Estimating time of a CAL module execution

Statistic under AI

Jan Bielecki

March 14, 2021

March 14, 2021

1 Introduction

Computation Application Language (CAL) is designed to write large scale application in due to perform big data processing. Each CAL program consists of modules that are separetely executed in sequence (with parallelization possibility) on virtual machines and sending the results futher to next module of an application. CAL language is a part of the system developed under the Baltic LSC[1] project.

During the data processing within the application run, each module is executed with some input data. It could be different kind of data e.g. data frame or set of images. The execution of a module is invoke on a virtual machine with limited resources (RAM, mCPUs, GPUs). The details of an input data and the execution environment resources will be used to estimate the overall application execution time and price on a concrete cluster.

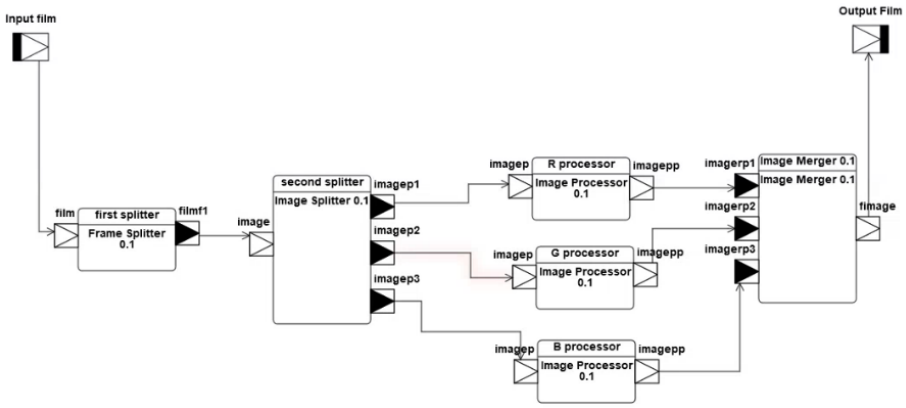
Baltic Large Scale Computing (BalticLSC[2]) project is using CAL language to perform data processing tasks. Each task is an execution of a CAL application. Each module can be used in many CAL applications. We can say that a single module is a one block (commonly as a docker image) and an application is the schema of executing concrete set of blocks. Figure 1 shows the example application that consits of a few modules to perform some image processing. Some of the modules (R, G and B *processors*) can be executed in parallel.

The aim of this project is to estimate the module execution time based on input data and limited resources of the execution. The choosen approach is to create machine learning models using historical data of the module executions times.

The WCET[1] execution time is a crucial feature of a real-time systems[6] when response should be guaranteed within a specified timing constraint or system should meet the specified deadline. In this project we do not strictly focus on providing an estimation of maximum execution time (WCET). We will

[1] Worst-case execution time - the maximum amout of time that the execution can take.

Figure 1: The example of an image processing application written in the CAL language.



try to estimate the average-case execution time (ACET) which is absolutely enought for this use case requirements.

2 Model features

To predict time of module execution we create a machine learning model for each module. With each execution of the module, within some application, we will get another data point to train our model. We will use the following features (explanatory variables)[2] as an input data for the model:

1. mCPU limit - called *mili cores* - the fraction of a physical CPU used to carry out the module execution,
2. total size of an input data,
3. max element size of an input data (if the input data is a set of files type it is the biggest part of data to be processed),
4. avg size of an input element,
5. number of input data elements.

This last three features makes sense only if the input data is a set of files. Otherwise, if the data is just a single file (e.g. data frame), the features set should be reduced to just the first two elements from the list above.

Obviously, our dependent value (that we are going to estimate) is an execution time of a module. The example data frame that will be used to train and validate our models is presented in the table 1.

Module ID	mCPU	total [KB]	max [KB]	avg [KB]	#	time [ms]
1	1.1	12414	1341	871	21	7813
1	0.5	12414	1341	871	21	12406
1	3.6	54001	2190	891	82	9017
...	...	...	...	...	...	...

Table 1: The example data frame for models training and validations.

[2]As a docker container that will be used to execute a module do not use SWAP memory, RAM is not colerated with execution time. In this program we will not use modules with GPU support. Summarazing, the GPUs and RAM resources limits are not taken into account for modeling.

3 Algorithm

As a machine learning algorithm for our project we will use *Epsilon-Support Vector Regression*[5] (SVR) - it is based on Support Vector Machine (SVM, orginally named *support-vector networks*[3]).

We will use SVR model with a RBF kernel[4] which is the most known flexible kernel and it could project the features vectors into infinite dimensions. It uses Taylor expansion which is equivalent to use an infinite sum over the polynomial kernels. It allows to model any function that is a sum of uknown degree polynomials.

4 Testing modules

In the project we will use the modules described below to perform models evaluations:

1. Face recogniser - takes set of files (images) as an input data and marks faces of people on each image.
2. ...

5 Training and test datasets

6 Models learning and validation

7 Conslusions

References

[1] BalticLSC: main webpage of the project, visited 14.03.2021, `https://www.balticlsc.eu/`

[2] BalticLSC: BalticLSC Admin Tool Technical Documentation, Design of the Computation Application Language, Design of the BalticLSC Computation Tool, visited 14.03.2021, `https://www.balticlsc.eu/wp-content/uploads /2020/03/O5.2A_O5.3A_O5.4ABalticLSC_Software_Design.pdf`

[3] Support-Vector Networks by Corinna Cortes , Vladimir Vapnik, 1995 `https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.9362`

[4] Radial basis function kernel, Wikipedia, visited 27.12.2020 `https://en.wikipedia.org/wiki/Radial_basis_function_kernel`

[5] Support Vector Regression, Dr. Saed Sayad, visited 17.01.2021
https://www.oreilly.com/library/view/introduction-to-machine/9781449369880/ch04.html

[6] Estimation of the execution time in real-time systems, 22.01.2016
https://link.springer.com/article/10.1134/S0361768816010059