HIDL Project

Name: Manikya Swathi Vallabhajosyula

# Title:

Predicting near optimal execution settings (starting with execution time) for heavy resource consuming jobs to get optimal performance (This module is part of Building an Intelligent Assistant for Computational Scientists Project)

Motivation:

The availability of high-performance computing (HPC) cyberinfrastructures (CI) like Ohio

Supercomputer (OSC ) or Titan enables scientists to perform computationally intensive experiments (even at exa-scale). Researchers from different domains share these resources to perform tasks ranging from genome sequencing to mapping climate change patterns. Since these resources are allocated per hour and are charged based on a submitted job's processing and memory requirements, each researcher must finetune their application/workflow to get the best results by optimally allocating the resources and consuming them efficiently. They run small-scale simulations of the workflow to identify potential tools along with their settings. Though most of this work is redundant among researchers of the same domain (sometimes across domains when standard tools are used), the ideal performance optimization information is not shared. Information on the application settings could be found in domain specific journals or related published work. However, it is time-intensive to do a thorough literature review on all the relevant work and it is time-intensive to replicate their workflow on an CI different than that presented in these publications. Also, shared resources like the OSC have system and tool traces which could act as training material for a recommendation engine that predicts near-optimal settings cater to targeted tools for a researcher.

**Note:** In the current Project - I explored what information could be pulled readily from logs (at user end), input data and additional profiling data. I explored on how this information could be used to predict probable execution time, given system resource limitations. [i.e., the probable time the submitted job might run - hence the constraint systems should be blocked for that amount of time]

# Problem Statement:

A researcher spends most of their time fine-tuning the application hyperparameters (say how deep should a NN be?, how much should the batch size be? Or What is the better trim quality to obtain a higher contigassembly?) to obtain best accuracy of their model. But they fail to consider if the resources they are requesting from the supercomputer (like CPU/GPU cores, RAM) are being optimally (or fully) being utilized by their job. As a result, they end up blocking more resources than their job’s requirement (hence pay more than required) and end up with jobs with higher execution time (as all the processors requested per node not kept to use) [*I heard that this is a crucial issue with people investing on 3rd party cloud systems alike Aws or Azure*]. Through our research, we hypothesize that we can build an artificially intelligent tool that could provide tailored recommendations (**this project explores only predicting wall-time/execution time**) to the end user based on their workflows and available resources.

# Key Contributions:

1. Setting an environment to configure co-design frameworks and profilers
2. Setting up a framework to generate training data for a given target application and Sample data (or even original data - with a little domain information)
3. Identifying feature categories - and having a clear distinction of how to capture them (i.e. w.r.t to the type or application - computation/IO/communication intensive)

# Introduction:

Below is the pipeline, I propose to predict wall-time from several features collect from CI (or a simple Linux box/VM)

1. The applications are purely treated as black box applications - managed with few hyper-parameters to reach user’s desired results. They are:
   1. Gray-Scott[1] - a simple chemical diffuser simulator. Reaction and diffusion of chemical species can produce a variety of patterns, reminiscent of those often seen in nature. The Gray Scott equations model such a reaction. For more information on this chemical system see the articles "Complex Patterns in a Simple System” and "Pattern Formation by Interacting Chemical Fronts," [2]. This is highly IP intensive - as it writes these reactions to files.
   2. Trimmomatic - a pre-cleaning tool used to clean the raw SRAs (sequence Raw Archives) [gene sequenced from machines like ILLUMINA[3]]. The output of this phase enhances the quality of the assembly generated by the gene assembler tools. This is Both CPU and IO intensive.
   3. DNNs (TensorFlow - VGG16, ResNet50 & InceptionV3] - Models run on single node for the current project (could be extended to multi-node - see Future Work Section). Compute Intensive (If multi node is included in future compute communication and OpenMPI calls could be added to features)
2. The features I identified to make the prediction of execution time are categorized into three classes:
   1. Input Features- The features w.r.t to the input provided to the target application.
   2. System Specific Features - The set of features that explain the computation power of the target systems where the training data is generated. They describe the execution environment.
   3. Application Features - These features that are fine-tunes to get user desired results from the application. Can also call these as application hyper-parameters.

These features are the ones I capture on the next step - generating training data[Model details could be found in section **Methodology**]

1. Since, there isn’t any training data available with me, the first challenge is to generate training data w.r.t the application I choose (as target applications). It would be practically impossible to generate 1K samples with original data and executions without losing resources (in other words manual fine tuning would be less expensive and faster). So that would be expensive to generate the training data at full-scale. So to reduce the cost induced by training data generation, I propose to take a sample representation of the original data set(1/100th or 1/1000th) and run the application on a demo environment (say a Linux VM and on OSC but with limited resources). Then capture the required features. The training data is generated on two systems based on the application’s requirements:
   1. OSC (Pitzer and Owens - Single Node - GPU and CPU)
   2. LINUX VM (Using Parallels on Mac) - varying Nodes and Memory

I tried to keep most features in numerical format (converted categorical to meaningful numerical values)

1. Once the training data is generated, I ran co-relation analysis to see if any specific feature has significant impact on the wall time directly. As there are multiple features that act together to effect the execution time, I ran PCA (principle component analysis) and re-created a new set of features (from existing features - say like pca1=a.f1+b.b2+c.fc - where a, b, c are weights based on the information gain from features f1, f2 and f3)
2. Tried three different regression models to see which one is suitable for the target application and reporting the results on **Test Data.** [*This work doesn’t show the inference stage (where some of the profiling information will be unavailable)- this is future work*]

# Methodology:

The below Concept map shows the various sources of information that contribute to the features:

Graphical user interface, application

Description automatically generated

*Figure 1: The concept map that gives overview of tools and application used for this project and their respective inputs/outputs/logs.*

**The below shows the Various workflows I followed to gather the training Data and build the models:**

Graphical user interface, application

Description automatically generated

*Figure 2: The current Project Workflow*

Graphical user interface

Description automatically generated with medium confidence

*Figure 3: The Workflow tool wrappers - around the target application - runt time*

I used psutils[6] - in python to gather some of the system features

## Hardware Configurations:

| **Linux VM - 1 Node** | **OSC - 1 Node** |
| --- | --- |
| **Processor**: 2.3 GHz 8-Core Intel Core i9  **Memory**: 16 GB 2400 MHz DDR4  *The VM is created with combination of below:*   1. Cores-1, 2, 4 2. Memory - 2GB, 4GB | 1. **Owens (CPU):** Dell PowerEdge C6320 two-socket servers with Intel Xeon E5-2680 v4 (Broadwell, 14 cores, 2.40GHz) 28 processors, 128GB memory 2. **Pitzer (CPU):** Dual Intel Xeon 6148s Skylakes 40 cores per node @ 2.4GHz, 192GB memory 3. **Pitzer (CPU):** Dual Intel Xeon 8268s Cascade Lakes 48 cores per node @ 2.9GHz 192GB memory 4. **Pitzer (GPU):** Dual Intel Xeon 8268s - Dual NVIDIA Volta V100 w/32GB GPU memory 48 cores per node @ 2.9GHz, 384GB memory   Gathered this Info from OSC webpage |

## Sample Data - to generate Training Data & Tools

* 1. **Data Generation:**
* Grey Scott - No sample data needed, use smaller dimension matrix
* Trimmomatic - Taking the 1K, 10K, 1L and 1M raw reads (4 lines each) from a set of 10 original downloads from NCBI[4]. And executed a few full trims.
* DNNs - The original WIKI + IMDB data[5] - pre-cleaned->shuffled -> sample of 100, 200, 400 (to run on Stand alone Linux VM) and 1000, 2000, 5000 and 10000 (for OSC)
* ***But unable to run on Linux VM (as the models were to large for 4GB memory***

Had used the same data sets as test sets as well

* 1. **Tools**
     1. Tau[7]: to measure the io bandwidth and bytes written - profiling (can use communication measures as well - future)
     2. psutils[6]: Used this python too to get the system core information and other processor and memory information (application independent & request dependent)
     3. CODAR Cheetah [8]: the co-design framework used to run bulk jobs without having the user overhead of maintain several slurm scripts and work directories. Compatible with any Linux box. Can have post-application run hooks that help us gather the information as training data csv. (This really made generating training data a breeze)

1. Input, Application and System Features

The above configurations are used in the experiments I ran.

| **Feature**  **Type** | **Name and Description** | **Data Type** | **Application** | | |
| --- | --- | --- | --- | --- | --- |
| **1** | **2** | **3** |
| System | sys\_phy\_cores\_count, sys\_tot\_cores\_count - No. of cores per node  sys\_cpufreq\_mhz - CPU frequency  sys\_phy\_mem\_bytes, sys\_swap\_mem\_bytes - Physical and Swap Memory  max\_write\_bw\_mbps, max\_read\_bw\_mbps - IO MBPS  max\_bytes\_read\_mb, max\_bytes\_written\_mb - IO MB | Count, Bytes, MB, MBOS | O | O | O |
| is GPU? | T/F | X | X | O |
| sys\_ntasks\_per\_core, run\_nprocs - No of Processors/threads | Count | X | O | O |
| sys\_processor | 32/64 bit | FUTURE | | |
| Input | exe\_L, exe\_Du, exe\_Dv, exe\_F, exe\_k, exe\_dt - The diffusion formula input | Int and Float | O | X | X |
| file\_type - indicates the size of file (the no of raw reads in SRA, no if images in training data) | Size (count) | X | O | O |
| More features like - image dimensions, scarcity, other image meta data that effect execution time |  | X | X | F |
| More features like - length of raw sequence, the quality metrics of raw sequence - that effect execution time |  | X | F | X |
| Application | exe\_steps, exe\_noise |  |  |  |  |
| exe\_output, exe\_checkpoint, exe\_checkpoint\_freq, exe\_checkpoint\_output, exe\_adios\_config, exe\_adios\_span, exe\_adios\_memory\_selection, exe\_mesh\_type  : ***can try something else aprt from ADIOS for disk IO*** |  | F | X | X |
| model\_name - VGG16, InceptionV3, ResNet50  batch\_size- 4, 8, 16, 32, 64  epochs- 1, 5, 10, 100  learning\_rate- 0.001, 0.005, 0.0001 | Categorical,  Numerical |  |  |  |
| FP16, model\_name, MPI  :**Can try different DNN optimizations and include more Numerical/other features that describe the model (No. of parameters, Model states, functional/sequential model, FLOPS?)** |  | X | X | F |
| Predict Variables | exe\_time or fit\_time (Seconds) |  | O | O | O |
| epochs\_times\_one - First epoch time  epochs\_times\_avg - All epochs average |  | X | X | O |
| test\_accuracy |  | X | X | F |
| Trim\_Quality\_score |  | X | F | X |

The above table shows the overall features used in all three applications where:

| **Applications:**   1. Gray-Scott 2. Trimmomatic 3. DNNS | **Codes:**  X - Not Applicable  O - Applicable  F - A future possibility (Gathered currently, but did not vary thus is constant in training data) |
| --- | --- |

1. Co-relation Analysis and Principal Component Analysis
   1. Gray-Scott

| The following show the co-relation analysis done on pre-PCA training data: |
| --- |
| The above - co-relation map shows that “total\_exec\_time\_ms” (in secs) - depends on the matrix dimentions exe\_L and no of steps in the computations (which verifies the ttheritical intution)  No of processors - can go high if no. of cores are more.  Additionally, I removed the outliers. |
| *Scatter Plot: between the prominent Features and total\_exec\_time\_ms*    Just these features aren’t sufficint to build an intution about execution time, So we need to merge features - PCA |
| *PCA*    As we can see, the fiirst significat feature PCA 0 had more infromation from exe\_L, run\_procs. |
| *PCA Features vs execution time scatter plot:*    These scatter plot shows better relation than the co-relation with original non-transformed features. |

## Trimmomatic

| The following show the co-relation analysis done on pre-PCA training data: |
| --- |
| The above - co-relation map shows that “total\_exec\_time\_ms” (in secs) - depends on the no. of system cores (as it can span many threads).  Also more are the quality reioremens like (leading, trailing), it takes more execution time (negative co-relation.  The highest co-related feature is the size of the processing input (theoritically we knew this would be it!) (Scatter Plot below) |
| *Scatter Plot: between the prominent Features and total\_exec\_time\_ms*    The other features are not as prominnat. |
| *PCA*    As we can see, the fiirst significat feature PCA 0 had more infromationfrom rr\_count (file size in terms of count). - more threads, less time (negative co-relation). |
| *PCA Features vs execution time scatter plot:*    These scatter plot shows better increase in time with PCA0 but it could be better for clustering |

## DNNs

| The following show the co-relation analysis done on pre-PCA training data: |
| --- |
| * Since we ran it on only OSC (could not fit models on Linux local box), swap memory was kept constant. * More cores, less execution time, More size of input more training steps - so more execution time.   This is as expected. (Scatter Plot below) |
| *Scatter Plot: between the prominent Features and total\_exec\_time\_ms*    GPU - Yes has lesser execution time than GPU no. |
| *PCA*    The same as co-relation analysis |
| *The Co-relation between PCA Fetaures and Optput Values:* |
| *PCA Features vs execution time scatter plot:*  Calendar  Description automatically generated  Future: Can try PCA 0,2,3,4,5,6,7 and 8 - excluding 1 to see if accuracy improves. (MSE comes down) |
|  |

## Prediction Models - Regression

On PCA Features: (All predicting execution time in seconds)

| **Application/Models** | **Linear Regression X** | | **1 Layer Neural Network Regressor** | | **Decision Tree Regressor** | |
| --- | --- | --- | --- | --- | --- | --- |
| **MSE** | **MAE** | **MSE** | **MAE** | **MSE** | **MAE** |
| Gray-Scott | 969.96 | 19.07 | 76.993 | 4.16 | 428.28 | 5.5 |
| Trimmomatic | 3.95 | 1.34 | 5.26 | 1.30 | 6.94 | 1.37 |
| DNNs - predicting per-epoch time | 49435.17 | 131.31 | 3861.53 | 20.86 | 14863.16 | 34.69 |
| DNNs- predicting overall training time | 140830.95 | 208.46 | 13806.21 | 37.84 | 43934.90 | 69.03 |

Where MSE is Mean Square Error and MAE is Mean Absolute Error.

***Note: I created models earlier before including the profiling features, and the scores weren’t good. They are better after including the profile data. (will share them as supplemental material)***

*Observations:*

* DNNs :
  + We need more data related and model related features to reduce the error in DNN models (MSE).
  + Predicting the per epoch time has lower error than overall training time. (more is the value to predict - more epochs the error is being increased.
* The results with Simple linear regression generate negative execution time - Hence ***this cannot*** be a model for us to use.
* Decision trees fit based on the type of features we picked -no. cores, no. threads (linear features) GPU/CPU, PE/SE (true or falue features) and also has few other categorical features (model names)
* Since we converted the categorical values into meaningful numerical values (like trimmomatic input to the no. of reads, DNN inputs to the no. of images in training data) we were able to run a simple 1 layer DNN to predict execution time - and this has minimum MAE.

# Future Work

There are still many things to extend to make this prediction work:

1. Training Data: The current training data has full size (real run samples) included in the training phase - but their counts are low, so they do not effect the model much. Must change the models to transfer learning or ensemble models so that they have more weight.
2. Should try multi-node DNNs (after setting up Cheetah on OSC) and use tau to measure OpenMPI communication profiling.
3. Should Include more features related to DNN models as well (instead of just mapping model to number - this worked because VGG16> ResNet50>InceptionV3 execution time so labeling them 1, 2, 3 should still make the co-relation viable - but not best). Numerically resenting the model as features can generalize the use of the dataset to more Image classification models. (Data set generalization)
4. Should include domain specific features about the quality of input data (for trimmomcatic) that characterized the execution time.
5. Should make the models for inference stage - at test phase we have profile information generated on fly along with application. But for inference, we will not have these values. So, I could either try leave-one-out-loop regression and predict the values in dependency order. Or, use the generic missing data filling techniques (mean, distribution by variance, 0).
6. Should use synthetic data to have better coverage of features and have more control on training data.

I will share a copy of the document in Drive [https://drive.google.com/drive/folders/1i64YRe\_Qc-ziXAz0VU3-ji3F1lrw2KMX?usp=sharing] and keep updating this future work over the coming weeks.

# References:

[1] <https://groups.csail.mit.edu/mac/projects/amorphous/GrayScott/>

[2] "Complex Patterns in a Simple System," by John E. Pearson and "Pattern Formation by Interacting Chemical Fronts," by K.J. Lee, W.D. McCormick, Qi Ouyang, and H.L. Swinney. These articles appeared in *Science*, Volume 261, 9 July 1993

[3] <https://www.illumina.com/techniques/sequencing/dna-sequencing/whole-genome-sequencing.html?scid=2021-269PPC3922&catt=platforms_ppc>

[4] <https://www.ncbi.nlm.nih.gov/sra>

[5] <https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/>

[6] <https://psutil.readthedocs.io/en/latest/>

[7] <https://www.cs.uoregon.edu/research/tau/home.php>

[8] <https://github.com/CODARcode/cheetah>