DOI: 10.2507/30th.daaam.proceedings.054

IMPLEMENTING BIG DATA PROCESSING WORKFLOWS USING OPEN SOURCE TECHNOLOGIES

Alexander Suleykin[a] & Peter Panfilov[b]

[a] Russian Academy of Sciences' V.A.Trapeznikov Institute of Control Sciences, Profsoyuznaya St. 65, Moscow 117342, Russian Federation
[b]National Research University – Higher School of Economics, Myasnitskaya St. 20, Moscow 101000, Russian Federation

This Publication has to be referred as: Suleykin, A[lexander] & Panfilov, P[eter] (2019). Implementing Big Data Processing Workflows Using Open Source Technologies, Proceedings of the 30th DAAAM International Symposium, pp.0394-0404, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-22-8, ISSN 1726-9679, Vienna, Austria
DOI: 10.2507/30th.daaam.proceedings.054

Abstract

In our implementation research, we apply workflow approach to the modeling and development of the Big Data processing pipeline using open source technologies. The data processing workflow is a set of interrelated steps which launch some particular jobs such as Spark job, shell job or Postgre SQL command. All workflow steps are chained to form integrated process and imitate the data load from staging storage area to the datamart storage area. The experimental workflow-based implementation of a data processing pipeline was performed that stages through different storage areas and uses actual industrial KPI dataset of some 30 millions records. Evaluation of implementation results provides proofs of the applicability of proposed workflow to other application domains and datasets which should satisfy the data format at input stage of the workflow.

Keywords: Workflow; Big Data: Data Processing Pipeline; KPI Data.

1. Introduction

A workflow technology [1] is widely used to describing and automating various kinds of processes, including data processing, application integration or business processes. In the core of the workflow approach is the complex process representation in the form of a so-called workflow that consists of elementary operations and dependencies between them, the executors of operations and the rules for processing external events. Dependencies between operations can be described both in terms of management flows (a sequence of operations, synchronization between them), and data flows. In the latter case, the workflow model can be related to the dataflow approach to distributed computing and data processing.

Many software tools have been developed that allow for describing particular workflows in various application areas and implementing them on different classes of computer systems, including distributed ones. One of the important areas of application of the workflow technology is the automation of scientific and engineering calculations [2]. In this case, the elementary operations-blocks of workflow are the starts of various computational blocks, models and packages. The drawback of existing workflow systems is the lack of support for blocks with an internal state that can repeatedly receive data and change their state accordingly. The implementation of this functionality would allow expanding the scope of application systems to new classes (iterative processes, dataflow processing, simulation, predictive analytics, Big Data).

The complexity of big data processing and analysis is extremely increasing due to data volume growth, data variety, velocity, different data formats of data transmission, integration problems and other data complexities. Building a robust, reliable and fault-tolerant data processing and storage framework that is capable of handling big data loads with data in various formats and high volumes from different data sources and systems represent a great challenge to application developers. Many researchers consider the workflow modeling as a viable approach to the design and implementation of distributed application systems for processing large volumes of data (Big Data) [3].

Avalanche-like data growth as a result of the rapid development of information technologies and systems has led to the emergence of new models and technologies for distributed data processing, such as MapReduce, Dryad, Spark [5]. Also special purpose systems for distributed computing based on the dataflow approach were introduced, for example, for processing large graphs [6]. In addition to the systems focused on batch (offline) data processing, systems and services for real-time (online) data processing such as Storm, Spark Streaming, Kafka Streams [7]-[9], [13]-[14] receive more attention due to user requirement of rapid and smart response to incoming data. These systems use the distributed data processing operations to support large volumes of incoming data and to adopt high speed of data arrival.

An important feature of existing data-driven software systems for distributed data processing is the abstraction of the programmer from the details of the implementation of computations by using ready-made primitives (distributed dataflow-operators, for example, map and reduce). On the one hand, this allows simplify writing programs that fit into the proposed model of computations, but on the other hand - makes it difficult to implement other classes of applications. For example, systems based on the MapReduce model are poorly adapted to implement iterative algorithms and strongly connected applications. To overcome limitations of existing distributed data processing models and technologies numerous specialized solutions for various types of applications have been developed [4].

Nowadays the open-source solutions are becoming more and more popular and Hadoop stack with its already improved Map Reduce data processing engine is one of the most widely used technologies for big data storage. Based on Hortonworks Data Platform stack [10], it delivers 100% open-source global data management platforms and services so customers can manage the full lifecycle of their data. This stack is widely accepted by many large companies for data processing, storage, analysis and visualization.

The current research is focused on application of open source big data technologies based on HDP Hadoop [11] ecosystem stack for building data processing workflow with all job dependencies and launching different jobs from one workflow orchestrator. The workflow is modeled and implemented based on sample industrial KPIs data showing applicability of proposed methodology for any real world data with particular formats at input. The workflow consists of a set of interconnected jobs – Spark jobs, shell jobs and Postgre SQL [12] commands. All workflow steps are connected and modeled in one data pipeline.

2. Workflow model of data processing pipeline

2.1. *Architectural view of the data processing pipeline*

For the purposes of our implementation research, we have architected a somewhat simplified but powerful multi-staged data processing pipeline that combine two major super stages or layers – Data Storage Layer and Serving Layer (fig. 1). Here Storage Layer and Serving Layer have their own Layers (sublayers), which are used for methodological correctness of data load. The data processing pipeline of the whole data movement is strict and should go through the following sublayers/stages inside Serving and Storage Layers as it is depicted in fig. 1. The details of the sublayer implementation such as open source technology used, functionality and data transformations are presented in Table 1.

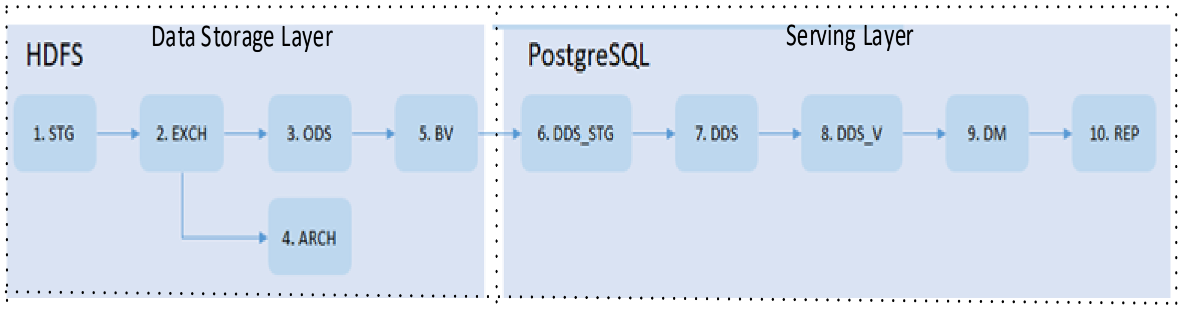

Fig. 1. The data processing pipeline workflow and layers interconnection

Name	Abbreviation	Location	Definition and functions	Transformations
Staging Buffer Area	STG/BUF	HDFS	The area of temporary data accumulation in the format corresponding to the source without any transformations. Streaming data comes from sources.	No
Staging Exchange Area	STG/EXCH	HDFS	The intermediate region for forming the next ETL processing packet. All accumulated data are moved from the buffer to form a data processing packet. It is assigned a unique BATCH_ID.	BATCH_ID
Staging Archive Zone	STG/ARCH	HDFS	Storage of the complete archive of incoming messages without transformation of the storage format. Incoming messages are archived after successful processing.	Archiving and enlarging storage files.
Operational Data Store	ODS/HIST	HDFS	The area in which the source data scheme is stored, but they are reduced to a single binary form of storage. It contains the entire history of changes and deletions.	Convert to binary storage format. Conversion from object to relational storage.
Batch View	ODS/BW	HDFS	It contains only an actual slice of the state of objects without a change history and deleted records.	Calculation of the actual data slice.
Detail Data Store Staging	DDS_STG	Postgre	Batch layer. A separate instance is created for each source system. One-to-one data is transferred from HDP and stored only between downloads. Both full data load and only line changes (deltas) can come.	
Detail Data Store	DDS	Postgre	The layer of the current data slice presented in a relational form.	Re-keying (generation of internal storage IDs). Conversion from object to relational storage. Normalization of data (if necessary). Creating a single data model (without unification) Storing a current data slice
Detail Data Store View	DDS_v	Postgre	The layer for data access to dds	All data access to dds should be organized using views on top of dds layer istelf
Data Mart	DM	Postgre	Groups showcases by a specific attribute, most often the subject area. Contains unified detailed data. It contains calculated indicators for use in reporting. Calculation of indicators used in several reports is necessarily submitted to this layer.	Data unification. Denormalization of data. Data Aggregation. Calculation of derived indicators used in several places.
Data Mart Reporting	DM_rep	Postgre	The final reporting layer. From it, data are used only for display in BI tools. It is forbidden to build some reports on the basis of others. Only with the transfer of the information used in the DM layer. Calculation of indicators specific to specific reporting. It can be both logical and physical.	Calculation of derived indicators specific to a particular report.

Table 1. Description and definition of selected sublayers of the data processing pipeline.

For the sake of versatility of the solution, we have deliberately narrowed the choice of open source Big Data-oriented technologies to only four technologies that meet the following limitations and requirements:

- Horizontally scalable for Batch layer – layer for data loads to archival and distributed storage
- Have distributed data processing engine in order to be capable with Big Data volumes
- Fault tolerant – if one machine fails, the solution should still work and no data loss
- All workflow steps should be orchestrated and combined in one data pipeline launching different jobs
- Have relational database for data access to different BI tools
- Be Open Source.

This way we have defined the list of technologies for building distributed data processing applications which are readily available for the workflow modeling of the Big Data processing pipeline (table 2):

№	Component	Definition
2	Spark	Distributed in-memory framework for high-load data processing [8]
1	HDFS	Distributed fault tolerant file system optimized for storage for processing large volumes of data [10]
3	PostgreSQL	Relational database to provide BI data to tools [11]
4	AirFlow	Universal Scheduler [12]

Table 2. Selected technologies for workflow modeling

2.2. *Implementation of the data processing pipeline's steps*

Each workflow step consists of different types of jobs – shell job, dummy job (specific for particular technology) for launching or ending ETL process, Spark job or Postgre SQL commands. All job results are being aggregated in one workflow orchestrator.

According to the architecture, the workflow steps are as follows.

Step 1. *STG (Staging) – incoming data files recorded by any ETL process in hdfs staging (STG) directory.*

The file formats should be in Xml data format with the following structure:

```
<entity_name>
    <SYSTEM_CODE> </SYSTEM_CODE>
    <EXTRACT_GUID></EXTRACT_GUID>
    <EXTRACT_DTTM></EXTRACT_DTTM>
    <ATTR_1></ATTR_1>
    <ATTR_N></ATTR_N>
    …
</entity_name>
```

Here SYSTEM_CODE refers to the code of the source system, EXTRACT_GUID is the guid of the input batch, EXTRACT_DTTM is date and time for capturing data from the source, and ATTR_1 and ATTR_N are the attributes of the entity instelf.

Step 2. *EXCH (Exchange) - the processed data packet (XML).*

To perform the calculation, it is necessary to prepare the data, which means in our case transferring data from the STG directory to EXCH with maintaining the directory structure of the output of the incoming ETL process. To transfer data, a special application was developed for transferring files from stg directory and creating a directory structure in exch directory of hdfs.

The transfer from STG to EXCH is carried out using the spark-submit utility with the following example input parameters:

- Download ID
- URI hdfs – it is needed specify the directory with the rights to change
- Directory with files
- Directory of receiving files.

Launching example:

```
spark-submit
--class spark.archiver.MoveFiles
--master yarn
--deploy-mode cluster
--name mv__test__0
--driver-memory 1g
--driver-cores 1
--executor-memory 2g
--executor-cores 3
--conf spark.dynamicAllocation.executorIdleTimeout=600s
--conf spark.dynamicAllocation.cachedExecutorIdleTimeout = 600s
--conf spark.port.maxRetries=50
--conf spark.dynamicAllocation.enabled=true
--conf spark.shuffle.service.enabled=true
hdfs://testcluster/apps/source_system/ver/latest/jar/archiver.jar
hdfs://testcluster/data/source_system/source_system/stg_data
hdfs://testcluster/data/source_system/source_system/exch_data
```

Step 3. *ODS (Operating Data Store) - accumulated data (data format - parquet).*
The transfer from EXCH to ODS is carried out using the spark-submit utility. The last 5 parameters to change:

- Upload id
- Hdfs uri - xml source
- URI hdfs - parquet receiver
- XML record tag
- Hdfs uri - schema json file.

Launching example:

```
spark-submit
--class batch.STG2ODS
--master yarn
--deploy-mode cluster
--name exch2ods__entity_name
--driver-memory 1g
--driver-cores 1
--executor-memory 1g
--executor -cores 1
--conf spark.dynamicAllocation.executorIdleTimeout =600s
–conf spark.dynamicAllocation.cachedExecutorIdleTimeout = 600s
--conf spark.port.maxRetries=50
--conf spark.dynamicAllocation.enabled=true
--conf spark.shuffle. service.enabled=true
hdfs://testcluster/apps/rnd/SOURCE_SYSTEMN-543-BatchLoadMethod/jars/transformer2.jar
--load_id=1
--job_id=1
--src_path=hdfs://testcluster/data/source_system/exch_data
--tgt_path=hdfs://testcluster/data/source_system/ods_data
--table=datamart
--pathXmlSchema=hdfs://testcluster/ctl/source_system/stg/public/datamart/datamart.json.
```

Step 4. *BV (Batch View) - data aggregated by the process_dttm attribute (data format - parquet).*
The transfer from ODS to BV is carried out using the spark-submit utility. The following 5 named parameters are responsible for the formation of the batch view:

- lid - boot id
- src_path - URI hdfs - source ods
- tgt_path - URI hdfs - receiver batch view
- message_id - Primary key
- extract_dttm - Timestamp
- load_method - the loading method can be DELTA, BATCH, FULL.

Launching example:

```
spark-submit
--class batch.ODS2BV
--master yarn
--deploy-mode cluster
--name ODS2BV__entity_name
--driver-memory 1g
--driver-cores 1
--executor-memory 1g
--executor-cores 1
--conf spark.dynamicAllocation.enabled=true
--conf spark.dynamicAllocation.minExecutors=2
--conf spark.dynamicAllocation.maxExecutors=30
--conf spark.dynamicAllocation.initialExecutors=2
--conf spark.shuffle.service.enabled=true
hdfs://testcluster/apps/rnd/SOURCE_SYSTEMN-543-BatchLoadMethod/jars/transformer2.jar
--load_id=1
--job_id=1
--src_path=hdfs://testcluster/data/source_system/ods/public/ datamart/*. Parquet
--tgt_path=hdfs: //testcluster/data/source_system/batch_view/ public/datamart
--load_method=DELTA
--ikey = date
--extract_dttm = process_dttm
```

Step 5. *Writing to the Postgre SQL database (DDS_STG) from BV is carried out using the spark-submit utility*. Arguments for starting the service are as follows:

- load_id - load identifier
- job_id - pipeline loading process id
- src_path - The path to the hdfs directory
- jdbc_driver_name - JDBC database driver, e.g. org.postgresql.Driver
- jdbc_driver_url - database URL
- jdbc_login - Database user
- jdbc_pass - Database Password
- db_schema_name - Database schema
- db_table_name - Database table e.g. datamart
- meta_mapping_path - Path to the dataframe mapping description file in the database table, for example res / config / simple-datamart-to-db.json
- conn_cnt - Number of partitions (a connection to the database is created for each partition), for example 10
- batch_size - The number of entries in batch, for example 10.

Launching example:

```
spark-submit
--class spark.connector.CopyParquetDataToDbConnector
--master yarn
--name 2db__datamart__test
--deploy-mode cluster
--executor-cores 2
--executor-memory 3g
--conf spark.dynamicAllocation.enabled=true
--conf spark.shuffle.service.enabled=true
--conf spark.dynamicAllocation.maxExecutors=20
--conf spark.yarn.executor.memoryOverhead=1g
hdfs://testcluster/apps/rnd/SKIMN-543-BatchLoadMethod /jars/connector.jar
--load_id=1
--job_id=1
--src_path=
hdfs://testcluster/data/source_system/batch_view/public/datamart
--jdbc_driver_name=org.postgresql.Driver
--jdbc_driver_url=jdbc:postgresql
--jdbc_login=hdfs_importer
```

```
--jdbc_g__d = = Import
--db_table_name=datamart
--meta_mapping_path=
hdfs://testcluster/ctl/sourcesystem/batch_view/public/datamart/datamart.json
--conn_cnt=30
- -batch_size = 1000
```

Step 6. *After writing to DDS_STG (5), a har archive is created from the exch folder using Hadoop archive script.* Script parameters are as follows:

- Download ID
- Directory for archiving
- Directory where to put the archive.

Step 7. *The creation of the har archive is completed by cleaning the exch folder*:

- Download ID
- URI hdfs
- Delete directory.

Step 8. *PG_etl_dm – the launch of the function for all calculations needed for data marts forming, and data marts creation itself.*

Step 9. *Pg_etl_gen_orphans – generation of entities which are not found their links in database.*

Pipeline data movement can be carried out either by manually running the corresponding scripts with parameters, or automatically using DAG Apache Airflow.

Beforehand, the distribution should be installed on the node with the spark client installed.

There some special system steps that are needed for successful DAG modeling and launching including:

- Begin – shows that DAG is started
- End – end of DAG processing
- Lid – generation of ID of DAG
- One success – shows that onle one incoming flow of the DAG is needed for starting of future steps
- All success – shows that all incoming steps are needed to be successful for future DAG implementation.

3. Experiments and results

3.1. Dataset description

The proposed workflow has been tested using actual Key Performance Indicators (KPIs) data from a railway company. The data is represented by star schema that consists of one fact table (the main table with events – KPIs) referencing so called dimension tables (dictionaries in this case). The data is normalized to the 3-rd normal form (3NF).

Nine entities have been used in our experiments for testing data processing workflow implementation. Those entities were datamart, cargo-type, data_type, date-type, metric-type org, unit, val_type and var. Data volumes of these entities are presented in table 3, which give us the total volume of data amounting 26,4 Gb. The description of entities and data types are presented in table 4.

Entity	Number of records
datamart	30 mln
cargo_type	46
data_type	2
date_type	5
metric_type	15
org	84
unit	120
val_type	5
var	418

Table 4. Data volumes of test entities

Entity	Attribute	Data type	Description
DATA_TYPE	ID	INTEGER	Dictionary – type of data for KPI. Can be approved or planned
	NAME	CHAR	
DATE_TYPE	ID	INTEGER	Dictionary – type of date period for selected KPI. Can be day, week, month, year
	NAME	CHAR	
METRIC_TYPE	ID	INTEGER	Dictionary – version of the KPI
	NAME	CHAR	
ORG	BK	INTEGER	Dictionary – branch of main company for selected KPI
	ID	INTEGER	
	ROOT_ID	INTEGER	
	CHILD_ID	INTEGER	
	VNAME	CHAR	
	NAME	CHAR	
UNIT	ID	INTEGER	Dictionary – unit of measurement
	NAME	CHAR	
VAL_TYPE	ID	INTEGER	Dictionary – type of cumulative data representation. Can be cumulative values by years, months, days, weeks
	NAME	CHAR	
VAR	VAR	INTEGER	Dictionary – the variable represents KPI name itself and definition
	NAME	CHAR	
DATAMART	DATE	DATE	Fact table, represent all KPIs of railway company
	MUNIT	CHAR	
	DATA_TYPE	INTEGER	
	VAR	INTEGER	
	VALUE	DOUBLE	
	VAL_TYPE	INTEGER	
	METRIC_TYPE	INTEGER	
	DATE_TYPE	INTEGER	
	SOURCE	CHAR	
	ORG	INTEGER	
	CARGO_TYPE	INTEGER	
	REASON_TEXT	CHAR	
	MOD_TIME	TIMESTAMP	
	USER_LOGIN	CHAR	
CARGO_TYPE	CARGO_TYPE	INTEGER	Dictionary – type of cargo for Cargo KPIs
	NAME	CHAR	

Table 4. Railway KPIs data description and its types

3.2. *Experimental setup*

Tests of the proposed data processing workflow were carried out in the simulation testbed. Platform servers are located on virtual machines based on the same physical server. The testbed was deployed using Microsoft Hyper-V virtualization based on a dedicated servers, and test runs were conducted using virtual machines.

The configuration of a server is shown in Table 5, and the configuration of test virtual machines is shown in Table 6 respectively:

Server model	CPU	RAM	Disks
Huawei FusionServer 1288H V5	12-core (24 logical HT) Intel Xeon Gold 6126 2.6GHz	512 Гб DDR4, 32GB, 2666MT/s, 2Rank (2G*4bit), ECC	9 TB 300GB, SAS 12Gb/s, 10K RPM Huawei Avago3508 RAID SR450C-M 2G

Table 5. Host Configuration - Virtualization Server

VM host	CPU (vCores)	RAM Gb	HDD Tb	Services and their role
pg-source_system.bd	8	16	1	Postgre SQL server – data base for Serving Layer
bd-hdp-edge01	8	32	1	Airflow server – ETL orchestration
source_systemn-hdp-01	4	32	1	• PostgreSQL metadata server for Ambari, Hive, Ranger • Ambari Server; Ranger Admin; HDFS JournalNode • Ranger KMS; Ranger Usersync • Zookepeer Server; Solr Server; Ambari Metrics Server • Ambari Metrics Grafana • Netdata-мастер; Influxdb
source_systemn-hdp-02	4	32	1	• HBase Standby Master; HDFS Standby Namenode • Spark2 History Server • YARN Active ResourceManager; YARN Registry DNS; YARN Timeline Service V2.0 Reader • Zookeeper Server; ZKFailoverController • Netdata-agent; Ambari Metrics Monitor
source_systemn-hdp-03	4	16	1	• HDFS Active NameNode; HDFS JournalNode • MapReduce2 History Server • Hive Metastore; Hive Server • YARN Standby ResourceManage • YARN Timeline Service V1.5 • ZooKeeper Server; ZKFailoverController • Netdata-agent; Ambari Metrics Monitor
source_systemn-hdp-04	4	16	1	• HBase Active Master • HDFS JournalNode • Zookeper Server • Spark2 Thrift Server
source_systemn-hdp-05	4	16	1	• HDFS Datanode • HBase Regionserver • YARN Nodemanager • Netdata-agent; Ambari Metrics Monitor
source_systemn-hdp-06	4	16	1	• HDFS Datanode • HBase Regionserver • YARN Nodemanager • Netdata-agent; Ambari Metrics Monitor
source_systemn-hdp-07	4	16	1	• HDFS Datanode • HBase Regionserver • YARN Nodemanager • Netdata-agent; Ambari Metrics Monitor
source_systemn-hdp-08	4	16	1	• HDFS Datanode • HBase Regionserver • YARN Nodemanager • Netdata-agent; Ambari Metrics Monitor
source_systemn-hdp-09	4	16	1	• HDFS Datanode • HBase Regionserver • YARN Nodemanager • Netdata agent; Ambari Metrics Monitor
source_systemn-hdp-10	4	16	1	• HDFS Datanode • HBase Regionserver • YARN Nodemanager • Netdata-agent; Ambari Metrics Monitor
source_systemn-hdp-11	4	16	1	• Kafka broker; kafka-hdfs-connector • NFSGateway: /mnt/hdfs • Hive client; Spark2 client • Netdata-agent; Ambari Metrics Monitor

Table 6. Testbed configuration

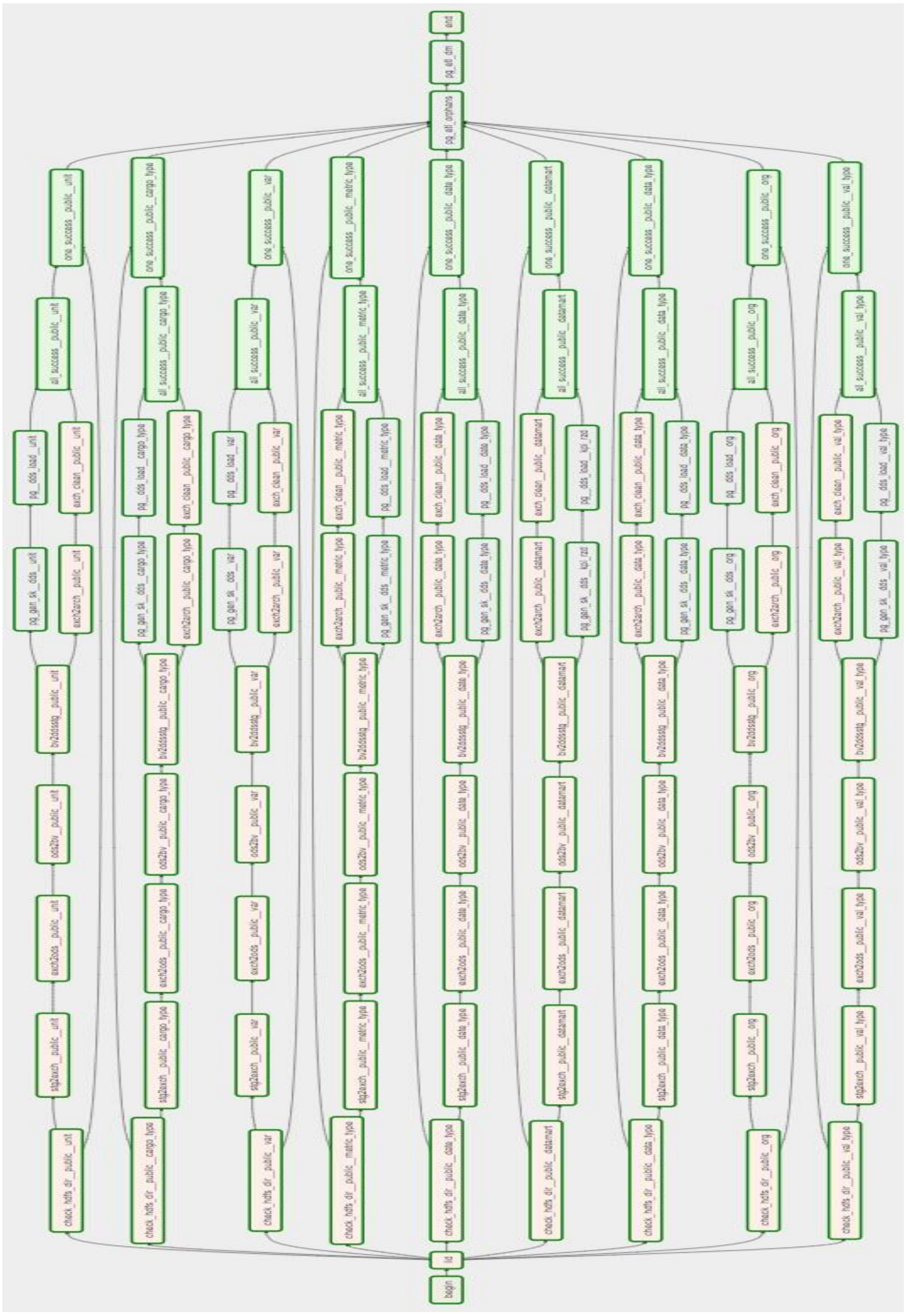
begin
lid
check_hdfs_dir__public__unit
stg2exch__public__unit
exch2ods__public__unit
ods2bv__public__unit
bv2ddsstg__public__unit
pg_gen_sk__dds__unit
exch2arch__public__unit
pg__dds_load__unit
exch_clean__public__unit
all_success__public__unit
one_success__public__unit
check_hdfs_dir__public__cargo_type
stg2exch__public__cargo_type
exch2ods__public__cargo_type
ods2bv__public__cargo_type
bv2ddsstg__public__cargo_type
pg_gen_sk__dds__cargo_type
exch2arch__public__cargo_type
pg__dds_load__cargo_type
exch_clean__public__cargo_type
all_success__public__cargo_type
one_success__public__cargo_type
check_hdfs_dir__public__var
stg2exch__public__var
exch2ods__public__var
ods2bv__public__var
bv2ddsstg__public__var
pg_gen_sk__dds__var
exch2arch__public__var
pg__dds_load__var
exch_clean__public__var
all_success__public__var
one_success__public__var
check_hdfs_dir__public__metric_type
stg2exch__public__metric_type
exch2ods__public__metric_type
ods2bv__public__metric_type
bv2ddsstg__public__metric_type
exch2arch__public__metric_type
pg_gen_sk__dds__metric_type
exch_clean__public__metric_type
pg__dds_load__metric_type
all_success__public__metric_type
one_success__public__metric_type
check_hdfs_dir__public__data_type
stg2exch__public__data_type
exch2ods__public__data_type
ods2bv__public__data_type
bv2ddsstg__public__data_type
exch2arch__public__data_type
pg_gen_sk__dds__data_type
exch_clean__public__data_type
pg__dds_load__data_type
all_success__public__data_type
one_success__public__data_type
check_hdfs_dir__public__datamart
stg2exch__public__datamart
exch2ods__public__datamart
ods2bv__public__datamart
bv2ddsstg__public__datamart
exch2arch__public__datamart
pg_gen_sk__dds__kpi_rzd
exch_clean__public__datamart
pg__dds_load__kpi_rzd
all_success__public__datamart
one_success__public__datamart
check_hdfs_dir__public__org
stg2exch__public__org
exch2ods__public__org
ods2bv__public__org
bv2ddsstg__public__org
pg_gen_sk__dds__org
exch2arch__public__org
pg__dds_load__org
exch_clean__public__org
all_success__public__org
one_success__public__org
check_hdfs_dir__public__val_type
stg2exch__public__val_type
exch2ods__public__val_type
ods2bv__public__val_type
bv2ddsstg__public__val_type
exch2arch__public__val_type
pg_gen_sk__dds__val_type
exch_clean__public__val_type
pg__dds_load__val_type
all_success__public__val_type
one_success__public__val_type
pg_etl_orphans
pg_etl_dm
end

Fig. 2. The output of workflow DAG run in Apache Airflow after modeling and implementation

3.3. Test results

As a result of modeling, the Directed Acyclic Graph (DAG) is created which consists of steps described above. Thus, after successful running of DAG in Airflow the DAG itself is becoming green showing all DAG steps have finished properly (fig. 2).

4. Conclusion

In our research we have modeled and implemented the workflow of the data processing pipeline that uses Open Source Big Data technologies. We have used technologies such as Apache Hadoop hdfs for data movement and conversion from XML data format to Parquet and archiving data to har files, Postgre SQL for data transformations, and Apache Airflow for Orchestrating ETL processes. As a result of the experimental research, the successful implementation of the workflow model and one full data processing pipeline have been created. The test data loads have been performed and demonstrated the versatility of the proposed workflow framework and its applicability for other domains and use cases.

5. Acknowledgments

The article was partly supported by the Academic Fund Program at the National Research University Higher School of Economics (HSE) in 2018 — 2020 (grant №19-01-086) and by the Russian Academic Excellence Project «5-100».

6. References

[1] Workflow Management Systems and Interoperability, Dogac, A., Kalinichenko, L, Ozsu, T, Sheth, A. (Eds.), Springer-Verlag, ISBN 13: 9783540644118, Berlin Heidelberg, 1998. Pages: XVII, 526.
[2] Workflows for e-Science: scientific workflows for grids. Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M. (Eds.), Springer-Verlag, ISBN 13: 9781846285196, London, 2007. Pages: XXII, 526.
[3] Zhang, B.; Yu, L.; Feng, Y.; Liu, L.; Zhao, S. (2018). Application of Workflow Technology for Big Data Analysis Service. *Appl. Sci.* 2018, *8*, 591.
[4] Suleykin, A. and Panfilov, P. (2019). Distributed Big Data Driven Framework for Cellular Network Monitoring Data, 2019 24th Conference of Open Innovations Association (FRUCT), Moscow, Russia, 2019, pp. 430-436. DOI: 10.23919/FRUCT.2019.8711912.
[5] Singh, D., Reddy, C. K. (2015). A survey on platforms for large data analytics. Journal of Big Data. 2015, Vol.2, No.1, pp. 8-8.
[6] Gonzalez, J. E. et al. (2014). GraphX: Graph Processing in a Distributed Dataflow Framework. OSDI. 2014, 14, pp.599-613.
[7] Zaharia, M. et al. (2012). Discretized Streams: An Efficient and Fault-Tolerant Model for Stream Processing on Large Clusters. HotCloud. 2012, Vol.12, pp.10-10.
[8] https://kafka.apache.org. Accessed on: 11-08.2019.
[9] https://spark.apache.org/. Accessed on: 11-08.2019.
[10] https://hortonworks.com. Accessed on: 2019-07-31.
[11] https://hadoop.apache.org/. Accessed on: 11-08.2019.
[12] https://www.postgresql.org/. Accessed on: 11-08.2019.
[13] https://airflow.apache.org/. Accessed on: 11-08.2019.
[14] https://www.confluent.io/connector/kafka-connect-hdfs/. Accessed on: 11-08.2019.
[15] Red Hat Jboss Fuse official website. https://www.redhat.com/en/technologies/jboss-middleware/fuse. Accessed on: 11-08.2019.