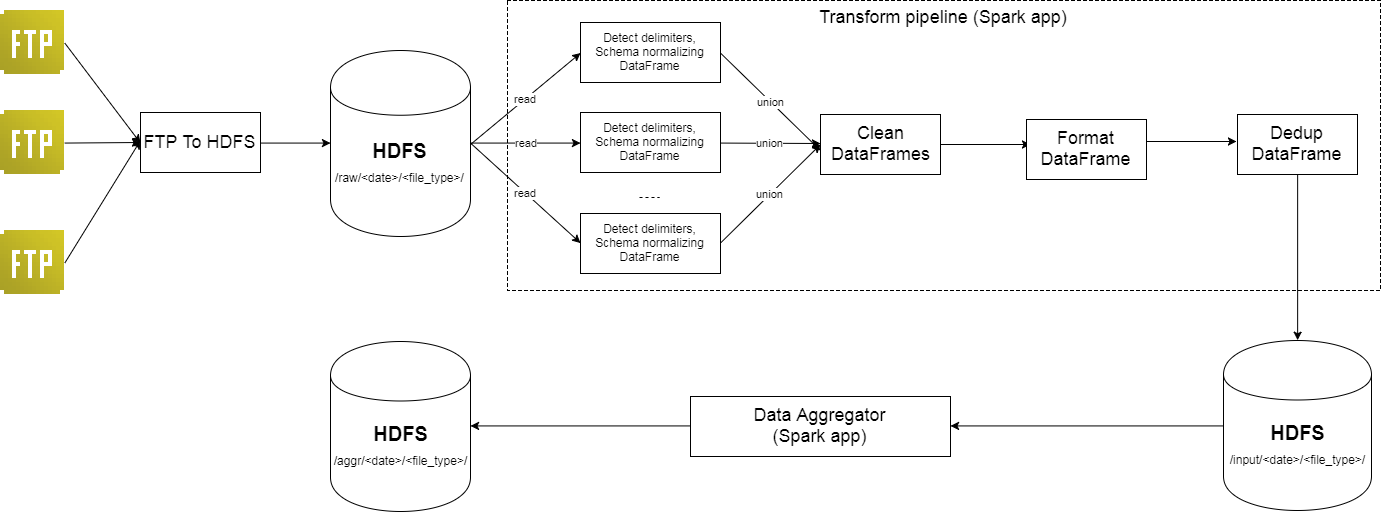
# High level design

There’re several choices to implement the data pipeline. Here I can list out 2 alternatives in my knowledge:

1. Using Hadoop MapReduce: Each processor (clean, format, dedup or data aggregation) is a MapReduce task. Oozie will be used to coordinate the workflow of pipeline processors (MapRed tasks).
2. Using Apache Spark: Each pipeline processor is a RDD. This save a lot of IO cost as the intermediate results between processors aren't written to disks, comparing to MapReduce where the output of a task will be written to HDFS so that it can be taken as input of the next task (one of our requirement is high performance). Another enhancement is that we can later reuse the code (RDD or DataFrames) when we want to implement the Pipeline as **realtime** one, using Spark Streaming. We will use Spark due to the mentioned advantages.

Below is the high level design of the system:



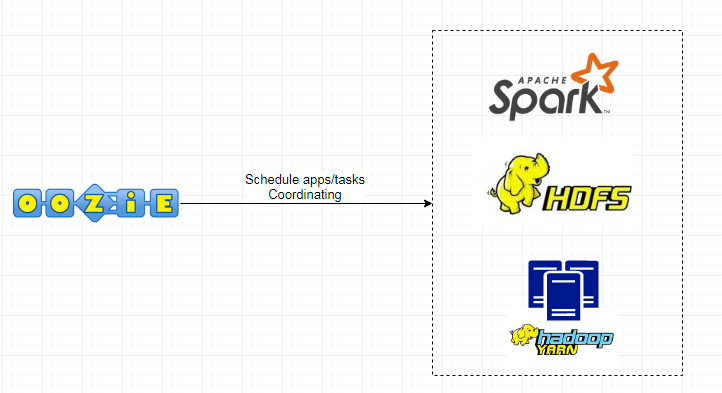
The system is composed from the following component

* **FTP to HDFS**: is a batch job to download CDR files from FTP server(s) and store as raw files in HDFS. It also responsible to identify the type of a file to put in to the right directory in HDFS, in which expected path is /raw/<date>/<file\_type>/. For example, all call history files on August 18, 2017 should be downloaded correctly to /raw/2017-08-18/call\_history/.
* **Transform pipeline**: is a Spark application (aka. Spark job) that run in batch mode. Take an input as a directory location to the raw CRS files of a type in a specific day, e.g. /raw/2017-08-18/call\_history/. It composes of several DataFrames to do the transformation of data.
  + Each file will be process one by one to: 1. Identify the delimiter, 2. Read as a dataframe, 3. Normalize schema (e.g. column names standardizing).
  + Clean: The dataframes in the previous steps is union in to one. Then data cleansing is perform to detect and filter invalid items (except those which can be corrected by Format.)
  + Format: correct items, properly format the data to a unique form, e.g date format or phone number normalization.
  + Dedup: use in place dropDuplication to filter out duplicated data. This can be done once we process data of a whole day in a batch. *To change to streaming ELT, we have to consider using Memcached or BloomFilter techniques.*

The output of this Spark job is place at the directory in the pattern /input/<date>/<file\_type>/, e.g. /input/2017-08-18/call\_history/.

* **Data aggregator**: Is a Spark job to perform data aggregation. It take the input, as the output of the Transform pipeline, e.g. /input/2017-08-18/call\_history/, then perform data proper data aggregation based on the type of file type. It uses Spark Dataframe to perform aggregation in SQL styles. More detail of this job can be found at the attached code.

# Technologies



Below are list of technologies we will use:

* Apache Oozie: is uses to schedule/submit tasks (FTP to HDFS, Transform Pipeline, Aggregator app). It will also coordinating tasks, track progress, alerting when tasks fail. Oozie will be installed in a dedicated server, so that it may become a single point of failure (SPOF). But, starting with Oozie version 4.1.0, we can configure multiple active Oozie servers against the same database to provide high availability for the Oozie service.
* Hadoop YARN: provide resource management. It will be the “operating system” to run HDFS and Apache Spark on top of it.
* Apache Spark: we develop the ETL and Aggregation pipeline on top of it. SPARK provide a high-availability, fault tolerant, distributed, scalable processing engine. It’s also a better performance platform comparing to alternatives, such as Hadoop MapReduce, due to its in-memory processing nature.
* HDFS: needless to say, it provides fault tolerant, distributed and scalable storage. Namenode is SPOF and the current machenism of restarting on failure doesn’t provide a quick failover. Some existing solution, like using two Name Nodes in an active/passive configuration, may help achieve faster recover.

# Hardware

Hardware specification relies on the technologies we use, which is HDFS, Spark, YARN and Oozie in our case. It’s also based on the data amount of data, which is 200GB (70000 files) per day. Here is list of out servers, with the assumption that HDFS is outside the system:

* 1 Oozie server: 4 vCPUs, 8GB memory, 500GB Disk.
* 1 Master node: to run Yarn master node and HDFS Name node. Specs: 16 vCPUs, 32 GB mem, 500GB Disk.
* 10 Yarn slave nodes: where to run Spark Executors. Spark will run most efficient if all data can be loaded into memory, roughly 200GB / 10 => 20GB, plus more mem for OS and other daemon like Yarn’s node manager and app master. Specs: 8 vCPU, 32GB mem, 500GB disk.

# Source code

Please find source code in folder telecom-data-pipeline.

Please find submitting instruction under telecom-data-pipeline/README.md.

The code for transformer app is in the class CallHistoryTransformerApp.

The code for aggregator app is in the class CallHistoryAggregatorApp.