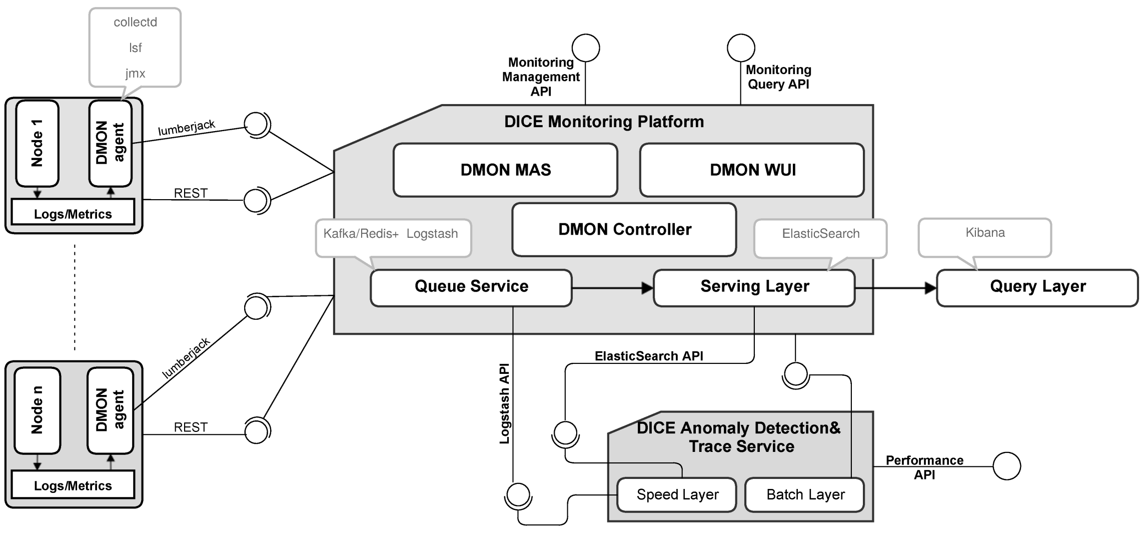

Fig. 2.1. *DMon architecture*

for gathering and processing logfile data. Kibana server provides a user-friendly graphical user interface. The main services composing DMon are the following: *dmon-controller, dmon-agent, dmon-shipper, dmon-indexer, dmon-wui* and *dmon-mas*. Each of these services will be used to control both the *core* and *node-level components*.

2.1. Core components. As the name suggests the *core components* are the backbone of the entire monitoring platform. They are used for collecting, processing, aggregating and transforming all the incoming monitoring data. One of the most essential features that is common for these components is ease of configuration, scalability and support for high throughput.

Elasticsearch [7] is an open-source, RESTful search engine based on top of Apache Lucene [12]. It is an inherently (horizontally) scalable solution which can perform near real-time processing with up to five-second latency. It also provides support for multi-tenancy, streamlined backup procedures as well as insuring data integrity. One of the most important capabilities of Elasticsearch is its ability to handle high throughput of tens or even hundred thousands of messages per second.

Logstash [17] is a tool developed in order to collect, process and forward events and log messages. Basically, it handles **E**xtract, **T**ransform and **L**oad (**ETL**) operations. It uses configurable plugins for input-output and filters in order to collect, process and load data. The input plugins can be configured to accept a wide range of inputs starting from TCP/UDP to Kafka [10] topics. Input plugins send the data for processing to the filter worker plugins. Finally, the processed data is routed to one or more output plugins such as Elasticsearch, Kafka, InfluxDB etc. Logstasg has the important property of being essentially stateless thus making it extremely scalable. For example it is possible for two Logstash instances to serve the same Elasticsearch endpoint.

DMon does not provide its own metrics visualization widgets, rather it leverages on Kibana server to create customizable dashboards for any number of metrics. Kibana [7] serves the role of browser based analytics and search interface for Elasticsearch. DMon platform includes support to graphically represent CPU, Memory and Network loads, as well as Big Data specific metrics in customizable Kibana dashboards. These visualizations are based on Elasticsearch queries and can be aggregated and plotted using a histogram based on their timestamps. Some of these visualizations are automatically created by dmon-controller service and saved into a dedicated index in Elasticsearch. It is possible to add additional visualizations manually to fit the needs of any developer.

All of the above mentioned components are part of the so called *Elasticsearch ELK* stack. This setup provides a very robust base for DMon and will be used as a proof-of-concept implementation.

2.2. Node-level components. DMon has to monitor a wide range of Big Data technologies each of which have different metrics and metrics systems. Because of these constraints we had to choose collectors that are flexible enough to accommodate a wide variety of technologies. Besides input restrictions the collectors must have a small computational footprint. By taking into account all the previous restrictions we limit the amount of "noise" produced by the presence of different collectors. Another critical feature that all the deployed collectors must obey is easiness of deployments. That is, it should be easy to deploy and configure them for thousands of physical and virtual machines.

Collectd [5] is an open-source POSIX daemon that collects, transfers and stores performance and network related data. Being a wide used tool, collectd provides the users with a great palette of options for collector plugins. This tool is used in DMon to collect system metrics, such as CPU utilization or memory / disk / networking load and throughput.

As previously presented the Logstash server is able to collect metrics and log files directly from the machine it is installed on. In this particular case it would mean that we need to have a Logstash instance on each of the monitored nodes. In the DICE context, this approach is not feasible especially because Logstash has a substantial computational footprint when using specialized filters such as grok[6]. Instead, we decided to use logstash-forwarder [7] to do the job of metrics forwarding. logstash-forwarder is designed for the purpose of log forwarding to one or more logstash server instances. By using this approach inside DMon we are eliminating node-level side effects caused by local processing of logs.

At this point it is important to note that there are several alternatives to logstash-forwarder and even collectd. Most notably there are the *Beats* [8] data shippers for Elasticsearch. Although Beats represents today an interesting alternative, this solution was not available at the design and implementation time of the DMon prototype.

Since most of Big Data frameworks are Java tools, we can use Java Management Extensions (JMX) to extract valuable metrics related to the JVM. In fact, a large number of Big Data frameworks already support exporting metrics via JMX. Thus, jmxtrans [9] tool is used in our architecture to collect attributes exported at JVM level. It's worth mentioning that both the core and node-level components of DMon may not be final and other solutions might be integrated. For example, it is possible to use rsyslog [10] instead of Logstash to process and load data into Elasticsearch. There are alternatives to Elasticsearch as well, such as NoSQL databases that support handling of time series data, such as InfluxDB [11] which is an emerging technology. Of particular interest is the collection of JMX metrics using collectd via a plugin. Although there are some available collectd plugins [12] that are able to accomplish this task, they have proven to be either slow or very resource hungry in preliminary tests.

2.3. Platform services. All components described in the previous sections have to be deployed and configured. The steps for configuration and deployment are accomplished by a number of Web services wrapping the core and the node-level components. The rest of this section details each of the wrapping services.

2.3.1. Core-level services. There are in total three core services: *dmon-controller*, *dmon-shipper* and *dmon-indexer*. All have been implemented using the Python programming language. Specifically with the Flask microframework [8]. The interface used by the services to communicate with each other takes the form of JSON encoded messages.

The **dmon-controller** service is essentially the service with which all other components communicate. It is in fact the main point of integration with the rest of the DICE solution. In particular, it will be used by all DICE components that require monitoring data. The REST API is split into two main parts: Monitoring Management API and Monitoring Query API. A swagger [13] based web UI is used for all developed services for

[5]`https://collectd.org/`
[6]`https://www.elastic.co/guide/en/logstash/current/plugins-filters-grok.html`
[7]`https://github.com/elastic/logstash-forwarder`
[8]`https://www.elastic.co/products/beats`
[9]`http://www.jmxtrans.org/`
[10]`http://www.rsyslog.com/`
[11]`https://influxdata.com/`
[12]`https://collectd.org/wiki/index.php/Plugin:GenericJMX`
[13]`http://swagger.io/`

```
{
  ''DMON'':{
    ''query'':{
      ''size'':''<SIZEinINT>'',
      ''index'':''<indexname>'',
      ''ordering'':''<asc|desc>'',
      ''queryString'':''<query>'',
      ''tstart'':''<startDate>'',
      ''tstop'':''<stopDate>''
    }
  }
}
```

Fig. 2.2. *Example of query (JSON format)*

easy of use and a form of interactive documentation of all REST APIs.

The *Monitoring Management API*, namely *Overlord*, is used to register nodes, change configuration parameters and current status of all node-level components. It can also be used to deploy and configure node-level metrics on to registered nodes. Because of this, when registering nodes it is required that credentials for each node be supplied (username, password or key). If node-level components and services have already been deployed by other tools they only have to register the already deployed node-level service endpoints. In this scenario credentials are not needed.

Long term storage of metrics is of course a problem that has to be dealt with in all data warehousing solutions. In the case of DMon one may use the management API to create new indexes which store monitoring data. We can also export these indexes or even dump the entire dataset into a different format. By default, DMon creates an index every 24 hours. These indexes can be queried either on at a time or all at once. The exported indexes can be at any given time be reloaded into DMon or into a different Elasticsearch deployment for offline processing.

Metrics version annotation is also supported by the dmon-controller. By this we mean that metrics pertaining to a specific application version can be annotated using tags. This way we can easily query, aggregate or even compare metrics of the application. Creation of a separate index for each application version is also possible. However, it is not as versatile a solution and makes comparison of different application version performance more difficult. The dmon-controller is also responsible for generating and enacting configurations for all core components (Elasticsearch, Logstash and Kibana). The configuration is largely dependent on the data provided during the registration of each node. This data is then used to configure each component of DMon.

As already mentioned, the type of node-level component needed for monitoring is based on the Big Data service that run on each machine. During registration a list of services that are deployed on each node can be defined which are then used to setup and manage node-level services and components. Querying DMon is done using the *Monitoring Query API*, namely *Observer*. In contrast to the Management API, this one doesn't require authentication. A query example is included in Figure 2.2, showing attributes that can be specified: the size of the returned response, its ordering, or start and stop dates (in UTC). The *queryString*, which follows the same format and rules as Kibana queries, actually defines the predicate to be run on the Elasticsearch and can be used to aggregate data, or perform additional operations on the stored data [7]. Query's response may be returned in several formats, currently supported are: CSV, JSON, or plain text. Support for RDF+XML encoding using OSLC Perf. Mon 2.0 vocabulary [9] was also developed but only system metrics are curently exportable using this format. The CSV and RDF+XML query responses are generated using the dmon-controller service, which takes the JSON response from ElastiSearch and converts it to the target format.

The **dmon-shipper** microservice is meant to deploy, manage and configure logstash instances. In contrast to dmon-controller service, this service has to be located on the same machine with the controller logstash instance is located. This service is not meant to be used by external tools and services, being an internal component of DMon platform. The **dmon-indexer** microservice is used to control nodes comprising the Elasticsearch cluster.

Splitting the control of various core and node-level components into microservices we can easily separate the application logic of DMon from the code that actually drives and enacts them. Another important point is that all services besides the dmon-controller are essentially stateless. For example neither service stores the current state of the components it controls, rather it has to poll the status of the component. The dmon-controller stores some basic state and node-level information inside a relational database, which can be exported, imported, versioned or even backed up.

2.3.2. Auxiliary platform services. These are optional services that add support for scalability, availability and ease of usage of the platform. At the time of writing of this paper, these services are under development and they have not been validated yet.

Queuing service. In some instances of DMon platform, metrics that are sent by the node-level components might exceed the capabilities of Logstash to process them effectively. This might lead to data loss. There are ways to mitigate this problem. First, we could increase the number of workers assigned to the filter plugin. In the Logstash documentation it is specified that some filters (specifically grok) might cause slowdown in metrics ingestion. Second, if increasing the number of workers is not an option we could create a second instance of Logstash which can handle some of the load. The exact way Logstash would behave in these scenarios is still under investigation since Logstash and elasticsearch configurations are automatically generated and controlled by DMon. Because of this future work will involve creating performance models of all core components.

The third variant is to use a queuing service that receives all metrics and from which the Logstash instances can consume data. Certainly, this will mitigate the data loss problem but could potentially increase the time it takes for a specific metrics reading to be processed and indexed inside elasticsearch. This service is pictured in Figure 2.1, possible candidates for its implementation ranging from Kafka or Redis to a combination of MongoDB [5] and RabbitMQ [16]. The full technical stack is still an open question and will be addressed in future versions of DMon.

DMon's **dmon-wui** is a user-friendly web user interface that gives end-users an easy access to management operations of the platform. It will also include an overview of the metrics collected from the current Big Data deployment. The dmon-controller service is able to generate a dashboard definition file for Kibana [7] engine containing both system metrics (CPU, memory, network etc.) and the most popular metrics for supported Big Data frameworks.

Automatic scaling and management of the DMon platform is going to be possible using the envisioned **dmon-mas** service (DMon multi-agent service). The service is based on a multi-agent system architecture with scaling and management capabilities in mind. For achieving these goals it relies on monitoring data for the currently deployed monitoring solution so that scaling of various components and services are going to happen based on the platforms performance. By using this approach we intend in improving overall performance and resilience to the system by enhancing the DMon platform with self-healing capabilities.

Being a multi-agent system, *dmon-mas*, is going to use a variety of specialized agents. Some of the agents are going to be in charge of monitoring the current deployment, while others will be in charge of reasoning on the gathered data. Besides these two types of agents another category will comprise agents responsible of enacting the changes imposed by the performance analysis agents. Provisioning agents will be added to the platform in order to facilitate provisioning of new Virtual Machines (VMs) on a wide variety of cloud platforms like Flexiant Cloud Orchestrator (FCO), Amazon or OpenStack.

Most of the configuration and management task needed will be accomplished using the dmon-controller service. Due to the clear separation of roles between individual components, we mention that the dmon-mas service is not being part of the core services. Hence it is not mandatory, but rather recommended in order to improve performance, to start the dmon-mas in order to obtain correct functionality of the overall DMon. Currently the dmon-mas service is under investigation and a concrete working solution for the service is being implemented.

2.3.3. Node-level Services. The **dmon-agent** service is used to manage and configure all node-level components. Similarly to the dmon-shipper and dmon-indexer services, it is also stateless. The dmon-controller service issues request to each dmon-agent service with a JSON payload that contains all required information for controlling the node-level components. Each monitored node has to have a dmon-agent instance running on it.

As of writing this paper, the dmon-agent supports collectd, logstash-forwarder and jmxtrans as metrics collecting components. It is able to install all of these supported components. The installation is based on the type of big data services and the roles assigned to the node where the dmon-agent service is installed. The dmon-controller is not responsible for picking what component each dmon-agent is deploying and managing, it only sends the list of roles each node has. It is also able to interact with Hadoop, Spark and Storm deployments. This interaction is required to change or activate the metrics system for the supported big data services.

3. Initial Validation.

This section details the Cloud deployments and reports on initial validation results of the platform against Oryx2 and Storm frameworks.

3.1. Cloud deployment.

In order to validate the deployment of the platform in Cloud environments, we selected two setups: one public Cloud provider (namely Flexiant Cloud Orchestrator) and one private, OpenStack powered, Cloud environment hosted by West University of Timisoara. The deployment on Flexiant Cloud Orchestrator used one VM with 4 vCPUs, 8 GB RAM and 250GB hard disk.

The current version of the monitoring platforms installation procedure requires the use of *aptitude*[14] package management system present in Debian based Linux distributions. However, it can run on any POSIX compliant operating system as long as all dependencies have been installed manually.

For future development we are considering *Chef* recipes for deployment.

3.2. Deployment using Vagrant.

For development purposes, Vagrant [14] deployment scripts for DMon, CDH and Storm were also created. The first script installs and configure a distribution of DMon where all core components and services are collocated on the same VM. The second vagrant script provisions 4 VMs on which it installs the newest version of the CDH together with a version of Oryx 2 toy application. The usage of all Vagrant scripts documentation can be found in the Github repository [15]. Using these scripts anyone can create a standard development/demo environment, which contains not only the latest version of DMon. All VMs are provisioned with 2 CPUs, 4 GB RAM and a HDD of 50 GB.

3.3. DMon validation against Oryx2 and Storm.

In order to validate the platform against state-of-the-art Big Data technologies, we have deployed Cloudera Distribution for Hadoop (CDH) 5.4.7 and Oryx 2 on a cluster of 14 nodes on FCO as well. All VMs have the same specifications as the monitoring VM described in section 3.1. For this first round of validation we assume a worse case scenario in the sense that all DMon services and core components are collocated on the same VM sharing a common resource pool. Although, tests were done on a fully distributed deployment the majority of the testing and development was done on a single VM.

It is important to note that Oryx 2 is treated in this setup as a collection of different Big Data services not as a Big Data service. This means that Oryx2 metrics are comprised of metrics from HDFS, Yarn, Spark and Kafka (Kafka monitoring is still in early development). No Oryx 2 specific metrics are monitored.

Each time a new node is added to the DMon platform for monitoring, the platform automatically installs the monitoring agent on it. The monitoring agents collect the data from the local files and sends the data to DMon. Currently, metrics for the following technologies are collected: Apache YARN and Apache Spark [18]. System metrics (CPU, memory, network, disk, etc.) are also collected using collectd plugins. These technologies are used by Oryx 2 for both the batch and speed layer. All metrics are sent to a logstash server instance which uses the custom filters generated by DMon to transform and then load the metrics into the elasticsearch instance. During a one hour period DMon was able to collect extract the monitoring data and index it into elastic search over 337,200 events of which more than half are Big Data service specific. All metrics have been collected at a 10 second interval. Each event contains information extracted from the Big Data metrics systems and can contain as much as 20 different key value pairs.

Another set of empirical validation has been run on a smaller deployment for Apache Storm. This deployment consisted on 4 VMs with the same setup as before. A demo topology was loaded consisting of 1 spout and two bolts. DMon was able to automatically detect the running topology and dynamically adjust the mapping of pertinent metrics to each bolt and spout. The automatic topology detection is accomplished via the scanning

[14]https://wiki.debian.org/Aptitude

[15]https://github.com/dice-project/DICE-Monitoring/blob/master/src/

TABLE 4.1
Comparison of tools

	Nagios	Ganglia	SequenceIQ	Apache Chukwa	Sematex	DataDog	DMon
Scalability	Manual	Manual	-	Manual	Yes	-	Self-scaling
Elasticity	None	None	-	None	Yes	-	Yes
Deployment Model	VM	On-premise	Service	On-premise	On-premise Service	Service	On-premise Service
Installation	-	Manual	-	Manual	-	-	Service via REST API
Big Data frameworks support	Poor	Poor	Hadoop 2.x	Hadoop 2.x	Good	Good	Good and Extensible
Visualization	User Defined	Predefined	Predefined	Predefined	User-defined	User-defined	User-defined
Analytics	Alerts	-	ML Support	Anomaly detection	Alerts	Alerts, correlation	Anomaly Detection
Real-time data support	Yes	Yes	Yes	No	Yes	Yes	Yes
Licensing	Freemium	BSD	Commercial	Apache 2	Freemium	Freemium	Apache 2

for the Storm REST API. Once this is found the description of the current topology is used for the automatic mapping. In a one hour timespan DMon collected over 84,600 events.

4. Similar tools. The section presents a brief overview of different monitoring solutions that can be applied to Big Data frameworks. Some of most popular open-source and commercial solutions are contrasted to DMon on different dimensions in table 4.1.

In the context of monitoring tools, scalability is key as Big Data deployments may include thousands of nodes. Although the selected technologies (ELK stack) easily support horizontal scalability, sometimes the throughout of generated monitoring data may exceed Logstash's processing capacity. In order to cope with this, a message queue should be employed 'in front' of Logstash server(s). In our case, Kafka provides a distributed backbone and data pipeline that enables the integration into the DICE monitoring.

Elasticity, the ability to adapt to data throughput is another key design driver for our platform. DMon multi-agent (dmon-mas) service provides up/down scaling of the platform based on observed data throughput.

In terms of deployment and installation approaches, platforms may be either installed manually or automatically deployed using specialized software infrastructure, namely content management systems. The reviewed platforms all require manual installation, whereas DMon provides scripts for node provision and configuration. These may be included in orchestration frameworks. The node components are transparently installed upon node addition by the DMon controller service, thus requiring no specialised skills nor personnel.

Extensibility of the platform, i.e. easy integration of new frameworks, was central to our design. The platform provides a uniform interface to a number of Big Data frameworks. Including support for a new Big Data frameworks requires proper configuration of nodes' roles and adaptation of Logstash parsers. In this way, not only Big Data frameworks can ingest data to our platform, but we can also collect log data produced by any custom data intensive application.

In most of reviewed platforms, analytics against collected monitoring data is handled via user-defined alerts. Although these provide valuable data for Ops teams, they do not provide the level of insight required by Dev teams for optimization and validation purposes. More sophisticated, contextualized methods and tools are required. The DICE Anomaly Detection component is able to detect such anomalies and with the help of the DICE Enhancement tools will feedback this information into design-time models.

5. Conclusions and Future Work. This paper presents the architecture and initial validation of the DICE Monitoring platform, which is a distributed, highly available platform for monitoring Big Data technologies. The goal of the initial version of the platform is to demonstrate a working Proof-of-Concept that collects, stores and processes monitoring data from multiple Big Data technologies, currently supported frameworks being Apache HDFS, YARN and Spark. Designed using a microservices architecture, the platform is easy to deploy, and operate on heterogeneous distributed Cloud environments. We reported successful deployments on Flexiant Cloud Orchestrator and OpenStack using Vagrant scripts.

We are planning to extend the platform to better support scalability and elasticity in Cloud environments (e.g., Queuing service, dmon multi-agent service) and to offer a more intuitive and user-friendly Web interface (e.g., metrics visualization UI, customizable dashboard). Research-wise we will be investigating the integration of the platform with design-time artifacts, such as UML modes, meta-models and profiles in order to allow software engineers and designers to define the required metrics in their models and then propagate them along the deployment pipeline down to DMon configuration files. Streamlining the deployment of the platform by integrating it with DevOps oriented tools, such as Cloudify and/or Chef, is going to be addressed in the future.

Acknowledgment. This work was partially funded by the European Union's Horizon 2020 Research and Innovation Programme through the DICE action (`http://www.dice-h2020.eu`) under Grant Agreement Number 643946.

REFERENCES

[1] G. Aceto, A. Botta, W. de Donato, and A. Pescapè. *Cloud monitoring: A survey.* Computer Networks, 57(9):2093–2115, 2013.

[2] K. Alhamazani, R. Ranjan, K. Mitra, F. A. Rabhi, P. P. Jayaraman, S. U. Khan, A. Guabtni, and V. Bhatnagar. *An overview of the commercial cloud monitoring tools: research dimensions, design issues, and state-of-the-art.* Computing, 97(4):357–377, 2015.

[3] L. E. Bautista Villalpando, A. April, and A. Abran. *Performance analysis model for big data applications in cloud computing.* Journal of Cloud Computing, 3(1):1–20, 2014.

[4] G. Casale, D. Ardagna, M. Artac, F. Barbier, E. D. Nitto, A. Henry, G. Iuhasz, C. Joubert, J. Merseguer, V. I. Munteanu, J. F. Perez, D. Petcu, M. Rossi, C. Sheridan, I. Spais, and D. Vladuic. *Dice: Quality-driven development of data-intensive cloud applications.* In 7th IEEE/ACM International Workshop on Modeling in Software Engineering, MiSE 2015, Florence, Italy, May 16-17, 2015, pages 78–83, 2015.

[5] K. Chodorow and M. Dirolf. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage.* O'Reilly Media, 2010.

[6] M. Fowler. *Microservice overview,* Guide. Available at http://martinfowler.com/microservices/, Jan. 2016.

[7] C. Gormley and Z. Tong. *Elasticsearch: The Definitive Guide.* O'Reilly Media, 2015.

[8] M. Grinberg. *Flask Web Development: Developing Web Applications with Python.* O'Reilly Media, Inc., 1st edition, 2014.

[9] S. Kennedy and L. Jiu. *Facilitating collaboration and interaction across the enterprise with oslc.* In Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research, CASCON '13, pages 374–375, Riverton, NJ, USA, 2013. IBM Corp.

[10] J. Kreps, N. Narkhede, and J. Rao. *Kafka: A distributed messaging system for log processing.* In Proceedings of 6th International Workshop on Networking Meets Databases (NetDB), Athens, Greece, 2011.

[11] N. Marz and J. Warren. *Big Data: Principles and Best Practices of Scalable Realtime Data Systems.* Manning Publications, 2015.

[12] M. McCandless, E. Hatcher, and O. Gospodnetić. *Lucene in Action.* Manning Pubs Co Series. Manning, 2010.

[13] S. Newman. *Building Microservices: Designing fine-grained Systems.* O'Reilly Media, Incorporated, 2015.

[14] M. Peacock. *Creating Development Environments with Vagrant.* Community experience distilled. Packt Publishing, 2013.

[15] D. Pop. *Machine learning and cloud computing: Survey of distributed and saas solutions.* Technical Report 2012-1, Institute e-Austria Timisoara, December 2012.

[16] G. Santomaggio and S. Boschi. *RabbitMQ cookbook.* Packt Publ., Birmingham, 2013.

[17] J. Turnbull. *The Logstash Book:.* James Turnbull, 2013.

[18] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. *Spark: Cluster computing with working sets.* In Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

Edited by: Viorel Negru
Received: June 15, 2016
Accepted: October 5, 2016