RaDEN: A Scalable and Efficient Radiation Data Engineering

Hadi Fadlallah
Lebanese University
Beirut, Lebanon
Hadi.Fadlullah@gmail.com

Yehia Taher
Université de Versailles Saint-Quentin -en-Yvelines (UVSQ)
Versailles, France
yehia.taher@uvsq.fr

Ali Jaber
Lebanese University
Beirut, Lebanon
ali.jaber@ul.edu.lb

***Abstract*— Detecting and monitoring radiation level is one of the critical duties for governments and researchers because of the high threats it oppose to humans. It was challenging in the past century to have a centralized radiation monitoring system until the rise of IoT (Internet of Things). Radiation level is measured using wireless sensors that outputs data which are transferred to a back-end server that monitors radiation and alerts when high radiation levels are detected, the server also stores the data for further analysis. The traditional data warehousing systems cannot handle this type of data any more due to (1) data collection speed, (2) rapid data growth, and (3) data diversity. With the rise of Big Data notion, new technologies are developed to handle data with similar characteristics. In this paper, we proposed RaDEn a scalable and fault-tolerant radiation data engineering system that relies on Big Data technologies such as Hadoop, Kafka, Spark, and Hive. The system is responsible of (1) reading data from sensors and other sources, (2) monitor the radiation level in real-time, (3) storing the data, and (4) providing on-demand data retrieval to users. In addition, we have implemented our system and conducted experiments in a real case scenario in collaboration with the department of environmental radiation control at the Lebanese Atomic Energy Commission (LAEC-CNRS).**

Keywords—Radiation, data engineering, Big Data, radiation monitoring, real-time processing

I. INTRODUCTION

Radiation pollution is a critical concern due to its detrimental impact on living beings and environment. There are different types of radiation stemming from various radioactive materials and natural resources [1]. The higher level of these radiations specifically the gamma radiation causes severe damage to human health [2]. Therefore, controlling radiation level is critically important. In order to do so, monitoring radiation sources is an indispensable task.

The advent IoT (Internet of Things) specifically, sensors have paved the foundation of building smart ecosystems that enable collecting radiation data, processing, and analyzing radiation level in real-time [3]. Radiation sensors collect and transmit data via communication network such as telecommunication network, Wi-Fi, and Internet to the computational engine for measuring radiation levels.

Radiation monitoring sensors records data continuously; in consequence, massive volume data can be generated in a high speed. Conventional data engineering technologies such as data warehouse are not adequate to handle this type of data.

Several data engineering technologies have been proposed in literature such as [5],[6],[7],[8],[9],[10],[11],[12],[13],[14],[15],[16] and many others. These solutions aims engineering radiation pollution data. However, existing solutions have several limitations that we summarized as follows: (1) Existing technologies rely mainly on traditional data technologies. (2) Most of them are focused on the data collection only. (3) Real-time data collection and processing is outside of the scope of existing technologies. (4) Scalability and fault-tolerance have not been dealt with by the technologies discussed in the previous sections. A solution that can address these limitations is an indispensable need.

In this paper, we have proposed a solution called RaDEn, which is a scalable and fault-tolerant system for radiation data engineering that relies mainly on new data technologies that are able to handle massive volume of data generated in high speed. RaDEn has the ability to read data from different sources, monitor radiation level in real-time, storing data in a scalable repository that provides on-demand data retrieval to users for further analysis.

The remainder of this paper is organized as follows. In Section 2, we briefly introduce our solution called RaDEn. The development of RaDEn will be detailed in Section 3. Section 4 demonstrates RaDEn. We conclude our work in Section 5.

II. AN OVERVIEW OF RADEN

RaDEn is a scalable platform developed for radiation data engineering. It allows fetching massive volume of data from different sources. RaDEn enables user collection different types of data including such as structured databases, data streams and flat files. RaDEn has a radiation data lake which stores data a scalable cluster, process then with advanced techniques and visualize data using the best fit methods.

RaDEn adopted both realtime and batch style philosophies for collecting and processing data. The hybrid enables users to perform both realtime and batch style operations. The data streaming from sensors can be collected by the users in realtime and files can be ingested in storage as batch style. The processing and can be done the same ways. Besides these major operations it somewhat performs pre-processing tasks such data transformation and loading data into scalable data lake. Visualization is real-time meaning that the streams can be visualized with minimal latency and the can be done by files data.

RaDEn is built on cluster computing and parallel computing paradigm. In addition, it adopts the notion of Big Data. The cluster computing guides the solution adopt technologies that foster scalability whereas the parallel computing provides computation models for designing parallel operations using suitable programming model such as functional programming model.

RaDEn is built-on multi-layered architecture. Figure 1 shows the architecture of RaDEn. The figure shows that the RaDEn system can collect data from any number of sources. RaDEn consist of six layers which are explained in the following:

- ***Data Sources:*** Data sources layer consists of data streams sent from sensors installed in cities and mountains, relational databases where archive data are stored and flat files that can be exported from any old echo system

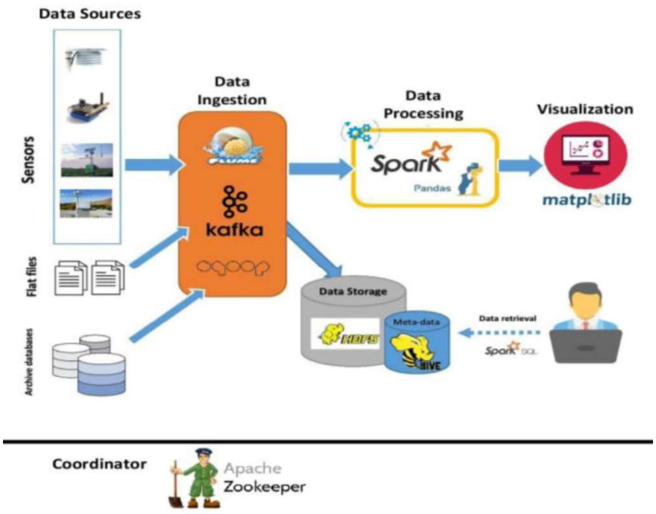


Figure 1 - RaDEn architecture

- ***Data ingestion layer:*** This layer is responsible of reading data from different sources and delivering them to data processing or data storage layer. This layer must ensure scalability and fault tolerance and must read a huge amount of data from different sources in real-time and batch mode. Data can be stored into Data storage layer directly or it can be sent to Data processing layer to be processed in real-time.

- ***Data Storage:*** This layer is responsible of storing data. It relies heavily on HDFS, which is a distributed file system that ensures high scalability and fault-tolerance. On the Top of HDFS we have to use warehousing technology to define and configure the metadata of the data stored in HDFS and let the users be able to perform easy data retrieval operations.

- ***Data Processing Layer:*** This layer is responsible of processing data and notifying the end-user when a high level radiation is detected. The nature of the data sources requires a distributed data processing platform that ensures a high scalability and fault tolerance.

- ***Data visualization:*** This layer is to visualize the results of data processing and is responsible of bringing the end-user in action by drawing real-time graphs that show radiation level timeline.

- ***Coordination layer:*** This layer is responsible of making all used technologies able to communicate with each other's. It is a service that runs in background and has the ability to connect with any technology used.

III. Development of RaDEn

RaDEn is developed in two phases. In the first phase, we developed the RaDEn system and in the second phase we developed an alarm system integrated within RaDEn.

A. RaDEn Core System

We have built a 4-node Hadoop cluster. To build this system we have first deployed four virtual machines where we have installed Ubuntu[1] 16.04 LTS as operating system and Hadoop 3.1.0 for data storage. We have configured the first virtual machine to act as the master node (name node) and the others to act as slaves (only stores data).

On the master node, we have also installed the data ingestion, processing and visualization tools. As a programming language, We have used python[2] because it is more powerful than other languages in data science domain due to the presence of many specialized libraries.

For data ingestion, we have installed Apache Kafka[3], Apache Flume[4], and Apache Sqoop[5]. We used Apache Kafka as the main data ingestion tools, because: (1) it has the ability to read data from sensors directly. (2) It guarantees scalability and fault tolerance. (3) It can read data in real-time and at rest. (4) It can send the data to processing engine and to the data storage layer. (5) It is easy to implement using python programming language. To use Apache Kafka, first we have installed the Apache Zookeeper[6] which acts as a coordinator that lets Apache Kafka communicate with other technologies. Then, we created two topics: (1) "readRadiationData" which will be used only to insert data to HDFS (Hadoop Distributed File System) without any processing, and (2) "readRadiationData_rt" which will be used for real-time processing and will insert data to HDFS Note that the first topic will be used for archive data only.

We have configured Apache Flume agents to read from Kafka topics consumers. The Flume agent is responsible of storing data from Kafka to HDFS. In addition, we used Apache Sqoop to read data from relational databases and store it into HDFS. Reading archive data from relational databases is not one of the main goals of the system, but it is an added value to allow user to migrate their old data from traditional warehousing system. In addition, we have used Apache Hive[7] to define the metadata of the file stored in HDFS to make the data retrieval process more easily using SQL-Like languages such as HiveQL and Spark-SQL.

---

[1] https://ubuntu.com
[2] https://python.org
[3] https://kafka.apache.org
[4] http://fume.apache.org
[5] http://sqoop.apache.org
[6] https://zookeeper.apache.org
[7] https://hive.apache.org

For Data processing, our solution relies heavily on Apache Spark[8] for these main reasons: (1) it uses micro batching processing instead of stream processing, which more guarantee fault-tolerance, and in this solution fault tolerance is critical even if it may cause a few milliseconds latency [4]. (2) It can process data at rest and in real-time. (3) Spark has a wrapper library called PySpark that allows creating and running Apache Spark jobs in python. (4) It is scalable so we can add more nodes when it is required.

We have created a single node Apache Spark cluster as the first phase of deployment and we can add other slave nodes when it is required. In addition, we have use pandas[9] python library because it provides many classes and functions that makes handling data easier.

For Data visualization, we used matplotlib[10] library which is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. It also allows drawing real-time graphs.

B. *RaDEn Alarm System*

The data processing and visualization is done by a python script, where we have implemented the radiation alert system (designed based on the Lebanese Atomic Energy Commission requirements) that work as the following: (1) First, the user must define a threshold value. (2) The radiation average is calculated based on the last 30 day from the current date. (3) Then, when the current radiation value is higher than the sum of the radiation average and the threshold a “level 1” alert must be raised which is the lowest alarm level. (4) If it is not raining during the “level 1 alert” then a “level 2 alert” must be raised because the rain increase the radiation level. (5) If an alert was raised (any level) and the radiation level is still high after 5 hours then a “level 3” alert must be raised which is critical and requires a technician to visit the sensor location to do a checkup.

IV. DEMONSTRATION OF RADEN

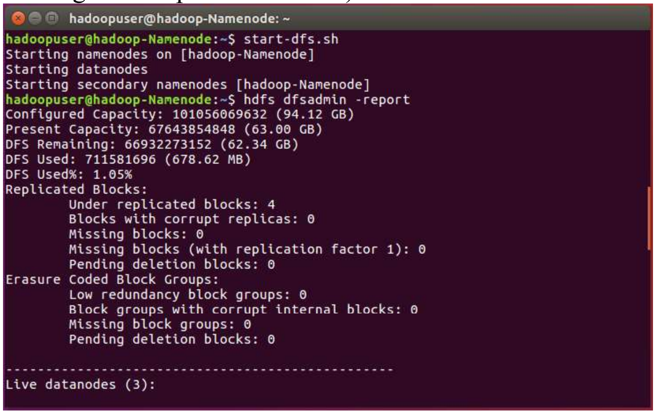
In this section, we demonstrate RaDEn. For our demonstration we used a radiation dataset supplied by the department of environmental radiation control at the Lebanese Atomic Energy Commission (LAEC-CNRS).

A. *Dataset*

The dataset was provided by the LAEC-CNRS, is in form of flat files, because accessing the sensors or the web server (relational database) was not made due to confidentiality issues. The dataset contains the data collected from 2015-08-01 to 2016-08-01 from a testing sensor that was installed in Beirut. It contains information related to radiation such as: radiation level, temperature, rain level, Sensor battery power, data collection time and external battery power.

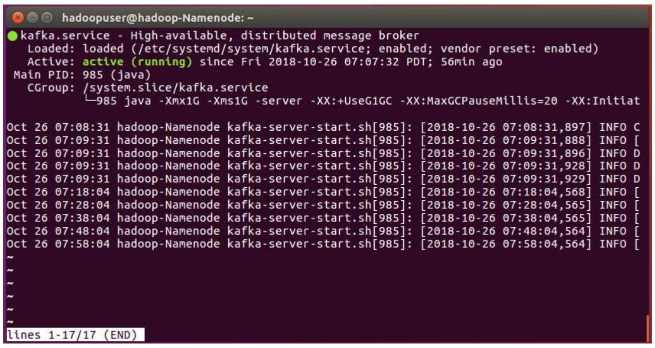
B. *Starting RaDEn*

Before starting the process, the user must start the following services: (1) Hadoop cluster (Figure 2), (2) Apache Kafka service (Figure 3), (3) Apache Spark cluster (Figure 4), (4) Apache Flume agent (Figure 5), (5) Python data processing script (if the user want to insert data to HDFS without visualizing data on a real-time graph, running this script is not needed).



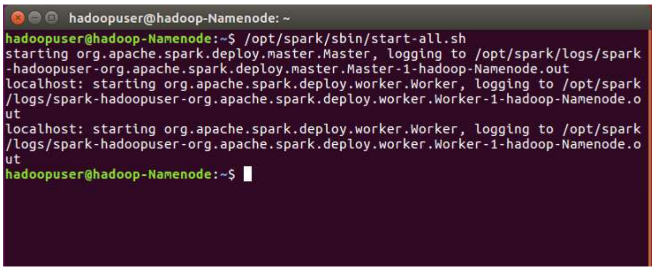
```
hadoopuser@hadoop-Namenode:~$ start-dfs.sh
Starting namenodes on [hadoop-Namenode]
Starting datanodes
Starting secondary namenodes [hadoop-Namenode]
hadoopuser@hadoop-Namenode:~$ hdfs dfsadmin -report
Configured Capacity: 101056069632 (94.12 GB)
Present Capacity: 67643854848 (63.00 GB)
DFS Remaining: 66932273152 (62.34 GB)
DFS Used: 711581696 (678.62 MB)
DFS Used%: 1.05%
Replicated Blocks:
        Under replicated blocks: 4
        Blocks with corrupt replicas: 0
        Missing blocks: 0
        Missing blocks (with replication factor 1): 0
        Pending deletion blocks: 0
Erasure Coded Block Groups:
        Low redundancy block groups: 0
        Block groups with corrupt internal blocks: 0
        Missing block groups: 0
        Pending deletion blocks: 0

-------------------------------------------------
Live datanodes (3):
```

Figure 2 - Starting Hadoop services



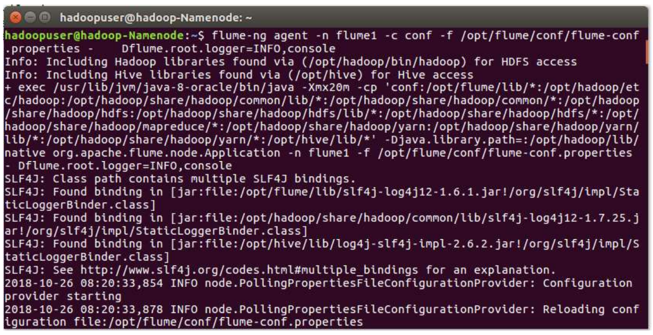
```
● kafka.service - High-available, distributed message broker
   Loaded: loaded (/etc/systemd/system/kafka.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2018-10-26 07:07:32 PDT; 56min ago
 Main PID: 985 (java)
   CGroup: /system.slice/kafka.service
           └─985 java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:Initiat

Oct 26 07:08:31 hadoop-Namenode kafka-server-start.sh[985]: [2018-10-26 07:08:31,897] INFO C
Oct 26 07:09:31 hadoop-Namenode kafka-server-start.sh[985]: [2018-10-26 07:09:31,888] INFO [
Oct 26 07:09:31 hadoop-Namenode kafka-server-start.sh[985]: [2018-10-26 07:09:31,896] INFO D
Oct 26 07:09:31 hadoop-Namenode kafka-server-start.sh[985]: [2018-10-26 07:09:31,928] INFO D
Oct 26 07:09:31 hadoop-Namenode kafka-server-start.sh[985]: [2018-10-26 07:09:31,929] INFO D
Oct 26 07:18:04 hadoop-Namenode kafka-server-start.sh[985]: [2018-10-26 07:18:04,568] INFO [
Oct 26 07:28:04 hadoop-Namenode kafka-server-start.sh[985]: [2018-10-26 07:28:04,565] INFO [
Oct 26 07:38:04 hadoop-Namenode kafka-server-start.sh[985]: [2018-10-26 07:38:04,565] INFO [
Oct 26 07:48:04 hadoop-Namenode kafka-server-start.sh[985]: [2018-10-26 07:48:04,564] INFO [
Oct 26 07:58:04 hadoop-Namenode kafka-server-start.sh[985]: [2018-10-26 07:58:04,564] INFO [
~
~
~
~
~
~
lines 1-17/17 (END)
```

Figure 3 -Starting Kafka services



```
hadoopuser@hadoop-Namenode:~$ /opt/spark/sbin/start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /opt/spark/logs/spark
-hadoopuser-org.apache.spark.deploy.master.Master-1-hadoop-Namenode.out
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark
/logs/spark-hadoopuser-org.apache.spark.deploy.worker.Worker-1-hadoop-Namenode.o
ut
localhost: starting org.apache.spark.deploy.worker.Worker, logging to /opt/spark
/logs/spark-hadoopuser-org.apache.spark.deploy.worker.Worker-1-hadoop-Namenode.o
ut
hadoopuser@hadoop-Namenode:~$ █
```

Figure 4 - Starting Spark services



```
hadoopuser@hadoop-Namenode:~$ flume-ng agent -n flume1 -c conf -f /opt/flume/conf/flume-conf
.properties -    Dflume.root.logger=INFO,console
Info: Including Hadoop libraries found via (/opt/hadoop/bin/hadoop) for HDFS access
Info: Including Hive libraries found via (/opt/hive) for Hive access
+ exec /usr/lib/jvm/java-8-oracle/bin/java -Xmx20m -cp 'conf:/opt/flume/lib/*:/opt/hadoop/et
c/hadoop:/opt/hadoop/share/hadoop/common/lib/*:/opt/hadoop/share/hadoop/common/*:/opt/hadoop
/share/hadoop/hdfs:/opt/hadoop/share/hadoop/hdfs/lib/*:/opt/hadoop/share/hadoop/hdfs/*:/opt/
hadoop/share/hadoop/mapreduce/*:/opt/hadoop/share/hadoop/yarn:/opt/hadoop/share/hadoop/yarn/
lib/*:/opt/hadoop/share/hadoop/yarn/*:/opt/hive/lib/*' -Djava.library.path=:/opt/hadoop/lib/
native org.apache.flume.node.Application -n flume1 -f /opt/flume/conf/flume-conf.properties
- Dflume.root.logger=INFO,console
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/opt/flume/lib/slf4j-log4j12-1.6.1.jar!/org/slf4j/impl/Sta
ticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hadoop/share/hadoop/common/lib/slf4j-log4j12-1.7.25.j
ar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/opt/hive/lib/log4j-slf4j-impl-2.6.2.jar!/org/slf4j/impl/S
taticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
2018-10-26 08:20:33,854 INFO node.PollingPropertiesFileConfigurationProvider: Configuration
provider starting
2018-10-26 08:20:33,878 INFO node.PollingPropertiesFileConfigurationProvider: Reloading conf
iguration file:/opt/flume/conf/flume-conf.properties
```

Figure 5 - Starting Flume agent

C. *Data Ingestion*

To simulate data ingestion we have create a directory where we must copy all flat files, and we created a terminal script that creates a listener on this folder. When any file is inserted, the script loops over the lines and send them one by one to the Apache Kafka producer. Once the data is sent to the Kafka producer, the Apache Flume agent sent it

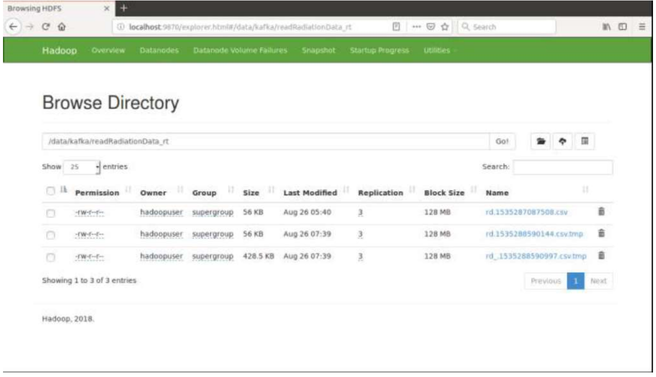
---

[8] https://spark.apache.org

[9] https://pandas.pydata.org

[10] https://matplotlib.org

directly to HDFS to a specific directory and the data is replicated on the three Hadoop data nodes. Figure 6 shows the data ingested into HDFS via Hadoop web interface. Also, figure 7 shows that the file stored in HDFS is replicated on 3 data nodes.



Browse Directory

/data/kafka/readRadiationData_rt

Show 25 entries

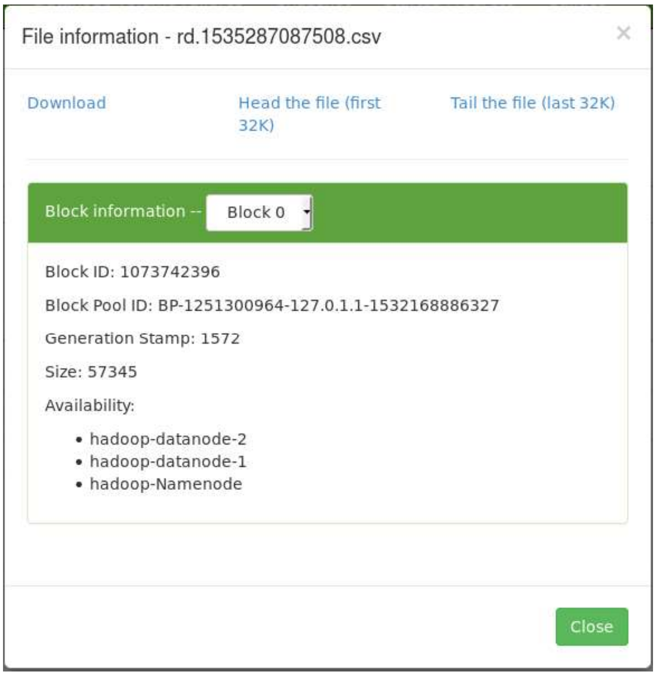
Search:

☐	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
☐	-rw-r--r--	hadoopuser	supergroup	56 KB	Aug 26 05:40	3	128 MB	rd.1535287087508.csv
☐	-rw-r--r--	hadoopuser	supergroup	56 KB	Aug 26 07:39	3	128 MB	rd.1535288590144.csv.tmp
☐	-rw-r--r--	hadoopuser	supergroup	428.5 KB	Aug 26 07:39	3	128 MB	rd_1535288590997.csv.tmp

Showing 1 to 3 of 3 entries

Hadoop, 2018.

Figure 6 - Files ingested into HDFS



File information - rd.1535287087508.csv

Block information -- Block 0

Block ID: 1073742396

Block Pool ID: BP-1251300964-127.0.1.1-1532168886327

Generation Stamp: 1572

Size: 57345

Availability:

- hadoop-datanode-2
- hadoop-datanode-1
- hadoop-Namenode

Figure 7 - Stored file availability

D. Radiation Monitoring

At the same time, the python script read the data from Kafka consumer using the PySpark library, the alarm script is applied, and the radiation level is visualized on a real-time graph using matplotlib library. In figure 8, we have shown sequential snapshots of the real-time graph that was visualized during the experiments and it shows the radiation level changes in function of date and time.

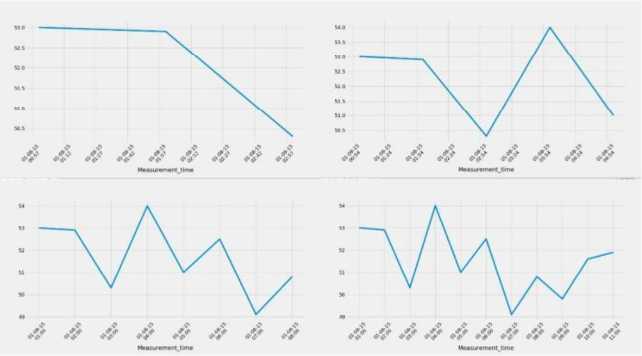
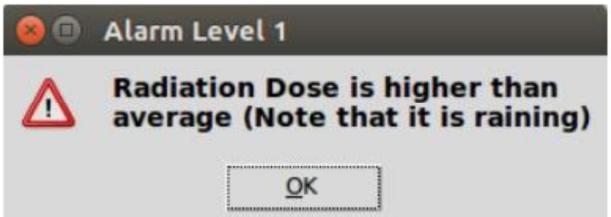


Figure 8 - Real-time graph screenshots

In addition, when an alert is raised it is shown in a message box where the alarm level is written in the title and the description is written in the body. In figure 9, we have showed the level 1 alert message box.



Alarm Level 1

Radiation Dose is higher than average (Note that it is raining)

Figure 9 - Alarm Level 1 Messagebox

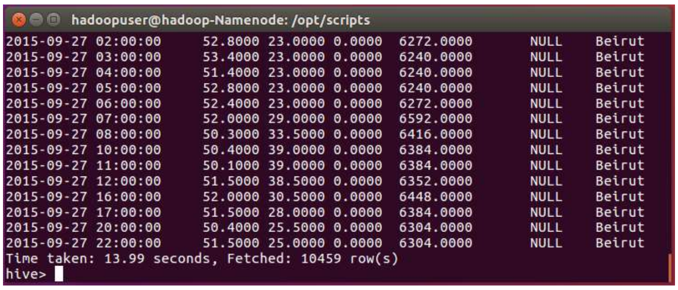
E. Data Retrieval

Based on the Flat files structure and for data retrieval purposes, we created an External Table using Apache Hive on the Directory location in HDFS. Then we created a view from this table to remove messy data such as duplicates rows and rows where dose rate is null.

After created external table on HDFS directory, we can search among the data imported to HDFS using Spark-SQL or HiveQL console. We have to write a query based on our requirements. As example, we need to retrieve all the data where the radiation level is higher than 50. First, we need to run the Spark SQL console using the spark-sql command or hive command to start Hive console, and we can use the following query:

```
SELECT * FROM vw_radiation WHERE dose_rate > 50;
```

In this example, it shows that the number of rows returned is 10459 rows in 13.99 seconds as illustrated in Figure 10.



hadoopuser@hadoop-Namenode: /opt/scripts

```
2015-09-27 02:00:00     52.8000 23.0000 0.0000  6272.0000       NULL    Beirut
2015-09-27 03:00:00     53.4000 23.0000 0.0000  6240.0000       NULL    Beirut
2015-09-27 04:00:00     51.4000 23.0000 0.0000  6240.0000       NULL    Beirut
2015-09-27 05:00:00     52.8000 23.0000 0.0000  6240.0000       NULL    Beirut
2015-09-27 06:00:00     52.4000 23.0000 0.0000  6272.0000       NULL    Beirut
2015-09-27 07:00:00     52.0000 29.0000 0.0000  6592.0000       NULL    Beirut
2015-09-27 08:00:00     50.3000 33.5000 0.0000  6416.0000       NULL    Beirut
2015-09-27 10:00:00     50.4000 39.0000 0.0000  6384.0000       NULL    Beirut
2015-09-27 11:00:00     50.1000 39.0000 0.0000  6384.0000       NULL    Beirut
2015-09-27 12:00:00     51.5000 38.5000 0.0000  6352.0000       NULL    Beirut
2015-09-27 16:00:00     52.0000 30.5000 0.0000  6448.0000       NULL    Beirut
2015-09-27 17:00:00     51.5000 28.0000 0.0000  6384.0000       NULL    Beirut
2015-09-27 20:00:00     50.4000 25.5000 0.0000  6304.0000       NULL    Beirut
2015-09-27 22:00:00     51.5000 25.0000 0.0000  6304.0000       NULL    Beirut
Time taken: 13.99 seconds, Fetched: 10459 row(s)
hive>
```

Figure 10 – Apache Hive query results.

V. CONCLUSION

In this paper, we designed a solution called RaDEn that is able to handle a massive scale of data in real-time and batch style. It allow user to process the radiation data coming from different sources, predict any possible radiation problems and visualize the data in real-time. In addition, it gives the user the ability to query and retrieve the data using a simple SQL-Like language. In addition, we explained how we implemented this solution and we showed a real case scenario.

We tried to cover all the challenges we identified at the beginning of our work but unfortunately, the implementation we have made have some limitations due to the following issues: (1) we received a very small dataset that cannot be considered as Big Data while our system is designed to handle Big Data. (2) We did not get permissions to access the sensors or the databases. (3) There is a lack of documentation for Big Data technologies. (4) The research time limit.

A list of works is lined up to be done in future. More powerful tools such as bokeh[11] and Kibana[12] can be used to increase performance and more options to the end-user and are able to draw a huge number of real-time graphs at the same time. In addition, adding some user interface will make this this solution more powerful, because the current implementation is not user friendly; due to the lack of user interface, also it requires a good knowledge in SQL to be able to retrieve data from HDFS. In addition, the solution should be extended enhance the user the ability to visualize the results of queries on different types of graphs.

Furthermore, we stored data as flat files in HDFS, to improve the performance; we could create an automate job that run periodically and move new data to another HDFS location and convert it to Optimized Row Columnar (ORC) files which gives faster results. Also in future extension, distributed search engines such as Elasticsearch and Solr[13] can be used for data retrieval process.

REFERENCES

[1] "Alpha, Beta, Gamma, X-Ray, and Neutron Radiation," Mirion technologies, [Online]. Available: https://www.mirion.com/introduction-to-radiation-safety/types-of-ionizing-radiation. [Accessed 17 September 2018].

[2] "Ionising Radiation and Human Health," Australian government - department of health, 07 December 2012. [Online]. Available: http://www.health.gov.au/internet/publications/publishing.nsf/Content/ohp-radiological-toc~ohp-radiological-05-ionising. [Accessed 17 September 2018].

[3] "Wireless Sensor Networks to Control Radiation Levels," Libelium, 19 April 2011. [Online]. Available: http://www.libelium.com/wireless_sensor_networks_to_control_radiation_levels_geiger_counters. [Accessed 17 September 2018].

[4] C. Prakash, "Spark Streaming vs Flink vs Storm vs Kafka Streams vs Samza : Choose Your Stream Processing Framework," 21 March 2018. [Online]. Available: https://www.linkedin.com/pulse/spark-streaming-vs-flink-storm-kafka-streams-samza-choose-prakash/. [Accessed 10 9 2018].

[5] Hsin-Fa Fang, Jeng-Jong Wang, Ing-Jang Chen and Jih-Hung Chiu, "The application of GPS, GIS and GPRS in Environmental Radiation Survey," Taiwan.

[6] G. Segura Millan, D. Perrin, L. Scibile, "RAMSES: The LHC Radiation Monitoring System for the Environment and Safety," in 10th ICALEPCS Int. Conf. on Accelerator & Large Expt. Physics Control Systems, Geneva, 2005.

[7] L. R. NAIK, "An Integrated System for Regional Environmental Monitoring and Management Based on Internet of Things," international journal of professional engineering studies, vol. 9, no. 3, pp. 182-186, 2017.

[8] Pablo Andrade Grossi, Leonardo Soares de Souza, Geraldo Magela Figueiredo, Arthur Figueiredo, "Management Information System Applied to Radiation Protection Services," in 2013 International Nuclear Atlantic Conference - INAC 2013, Brazil, 2013.

[9] Kalpana.k, Shruti, Shweta, Bhagyasri, "An Integrated system For Regional Environmental Monitoring and Management Based on IoT," Internatonal Journal of Information Technology and ComputerEngineering, no. 16, pp. 60-64.

[10] Eran Vax, Benny Sarusi, Mati Sheinfeld, Shmuel Levinson, Irad Brandys, Danny Sattinger , Udi Wengrowicz, Avi Tshuva, Dan Tirosh, "ERMS – Environmental Radiation Monitoring System," Beer Sheva, Israel.

[11] Vasile Buruiană, Mihaela Oprea, "A Microcontroller-Based Radiation Monitoring and Warning System," Romania.

[12] Xiaoyu Wanga, Zhaoguo Wang, Liyuan Xu, Deyun Chen, "Wireless Communications Radiation Monitoring System Based on ZigBee and GPRS," Advanced Materials Research, Vols. 403-408, no. 1662-8985, pp. 1956-1959, 2012.

[13] Camelia Avram, Silviu Folea, Dan Radu and Adina Astilean , "Wireless Radiation Monitoring System," in European Conference on Modelling and Simulation, Romania.

[14] Cheng-Jian, Z., Xian-Hua, L., Xiang-Yong, S., & Qing-Zhou, L., "Analysis on the correlation of atmospheric path radiation and air pollution index," in Urban Remote Sensing Event, 2009.

[15] Baker, C., Davidson, G., Evans, T. M., Hamilton, S., Jarrell, J., & Joubert, W., "High performance radiation transport simulations: preparing for Titan," in International Conference on High Performance Computing, Networking, Storage and Analysis, 2012.

[16] Jeong, M. H., Sullivan, C. J., & Wang, S., "Complex radiation sensor network analysis with Big Data analytics," in In Nuclear Science Symposium and Medical Imaging Conference, 2015.

---

[11] https://bokeh.pydata.org/

[12] https://www.elastic.co/products/kibana

[13] http://lucene.apache.org/solr/