Predicting Job Idle Time before Execution using Machine Learning

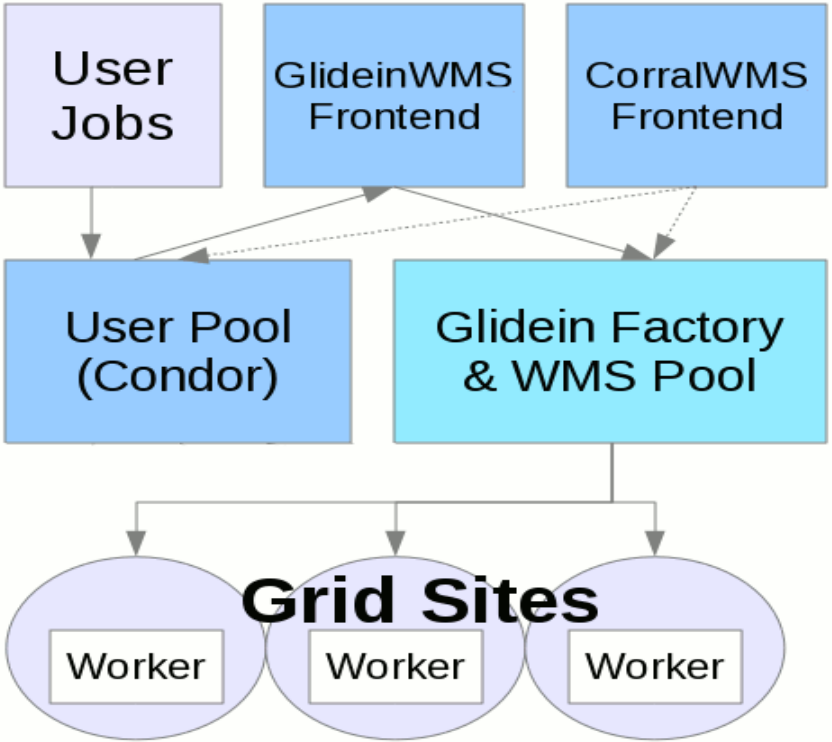
Mentee: Francisco Laris
Mentor: Bruno Coimbra

Abstract

Abstract: When jobs are submitted through HTCondor they are first submitted and added to a queue before beginning execution. This idle time can vary wildly from job to job resulting in some jobs start running later than expected. Machine learning coupled with data from Landscape on the jobs status and submission time, should give us means to create a model for predicting which jobs take longer to get out of the queue and possibly why.

Overview

1. Introduction
2. HTCondor
3. glideinWMS
4. Machine Learning
5. Limits



Method

- Train models using the data obtained from landscape.
- Select the best one for our purposes and make sure no overfitting is occurring.
- Perform a periodicity analysis using fourier transforms to check for emerging patterns over time.
- Check how accurate the predictive capabilities of our model are.

Data extraction

- Using elasticsearch_dsl we are able to extract random jobs from 2021.
- We specify a seed to make sure the code can be rerun with the same results.

```
search_history = Search(using=connection, index="fifebatch-history-*")\
    .params(scroll='25m')\
    .query('range', QDate={'gte': start_time, 'lt': end_time})\
    .filter('exists', field='JobsubJobId')\
    .filter("exists", field="QDate")\
    .filter("exists", field="JobCurrentStartDate")\
    .query('function_score',functions=({'random_score': {'seed': 60}}))
```

Data Cleanup

- Due to the nature of the data on Landscape each job has different parameters. This makes cleaning up the data so we don’t have null values necessary.

- Once undesired columns (parameters) are dropped we need split the DESIRED_USAGE_model parameter into 2 (ONSITE and OFFSITE) in order for the model to be able to access the information.

- These 2 columns contain either 1 or 0 corresponding to true or false.

Data Cleanup

- After doing this we generate our target variable “Total Queue Time”, which is defined as the difference between Job Current Start Date and QDate.

```
data["TotalQueueTime"] = data["JobCurrentStartDate"].astype(int) - data["QDate"].astype(int)
```

Data Cleanup

- A small amount of jobs have null values in DESIRED_USAGE_model.

- To fix this we try to reverse engineer what the usage model probably was by getting the mean total queue time for every option and substituting the null value with closest total queue time for it.

Data Cleanup

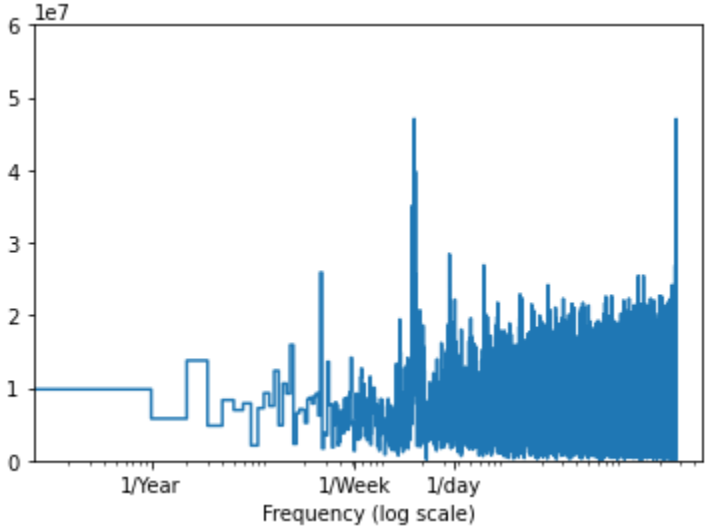
- We then remove outliers to better train the models.

```
def outliers(df, variable):
    upper_limit = df[variable].mean() + 3 * df[variable].std()
    lower_limit = df[variable].mean() - 3 * df[variable].std()
    return upper_limit, lower_limit
```

Data Analysis

- In order to account for possible cyclical patterns in the data we conduct a periodicity analysis using fourier transforms.

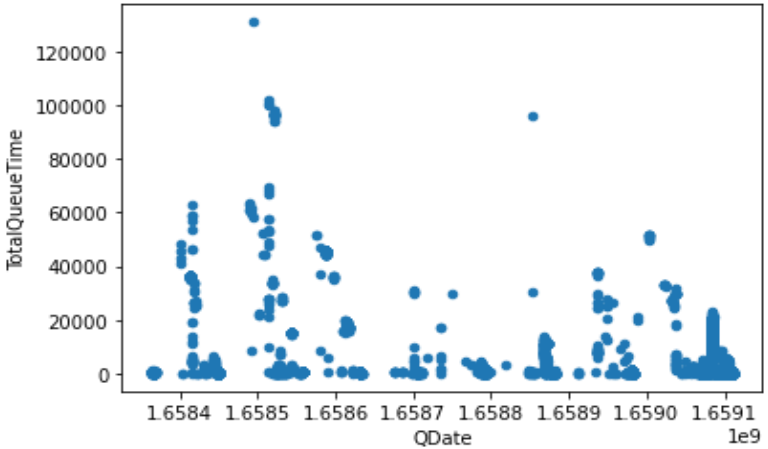
- The resulting graph shows that some cyclical behaviour happens at a frequency lower than 1 per day.



Data Analysis

- Our decision tree regressor shows a lot of promise during training, but once in testing it performance becomes by far the worst out of all models.

- This is a classic example of overfitting. To see why this is happening we may want to look at the following graph



Data Analysis

- Finally taking into consideration all of our variables we are able to get the R^2 value of 0.34 in the Linear regressor, with the other models proving worse due to overfitting.

- This means that without looking at the server and only using information from submission we are able to account for about ⅓ of total variation in Queue time.

- This number might be further improved by looking at more variables