International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 05 | May 2020 www.irjet.net p-ISSN: 2395-0072

Activity Duration Prediction of Workflows and analyses of Various Machine Learning Approaches

Vijay Ravi[1], Dr. Sharvani G. S.[2]

[1]Department of Computer Science and Engineering, R.V. College of Engineering, Bengaluru, India
[2]Associate Professor, Department of Computer Science and Engineering, R.V. College of Engineering, Bengaluru, India

---

Abstract - *Traditionally, workflow (a sequence of unit tasks) progress times have been based on the number of tasks completed at any point in time. This paper attempts to look into Machine Learning techniques to improve calculation of such workflow times. Each workflow is assumed to run on a remote server, and each unit task's time for completion largely depends on three important associative variables: past history durations, RAM, and network speeds. The paper discusses multilayer perceptrons, multilinear regression, and a naive approach to address the problem.*

Key Words *– workflow, unit task, RAM, multilayer perceptrons, multilinear regression*

1. INTRODUCTION

The problem of predicting task times has useful applications in various domains of scientific research. An end user is pestered when they have to run a certain workflow on a remote server, but are unsure period of time they have to wait for before they can carry on with their regular activities. Smarter systems for workflow duration prediction are definitely the need of the hour, and this paper is an attempt to model one such system.

1.1 Services and Components used

Python libraries: The project makes use of the opensource Scikit-Learn library, Tensorflow toolkit for regressing/ run neural net over sample data collected by running various workflows over a number of various systems
Dockers: The service used to run workflows on remote server is one built with the microservice architecture, for efficient, quick return turnaround times
Kubernetes: Container management has been mainly via Kubernetes
HTML, CSS: The UI interface to display the predicted task times, and consequently the workflow times, is developed using HTML and CSS
GoLang: The microservices to run workflow on remote servers are built in the Go language

1.2 Existing Systems

Robust systems built upon the microservices architecture have already been studied and some even suggest Java or Javascript as the clear winner to building such application [1]. In one of these extensively researched papers, the authors suggest Golang to be a fluid choice to incorporate various use cases [2]. The algorithms discussed in this paper have been experimented on an industry deliverable, and GoLang has been the tool of development of the deliverable.

Various Machine Learning approaches have been thoroughly research upon: multivariate regression [3], multiple linear regression [4]. This paper takes a look at these approaches to the particular case of predicting workflow times modelling the problem in three variables. Workflow time prediction, especially on a remote server, is a relatively unexplored topic, and the discussed methodologies might at the end be of immense use in various other industrial deliverables.

1.3 Scope and Motivation

It is painfully annoying for an end-user to sit and wait until a remote server executes their task and return back to them their result. The user may have all data pertaining to the network speeds at this remote server, its hardware capabilities, its past history of executing the same exact task, and yet the user in most cases sits through the workflow. In extended workflows that take up to hours (installing an OS on a remote server is an example), the problem is intensified, and a solution to such problems is definitely the need of the hour.

This Machine Learning study takes a look at modelling start to finish solutions to the aforementioned problem. The proposed system addresses solutions to the problems ranging from simple to relatively complex.
The proposed solutions can be implemented in any client-server application/ industrial deliverable to improve wait-times and help users organize and plan their order of execution of workflows accordingly.

2. METHODOLOGY

2.1 Architecture

The naïve approach, as a precursor to exploring Machine Learning algorithms, and as a solution, is a very good starting point. We build upon this model to incorporate other algorithms.

A task is assigned with an ID and a float property avgTime (to hold running average time of the task). A bunch of such user defined tasks are woven to construct a workflow, and the client now starts a workflow on the server. Parallelly, a database is necessary for the main program to regularly update the running average for better prediction of workflow progress.

2.2 Naïve Learning of the algorithm

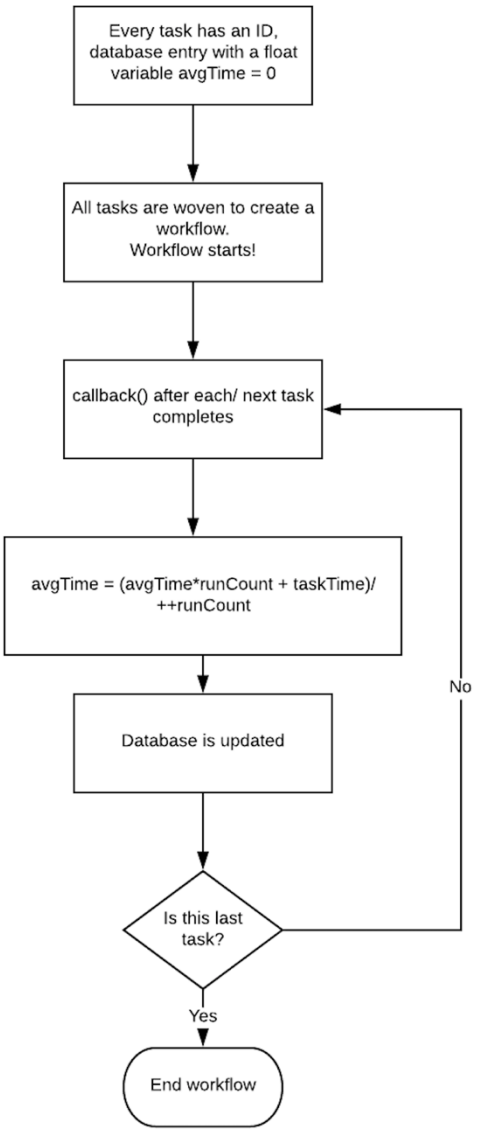

Fig 2.0: Naïve approach and flow chart

This approach updates the avgTime in the callback() function every once a task is completed. A task may fail, in which case, the progress is updated to 100% and the control is returned to the server. This naïve approach is based on calculating the progress as the average task time over the complete workflow time (sum of average times of all tasks from the database)

$$avgTime_{Ti} = ((avgTime_{T(i-1)} * runCount) + taskTime_{Ti})/(runCount+1)$$
$$Progress_{Ti} = Progress_{T(i-1)} + avgTime_{T(i-1)} /workflowTime \quad (1)$$

This solution works better as the number of times the same task is executed increases.

2.3 Architecture to make use of recorded data

A small-scale microservice-architecture based application can make use of the methodology previously described. But as applications scale up and multiple servers are used to execute workflows, it would help to consider task time as average variable and identify a bunch of input variables.

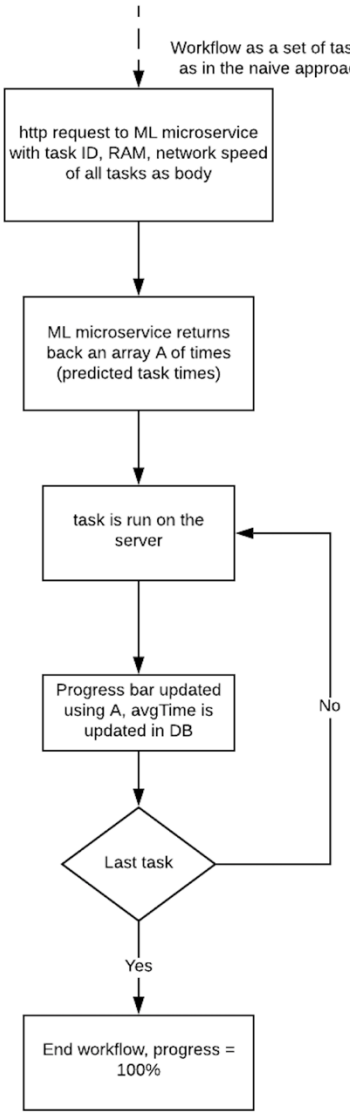

Fig 2.1: Flow Chart for model with an http API for ML inputs

Average time of the task, Random Access Memory of the server, and network speed of the connection to the internet at the server, can be chosen as input variables to model the solution. It is practical to use a server (a microservice that

consults the model given the value of these variables at the server end) that returns the expected time for task completion.

The Machine Learning model can be a simple Multiple Regression fit or a multilayer perceptron fit. The biases and activation functions are chosen to minimize aberrations. Once the model is sufficiently trained with existing data, the model now takes input from remote servers and returns predicted times of tasks of the workflow.

2.4 Perceptron Model

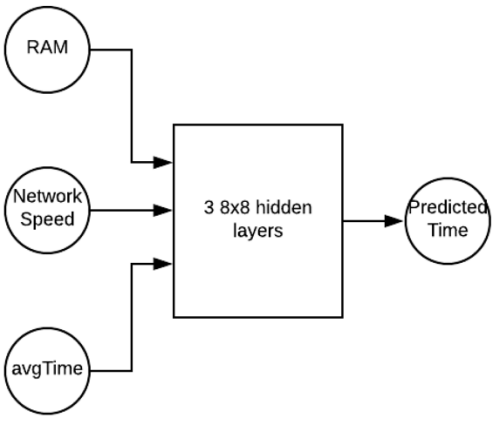

Fig 2.2: Perceptron model for a task of a workflow

A perceptron architecture to predict task times has been shown in the figure above. Three hidden layers with 8x8 connections have been chosen to tackle the problem. Industry deliverables usually have only a limited number of tasks that are ordered in various ways to form numerous workflows. So, it makes sense to have a perceptron designed for every task.

The predicted time is POSTed via http API back to the microservice that made the request in the first place, and the progress is thus calculated.

2.5 Multiple Linear Regression

Another possible architecture to solve the problem is to regress the problem over the three variables, instead of feeding it to a perceptron. The result is, in this case, for all intents and purposes equal to the perceptron way of tacking the problem – for all the three variables seem to be linearly related to the predicted time.

3. IMPLEMENTATION

3.1 Technology Stack

Our system uses an efficient tool stack that has been carefully chosen for full-on delivery and minimal delay.

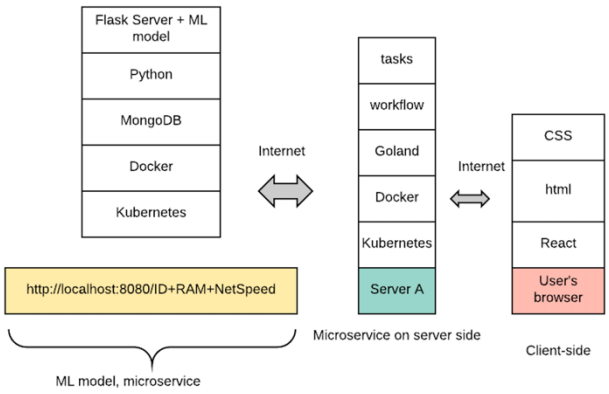

Fig 3.0: Tool Stack for the microservice based solution

Two microservices and a user-interface to control the workflow initiation and termination on the server were designed. The communication between the three entities were largely over the http protocol.

3.2 Naïve Approach

Four tasks were taken for the experiment, and these tasks were stitched into a workflow (a task is a user-defined json input; a workflow is a parsed collation of the json – in Golang). The docker image of the service now runs on the server.

```
},
"Input": {
    "pingCmd": "ping -c 1 127.0.0.1",
    "pingCmd1": "ping -c 8 127.0.0.1",
    "pingCmd2": "ping -c 3 127.0.0.1",
    "pingCmd3": "ping -c 12 127.0.0.1"
},
```

Fig 3.1: Each ping command is a task that will execute on a remote server, the 4 tasks form a cmd-command workflow

Workflow initiation initializes the avgTime of all the four tasks to 0s. Upon sending a http post to the server via the user interface, the workflow (list of cmd commands) is run, and the progress of completion is easily measured using equation 1.

The first time the workflow is run, a 25% progress is seen as each task completes. When executed for a second time, one would expect task A to show a smaller progress when completed, and this is exactly what happens. The naïve approach improves as the number of times the task is run increases.

Fig 3.2: avgTime and runCount as properties of a new task

The MongoDB database holds the average time of execution of all the tasks. The same workflow was executed repeatedly on the server. As expected, the progress values plateaued at

4%, 33%, 13%, and 50% respectively after executing the workflow for five times.

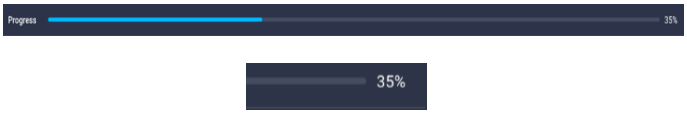

Fig 3.3: UI progress bar showing 35% progress post execution of task 2

3.3 Multiple Linear Regression

A sample of 15 datapoints were taken for each of the four tasks, and the data was regressed over the three previously discussed variables.

1	RAM (in GB)	Network Speed (in mbps)	avgTime (in s)
2	8	200	3.4
3	16	220	1.8
4	4	150	5.6
5	6	176	4.3

Fig 3.4: Datapoints for Task 1 collected by running the task on various sample servers

Each of the four tasks are now regressed (post normalizing) and a model is fit as shown below.

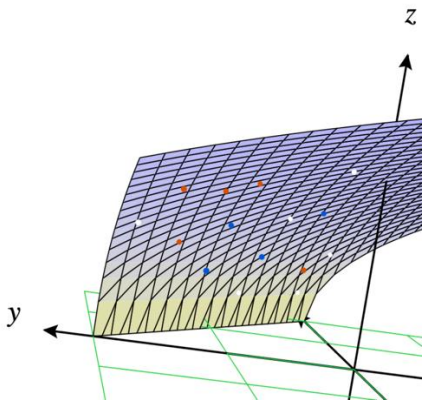

Fig 3.5: Model for task 1 to fit data with normalized RAM and network speeds on the x and y axes respectively: red datapoints lie above the model, white ones, below, and blue ones, on the plane; z-axis gives the predicted time

Post logging into the user interface, the server is claimed and the workflow is initiated. The server now queries with its data the microservice that holds the drained ML model. The ML model parses the information sent from the server, and returns the z-coordinates according to the plane that it had regressed.

A similar model can be trained with a neural network, and this was also essayed.

```
NN_model = Sequential()

# The Input Layer :
NN_model.add(Dense(3, kernel_initializer='normal', input_dim=tf.train.shape[1], activation='relu'))

# The Hidden Layers :
NN_model.add(Dense(8, kernel_initializer='normal', activation='relu'))
NN_model.add(Dense(8, kernel_initializer='normal', activation='relu'))
NN_model.add(Dense(8, kernel_initializer='normal', activation='relu'))

# The Output Layer :
NN_model.add(Dense(1, kernel_initializer='normal', activation='linear'))

# Compile the network :
NN_model.compile(loss='mean_absolute_error', optimizer='adam', metrics=['mean_absolute_error'])
NN_model.summary()

NN_model.fit(X_train, target, epochs=500, batch_size=32, validation_split=0.2)
predictions = NN_model.predict(x)
```

Fig 3.6: A similar model is trained using a 3 x 8 x 8 x 8 x 1 NN in keras

The parsed values are used to predict the estimated time, as in the case of multiple linear regression, and output is sent to the requesting microservice.

4. APPLICATION OF PROGRESS PREDICTING SYSTEMS IN REAL WORLD

In this paper, we have thus far discussed a number of possible solutions. Incorporating these methodologies for workflow-progress prediction in real-life applications can be of huge benefits.

Whether it is a train-booking portal for tickets, or an e-commerce website for checkouts, or a social media platform for content uploads, or a bank website that processes transactions, activities, workflows, and tasks are inherent features.

The following diagram describes a possible sequence of events to incorporating the discussed solutions with real-world applications. The numbers to go with, in the diagram describe the sequence of events to model such a system. The application backend is assumed to be hosted on AWS, a query is further sent to a remote server for workflow execution. The server consequently queries the ML microservice (the model developed) for the estimated time of a particular task in the workflow posted. The ML service returns an expected time back to the server. The server waits until the task is complete and then asks the application to display an update on the User Interface. The User is now informed about the progress of the workflow they initially POSTed. This cycle continues until all the tasks in the POSTed workflow definition are complete.

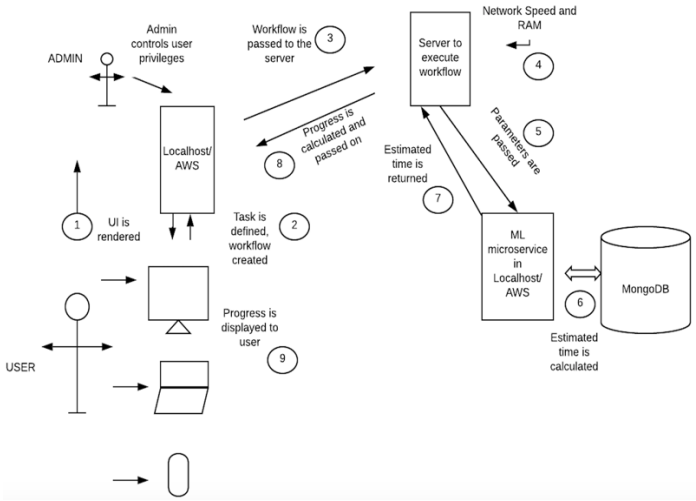

Fig 4.1: Datapoints for Task 1 collected by running the task on various sample servers

5. CONCLUSION

5.1 Unit-Testing

The solutions were put to test to predict task times and progress in various server conditions. The naïve approach was tested by executing different task sequences repeatedly on the same server. The Regression approach, meanwhile, was tested by running workflows on servers with different network speeds and RAM.

	Naive approach	Multiple regression
% workflows that finished +/-5 seconds of prediction	86%	91%

Fig 5.0: Data compiled over executing workflows for 30 times

The User Interface was tested by creating all possible sequences of workflows, and the resulting changes of the progress bar were recorded.

5.2 Future Work / Add-Ons

The dataset that the study bases its results can be broadened. Other solutions like random forests and recurrent neural networks can be explored upon to improve the performance of the algorithms.

6. ACKNOWLEDGEMENT

We would like to acknowledge the support provided by Teaching and Non-Teaching staff of Department of Computer Science & Engineering, RV College of Engineering through required assistance during the research work.

7. REFERENCES

[01] Markos Viggiato, Ricardo Terra, Henrique Rocha, Marco Tulio Valente, Eduardo Figueiredo, "Microservices in Practice: A Survey Study" 6th Brazilian Workshop on Software Visualization, Evolution and Maintenance.

[02] Biradar Sangam. M, R. Shekhar "Building Minimal Docker Container Using Golang", IJERCSE Vol5, Issue 3, March 2018.

[03] E C Alexopoulus, "Introduction to Multivariate Regression Analysis", Hippokratia, 2010 Dec; 14(Suppl 1): 23-28.

[04] Gülden Kaya Uyanik, Nese Güler, "A Study on Multiple Linear Regression Analysis", Procedia – Social and Behavioral Sciences 106:234—240, December 2013

[05] Robert Heinrich, André van Hoorn, "Performance Engineering for Microservices: Research Challenges and Directions" 8th ACM/SPEC International Conference on Performance Engineering (ICPE 2017)

[06] Babak Bashari Rad, Harrison John Bhatti, Mohammad Ahmadi, "An Introduction to Docker and Analysis of its Performance", IJCSNS, VOL.17 No.3, March 2017.

[07] M.R.M. Veera Manickam, M. Mohanapriya, Sandip A Kale, "Research study on applications of artificial neural networks and e-learning personalization", International Journal of Civil Engineering and Technology 8(8):1422-1432, August 2018

[08] Flask framework (open source) https://flask.palletsprojects.com/en/1.1.x/

[09] Tensorflow framework (open source) https://www.tensorflow.org/

[10] Golang framework (open source) https://golang.org/

[11] Dockers (open source) https://www.docker.com/

[12] Kubernetes (open source) https://kubernetes.io/