UNIVERSITAT POLITÈCNICA DE CATALUNYA

BDM PROJECT 1 REPORT

Landing Zone
Big Data Management

Broniewski, Adam
Kylynnyk, Vlada

Supervisor

Nadal, Sergi

Universitat Politècnica de Catalunya
April 2022

Table of Contents

Tables

Figures

A Project Overview

The business idea behind this data pipeline is called IdleCompute. It is a crowdsourced data analytics framework that connects tenants[1] with a need for computational resources to suppliers[2] willing to sell or donate their idle computer time.

The idea is to accept a tenant's data for analytics, and then complete the analytics in a distributed network. The proof of concept (POC) for this business idea will be executed in two streams completed in teams of two:

1. Matching and scheduling tenants and suppliers[3]
2. Storing and distributing tenant data for analysis

The scope of this project is focused on storing and distributing tenant data for analysis. The business idea is for a tenant with a data analysis task to provide a dataset and select some basic options like which analysis type they want, and which attributes should be used. The analysis would then be complete on their behalf using pre-made machine learning code flexible for any data analysis task. The data analysis modelling that will be completed for the initial POC will be a Complementary Naïve Bayes Classifier.

This report will provide an overview and justification of the architecture used to achieve the purpose. All supporting code can be found in the github repository[4] and in the appendix of this report. A full graphical representation of the data landing process is mapped out in Appendix A.

B Data Sources

Data will originate from tenants that upload their data for analytics through a custom, project specific website. The data types accepted for the initial POC will be json and csv. The two sources being used are a sample from Kaggle.

- CSV Data Source: Loan Default Dataset [1]
- JSON Data Source: News Category Dataset [2]

Table B-1: Extract of header and first line of data sources

CSV Data Source - Example	ID,year,loan_limit,Gender,approv_in_adv,loan_type,loan_purpose,Credit_Worthiness,open_credit,business_or_commercial,loan_amount,rate_of_interest,Interest_rate_spread,Upfront_charges,term,Neg_ammortization,interest_only,lump_sum_payment, 24890,2019,cf,Sex Not Available,nopre,type1,p1,l1,nopc,nob/c,116500,,
JSON Data Source - Example	{"category": "ENTERTAINMENT", "headline": "What To Watch On Hulu…", "authors": "Todd Van Luling", "link": "https://www.huffingtonpost.com/...", "short_description": "You're getting a recent...", "date": "2018-05-26"}

[1] Tenants can be individuals or organizations such as non-profits, businesses, or research centers
[2] Suppliers can be individuals with home computers or organizations with any size servers
[3] Matching and scheduling proof of concept is being completed by project team BDMA11-B1 (Tejaswini Dhupad and Chun Han Li)
[4] Github repository: abroniewski/IdleCompute-Data-Management-Architecture

C Data Collectors

The data collector is the IdleCompute website[5] that users visit to request data analytics. The data would be uploaded along with some information about the project and timing needs. The project metadata and timing needs are managed by the workflow of project BDMA11-B1. This project is focused on the storage and eventual analytics of the dataset that is uploaded.

The IdleCompute website is hosted on the same server being used for the landing zone and data storage. When a tenant uploads their dataset for analysis it is uploaded directly to the temporal landing zone.

To fully develop the POC the IdleCompute website can be implemented quickly as a Google Site with data getting uploaded into GoogleDrive. Alternatively, an existing paid domain[6] can be used with an additional upload page added. An API would need to be created to link from the website to the UPC VM or local machine as a permanent landing zone.

As part of the upload process, the data collector will use a timestamp, user attributes, and a user selection of task type to rename the dataset (see Table C-1). There will only be a single data analysis type available for the initial POC.

Table C-1: Metadata collected during file upload

Attribute	Source	Format
Timestamp-Year	Timestamp during upload	YYYY
Timestamp-Month	Timestamp during upload	MM
Timestamp-Day	Timestamp during upload	DD
Unique user id	Login credentials	ABC123
Unique task id	Autogenerated incremental id for each upload	001
Analysis type id	Selected by user during upload	AB12

The dataset will be named *YYYY-MM-DD-UserId-TaskID-TypeID.extension*

D Temporal Landing Zone

Each raw file uploaded from the website will be stored in its native format following the structure shown in the figure below. Raw data will remain in the temporal landing zone until the data analysis task is complete and the user receives results. The data will be removed from this landing zone 7 days after the client receives their analysis results.

[5] IdleCompute website is not currently setup, but this is the assumed setup that would be used.
[6] https://adambron.com is a website developed by Squarespace with unlimited storage space.

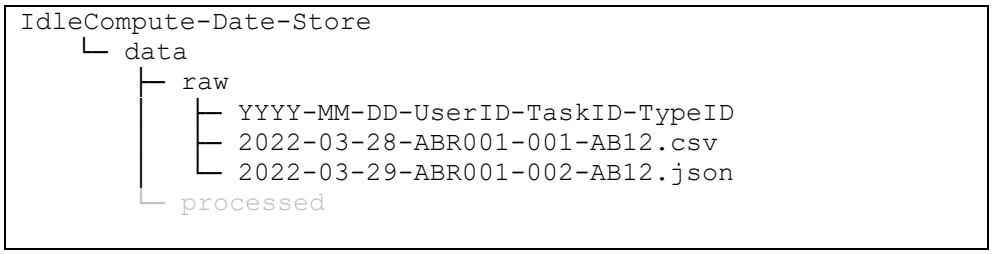


Figure D-1: Temporal landing zone data structure

A.1 File Structure

Given the information known about our current target customers, average data analysis projects have a dataset of between 1GB - 1000GB [3]. The project will plan for typical sizes to be around 100GB - 500GB in size. It is possible a future iteration sees much larger data sizes depending on how the business idea evolves.

The number of supplier nodes and amount of time we will have on each node for parallelized processing is, at this time, unknown. Even though our data sets are not extremely large, a distributed file system is essential for future scaling as well as to manage different data types (structured, semi-structured and unstructured) clients will be providing in the future. A distributed system will optimize our ability to distribute file chunks and complete parallel processing of large amounts of heterogeneous data on different machines. Due to the nature of work, we would need only to read the data, perhaps several times. HDFS is ideally suited for write-once and read-many time use cases. Also

Hadoop Distributed File System (HDFS) will be used as the file system.

A.2 File Format

The storage layout that will be used will be a hybrid layout. This format was selected to match the initial POC modelling requirements (see Project Overview), and to remain flexible for future modelling options as the business scales. The specific file type chosen is parquet.

Using the Naïve Bayes Classifier as an example, we know our tenants will provide labelled data with multiple attributes. This means we will need to split the data vertically between modeling attributes and the labelled data to generate training, testing, and validation data sets. Furthermore, using cross-validation, the data sets will be split horizontally as well into multiple subsets for training and testing. The map reduce framework will process each partition of the file separately, and each partition will be a different partition of the parquet file.

We currently assume that a client will upload a dataset with a combination of attributes that will or will not be included. Parquet will make it possible to employ column pruning to only access the attributes necessary for modelling without performing unnecessary disk reads. It also provides storage of basic statistical information that can help in filtering records while reading [4], which may be beneficial for some distributed ML algorithms. The primary partitions would be the

attributes selected for modelling, the labelled data, and the attributes not needed for modelling. Replication will remain at the default of 3 and is being used for contingency in case of issues with the file. If there is an added benefit of replication for use with the Naïve Bayes classification algorithm, this can be increased. Additional work still needs to be done to understand exactly how the Naïve Bayes Classifier works in a distributed environment[7], which may change this decision in the final iteration.

Additional considerations are that the file format should be optimized for reading and not writing, as the user will only provide the dataset once. There will not be any appends or modifications to the data once it is loaded. The ability to scan the data is another factor that will be considered after evaluating the Naïve Bayes distributed algorithm against the current file format.

As a columnar storage format, parquet is more efficient than row-based storage when reading a selection of columns, as the data for each column is adjacent. Avro, a row-based storage system, is more optimized for writing than reading, which is not required. Avro provides more flexibility for schema evolution which is also not needed. Parquet is better than Avro for reading a subset of columns.

When comparing between ORC and Parquet the deciding factor is better optimization of Parquet with Apache Spark, which is the currently planned as the primary tool for distributed data analytics. ORC is optimized to work with Apache Hive, which is geared towards a distributed database system.

E Data Persistence Loader

Given the small amount of metadata needed to execute data analysis on a given dataset, the persistent loader will store metadata through a combination of filename and folder structure. If the additional metadata needs arise, a document store will be considered, as websites tend to store information in json format. The full data landing process is mapped out in Appendix A.

The data persistence loader is implemented in a python function that creates directories based on the metadata included in the dataset filename. The filename is stripped of its timestamp information and renamed with only the UserID-TaskID-TypeID. The year (YYYY) and month (MM) will be used to create directories where the data will be stored.

While parsing the filename, a function is implemented to convert the files from json or csv to parquet format.

F Persistent Landing

Data converted into parquet format will be inserted into the data/processed directory based on the year and month the dataset is submitted. The original, unchanged file will also be moved into the same directory as a backup. This backup is maintained in case additional intervention is required so that the client does not need upload data

[7] Apache Spark will be used to execute the modeling and analysis. MLlib Guide available here.

again if there is a failure during replication. The data structure used in the persistent landing zone is seen in Figure F-1.

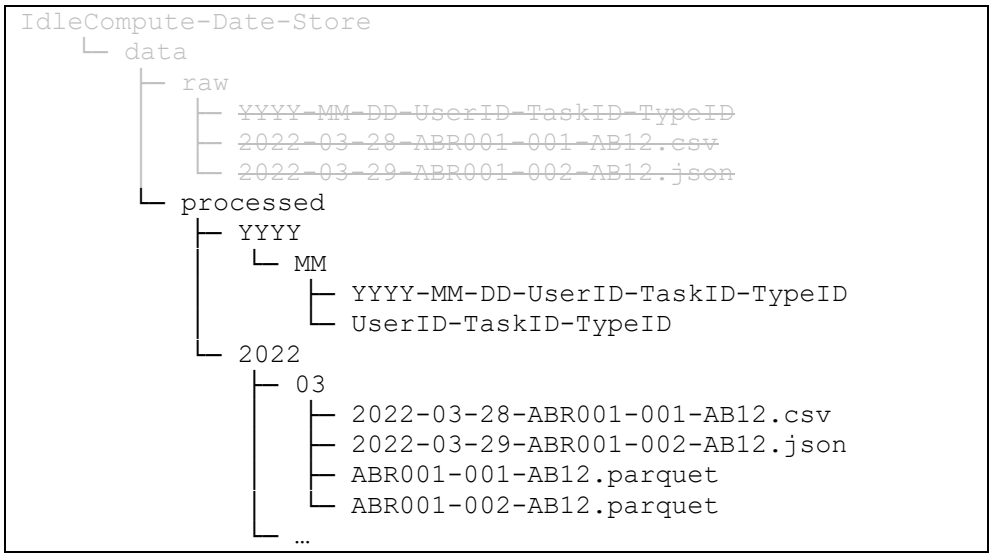


Figure F-1: Persistent landing zone data structure

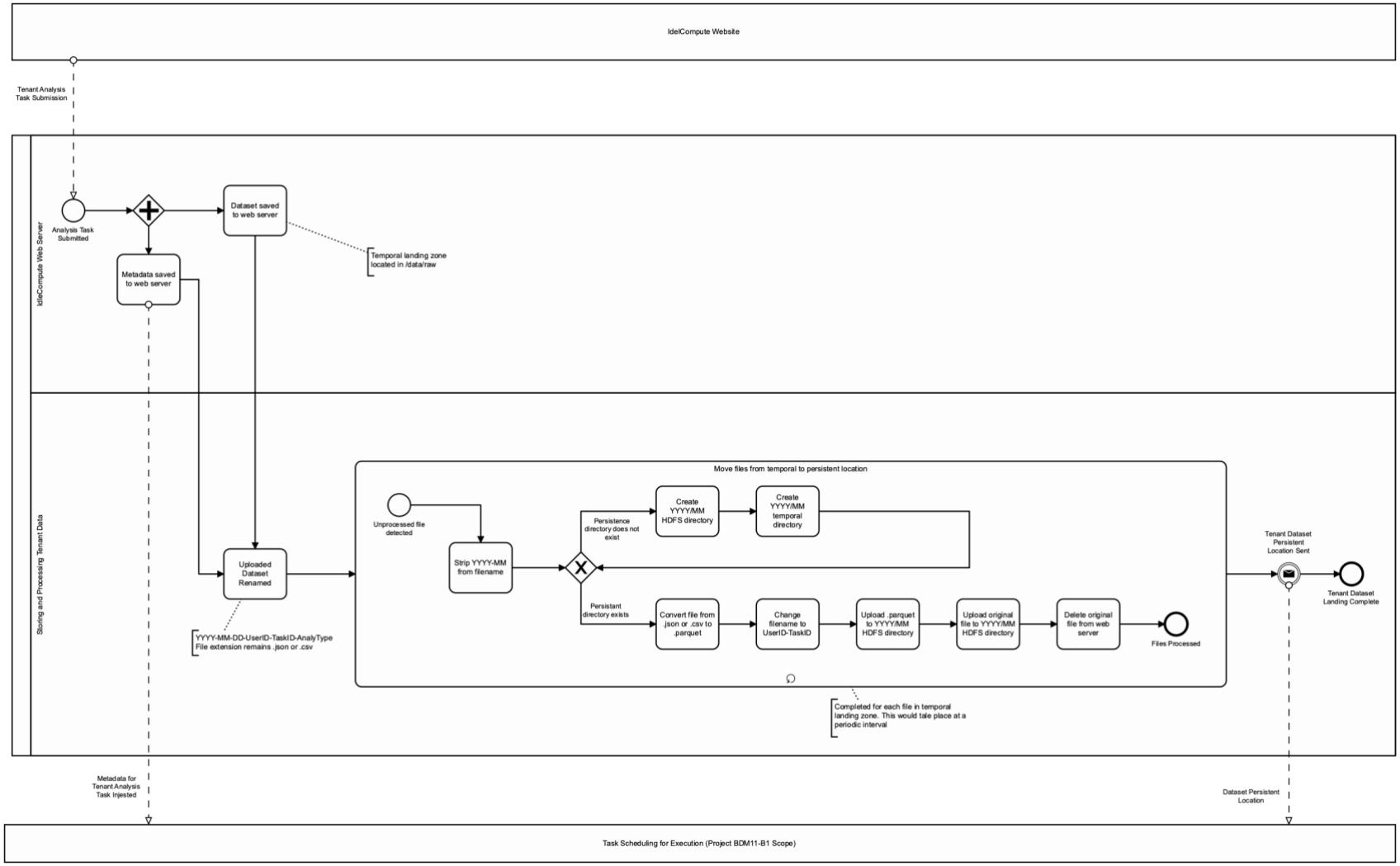
G Data Logging

Data logging will be implemented as part of Project 2, however a placeholder for data logging activities can be seen in the script for the landing zone. The logging activities will include a timestamp for the start and end of each component of the data transformation along with directories of where things are moving from and to.

The intent of the data logger is to facilitate error checking and as a data quality check to ensure all movement of files and transformations are occurring as expected. The data log will be implemented using pythons logging library[8].

[8] Python's Logging HOWTO documentation describes the use of different logging levels for debugging, info, warnings and errors that can be used to output messages in the terminal and save data to a logging file.

Appendix A Landing Zone BPMN Diagram



Appendix B Landing Zone Script

```
import os
from os.path import join
import re
from pyarrow import json, csv
import pyarrow.parquet as pq
from hdfs import InsecureClient
from tqdm import tqdm

TEMPORAL_DIR = '../data/raw'
CONVERTED_DIR = '../data/processed'
HDFS_DIR = './data/processed'
client = InsecureClient('http://10.4.41.37:9870', user='bdm')  # this IP should be changed to your personal VM

def convert_to_parquet(file_type, in_directory, in_filename, out_directory, out_filename):
    '''
    This function will take an input file in the form of CSV from a given directory,
    convert the file to a parquet, and place the file in a directory specified in parameters.

    :param file_type: extension, json or csv
    :param in_directory: directory where the CSV file exists
    :param in_filename: filename (including extension) that will be converted into parquet file
    :param out_directory: directory where the parquet file should be placed after conversion
    :param out_filename: filename that will be given to converted parquet file
    :return: None
    '''
    if file_type == "json":
        table = json.read_json(f'{in_directory}/{in_filename}')
        pq.write_table(table, f'{out_directory}/{out_filename}')
    elif file_type == "csv":
        table = csv.read_csv(f'{in_directory}/{in_filename}')
        pq.write_table(table, f'{out_directory}/{out_filename}')

# TODO: add log code for each LOG location below.
def create_persistent_directory():
    hdfs_existing_directory_year = client.list(HDFS_DIR, status=False)
    # LOG: Batch landing attempt timestamp
    for filename in tqdm(os.listdir(TEMPORAL_DIR)):  # iterate over all files in directory DIR
```

```
        if not filename.startswith('.'):  # do not process hidden files that start with "."
            metadata = re.split('[-.]', filename)  # splits the filename on '-' and '.' -> creates a list
            file_directory = f"{CONVERTED_DIR}/{metadata[0]}/{metadata[1]}"  # uses YYYY/MM subdirectory name
            new_filename = f"{metadata[3]}-{metadata[4]}-{metadata[5]}"  # new file name will be userID-taskID
            if metadata[0] not in hdfs_existing_directory_year: # creates directory if doesn't exist. Check year
                # LOG: New year + month directory creation attempt
                client.makedirs(f"{HDFS_DIR}/{metadata[0]}/{metadata[1]}", permission=None)
                # LOG: directory success + output directory created with client.resolve() command
            hdfs_existing_directory_month = client.list(f"{HDFS_DIR}/{metadata[0]}", status=False)
            if metadata[1] not in hdfs_existing_directory_month: # check if month exists
                # LOG: New month directory creation attempt
                client.makedirs(f"{HDFS_DIR}/{metadata[0]}/{metadata[1]}", permission=None)
                # LOG: directory success + output directory created with client.resolve() command
            if not os.path.exists(file_directory):  # creates the directory if it doesn't exist
                # LOG: temporal directory creation attempt
                os.makedirs(file_directory)
                # LOG: directory success with actual dir created
            file_type = metadata[6] # will be passed as parameter to convert_to_parquet
            persistent_file_location = f"{HDFS_DIR}/{metadata[0]}/{metadata[1]}"
            convert_to_parquet(file_type, TEMPORAL_DIR, filename, file_directory, new_filename)
            # LOG: file conversion success. From location, to location, file size
            client.upload(persistent_file_location, f"{file_directory}/{new_filename}") # upload parquet
            # LOG: parquet file upload success. From location, to location, file size
            client.upload(persistent_file_location, f"{TEMPORAL_DIR}/{filename}") # upload original file
            # LOG: original file upload success. From location, to location, file size
            os.remove(join(TEMPORAL_DIR,filename))

def delete_all_data_in_hdfs():
    client.delete(HDFS_DIR)
    client.makedirs(HDFS_DIR)

if __name__ == '__main__':
    delete_all_data_in_hdfs() # this function is included for testing
    create_persistent_directory()
    # LOG: Batch landing complete timestamp
```

Appendix C Virtual Machine Setup Instructions

```
Go to this site:
    https://upclink.upc.edu/vdesk/webtop.eui?webtop=/CommonWebtop_upclink.up
    c.edu&webtop_type=webtop_full

Click on:
    VPN UPC Link
    Start the VPN

Go to this site:
    https://virtech.fib.upc.edu/
Open Nebula login with:
    user: masterBD12
    password: asY76fkG12

Once in the dashboard, setup VM (press green +, name machine, keep default
settings)
    password: bdm

In local terminal: to connect and interact with VM. IP address can be found
in the OpenNebula dashboard. This will turn this instance of your local
terminal into the VM terminal
    ssh bdm@host_ip_address

    ssh bdm@10.4.41.37
    passowrd: bdm

In VM terminal: Start HDFS
    /home/bdm/BDM_Software/hadoop/sbin/start-dfs.sh

In local internet browser: Check HDFS status in browser (safari)
    http://10.4.41.37:9870

In VM terminal: Create HDFS directory. These commands go into your VM
terminal
    ~/BDM_Software/hadoop/bin/hdfs dfs -mkdir /user
    ~/BDM_Software/hadoop/bin/hdfs dfs -mkdir /user/bdm
    ~/BDM_Software/hadoop/bin/hdfs dfs -chmod -R 777 /user/bdm/

In VM terminal: map the HDFS directory to bash so that you can run commands
using just "hdfs" instead of the whole directory
    echo 'export PATH="$PATH:~/BDM_Software/hadoop/bin"' >> ~/.bashrc && .
    ~/.bashrc

Testing a file import
    In Python: import the following library
        from hdfs import InsecureClient

    In Python: Create a connection variable to HDFS. The IP will vary
    depending on your VM.
        client = InsecureClient('http://10.4.41.37:9870', user='bdm')

    In Python: Upload file from local to VM. The '.' represents your
    user/bdm HDFS directory.
        client.upload('.', '../data/processed/[filename]')

    In VM Terminal: Check to see if you file got moved
        ~/BDM_Software/hadoop/bin/hdfs dfs -ls /user/bdm
```

Appendix D Works Cited

[1] M. Y. H, "Kaggle," [Online]. Available: https://www.kaggle.com/datasets/yasserh/loan-default-dataset?select=Loan_Default.csv. [Accessed 31 March 2022].

[2] R. Misra, "Kaggle," [Online]. Available: https://www.kaggle.com/datasets/rmisra/news-category-dataset. [Accessed 28 March 2022].

[3] G. Piatetsky, "Poll Results: Where is Big Data? For most, Largest Dataset Analyzed is in laptop-size GB range," KDnuggets, 2015.

[4] S. N. Alberto Abelló, Big Data Management, Barcelona: Database Technologies and Information Management (DTIM) group Universitat Politècnica de Catalunya (BarcelonaTech), 2022.