Potential Analisys of the Data Processing on CSDs Units

L. M. VIANA, PESC – COPPE, UFRJ, RJ, 2021

***Abstract*—The main goal of this work is verify the performance of the new *computation storage devices, CSD* drives, which have embedded processing support. This drives has the capability of optimize imaging processing applications with focus on the GIS segment, and offer support to store efficiently the data received of geographic monitoring activities using satellite, airplane or drones collecting images, and compare the performance of this device with the conventional solution which use *solid state disks* without computation support, the conventional *SSD* drives, on search and the processing of an large volume of data. Our experiment show... <#1-Will be included here our best results and our conclusions>.**

***Index Terms*—Computation storage, Imaging processing, Learning systems, Supervised learning, Machine learning.**

I. INTRODUCTION

The main objective of this study is develop an optimized storage mechanism to support large image and video data, obtained from different sources, like satellite, airplane, drones and vigilance cameras. In this study we had available the performance of the CSDs storage drives in the execution of distributed write and read access, data classification, search and imaging processing of the data, with the goal to get a performance improvement of those devices in situations where we have a large volume of information composed by multiple files with big dimensions, this is bigger than 400 MB each, which need to be frequently processed to make format conversions, coordinate systems re-projections, image cache creation with different resolutions, image comparing to identify changes, pattern recognition, and others.

II. INITIAL CONCEPTS

The concepts of computation storage has been largely studied. Applying the processing resources directly in the storage units permit which any services supported by the operating systems or by the application layer been ported to the storage units, reducing the CPU work and getting better performance in the data transfer between the storage units and the memory.

Received in January 28th, 2021. PESC – COPPE, URFJ, RJ. Doctor Application Study. Coordinators: Felipe França, Diego Dutra, Tiago Alves. L. M. Viana, student at PESC – COPPE, UFRJ, RJ, (e-mail: lmarcio@cos.ufrj.br).

Actually the CSDs units with processing capacity are been largely used in applications which needs storage data in the compacted format and with security, providing efficient, security and space economy in the storage.

Others applications of CSDs units are related with the optimization of search activities done by database management systems, and by artificial intelligence applications.

The study involve the use of the CSDs units in the optimized storage of images and videos, for data processing and classification, offering an efficient database which have the capability to execute read and write, classifications, searches and imaging processing efficiently.

Figure 1: Server with 24 CSDs storage units.

In this study will be used the Newport CSDs storage units, with capability of data processing, with the objective to optimize the processing of large images and the search by information (Figure 1).

Hardware Architecture: Programmable SSD storage unit, developed to outperform the traditional SSD storage units, which was built around the personalized ASIC unit, Zynq Ultra Scale + FPGA of the Chinese manufacture Xlinx, use the communication protocol MVMe, with 4 lines of PCIe Gen 3 interface with bandwidth of 4 GB/s, and support to make searches using regular expressions and data cryptography to provide read and write access in real time, and to encode and decode the data in the storage unit I/O process (Figure 2).

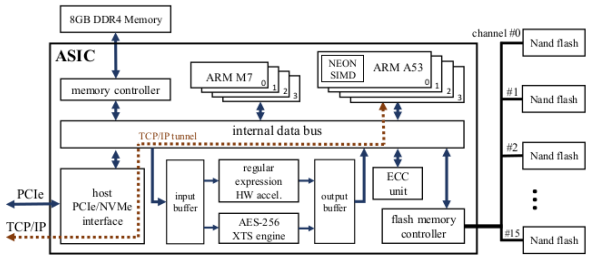

Figure 2: Hardware architecture of Newport CSD unit.

Software Layer – Image Server Kernel: Linux, NVMe communication protocol for distributed process intercommunication with each Newport CSD and the host computer, with the objective of bypass the performance problems identified with the TCP/IP Tunneling over NVMe/PCIe, which will reduce significantly the data transfer in the NVMe/PCIe interface (Figure 3).

Experiment Enviroment: Linux with GCC (C/C++), GDAL, Proj6 with SQLite, MapGuide Open Source, FDO, JBOSS / Wildfly with Java, RESTFUL WebService with XML and JSON; API with support for Java and Python, and Sucuri framework for process distribution.

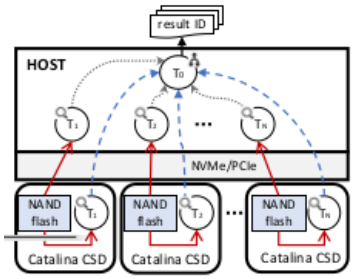

Figure 3: Software layer of image server kernel.

A. Problems with Image Data Collect

The use off many sensors collecting data and images sent over the existent telecommunication network to a data center, generate a bigger data traffic which turn the task of receive and process images impossible to scale efficiently.

With the grow of the capacity in the processing of the monitoring units, we can now realize previous data and image classification, to permit the remote unit to choice what will be sent to the data center. The identification of data and images patterns in the remote units could reduce the telecommunication traffic in 20 times the size of the data sent.

Hardware Architecture of the Remote Units: (1) Raspberry Pi 4 - Processor Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64bit SoC – Clock 1.5 Ghz – Memory RAM: 4GB DDR4; (2) Jetson Nano - NVIDIA Maxwell with 128 NVIDIA CUDA cores - Processor Quad-core ARM Cortex-A57 MPCore - 4 GB 64 -bit LPDDR4 - 16 GB eMMC 5.1 Flash (Figure 4).

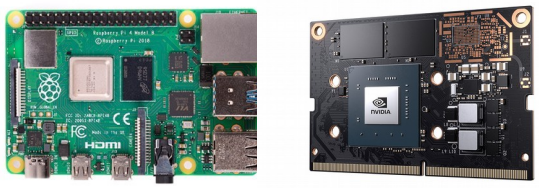

Figure 4: Hardware architecture of Raspberry Pi 4 and Jetson Nano

B. Sensor Network

With the implementation of a sensor network with many sensors distributed over a geographic area, like a city (Figure 5). We need reduce the volume of the data traffic over the telecommunication network, and the remote units have to use neural network models to classify the collected data and images to check if that information meet an specific criteria before sent that information to the data center.

Those data are concentrated in many servers with are responsible by receive the data and store them in a distributed format on the many CSDs units.

Each CSD unit complete a data classification follow the defined criteria by the neural network models loaded in each CSD unit and with the classification of the data, the CSD unit update the image database.

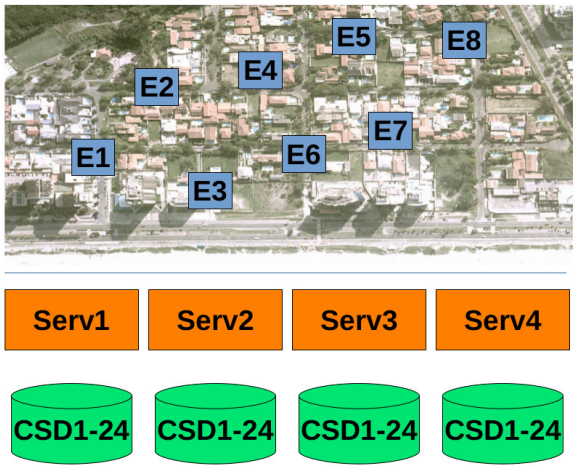

Figure 5: Software layer and the experiment enviroment.

The evolution in complexity of the deep neural networks models (Figure 6), increase the time needed to training the models, and grow the execution time and many others resources consuming. Because that it could be generated using some servers in the data center, the servers have high processing capability and can distribute the task to the CSDs units.

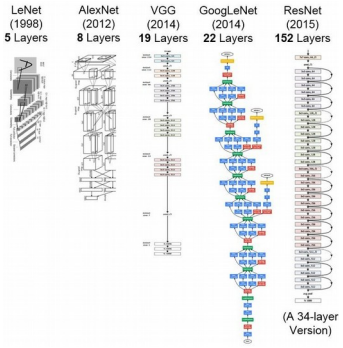

Figure 6: The evolution complexity of deep neural network models.

After the generation of the deep neural network model, they could be loaded in the remote units to provide the data classification remotely to reduce the data sent to the servers, but maintained the hard imaging processing tasks been executed in the servers with a distributed storage over the CSDs units.

III. EXPERIMENT ENVIRONMENT

To realize the experiments was implemented an Image Server, the Horus Image Server v1.0, which has the task to server like an image database with the capability of classify the features identified in each satellite image or any image generated by vigilance cameras.

The Image Server could be installed on an unique server or on multiples servers, working on a cluster, with each server working with multiples CSD storage devices with processing capability to offer better performance on classification and search of the information.

The Image Server accept the loading of multiples models which will be used in the analysis of the images. Each model has an autonomous process for data training and classification.

Data classification will occur in background on each CSD processing unit, and will update the correspondent tables and index. To training the model is permitted to load the images with a background mask to identify each feature. Each neural network model will be loaded as a server module.

To construct the modules for the neural network model is necessary to implement the interface IHorusModuleInterface (HMI) with is presented as the Listing 1, for this task, is used the GNU C/C++ with the Machine Learning Pack for C/C++ (MLPACK), or the Wizard and ClusWizard C/C++ libraries.

```
class IHorusModuleInterface {
      void hmiLoadModule(char* moduleName);
      void hmiInitModule(int argc, char** argv);
      void hmiStartModule();
      void hmiTrainModel(byte* data, double val);
      double hmiClassifyModel(byte* data);
      void hmiSaveModel(
             char* modelName,
             char* modelFileName);
      void hmiLoadModel(char* modelFileName);
      void hmiTerminateModule();
};
```

Listing 1: Definition of the Image Server module interface.

The modules are loaded in each processing node with the correct compilation: (a) In the host computer, using Linux for Intel x64 or (b) in the Newport CSD unit for ARM Cortex A57.

For classification each module will launch one or more independent processing threads which will execute the image classification with the update of correspondent tables.

The server has an API interface for Python with the Sucuri framework for distributed execution of the experiments tasks.

A. Architecture Data Flow

The Figure 7 shows the application data flow on the execution environment. The server core is located at the host machine which has three components: (1) Kernel; (2) Command Interpreter; and (3) Listener.

The client interface was built using the HorusRemote service which has support for Java and Python languages, and is used to do the requests to the server.

Over this API was built a Restful Web Service using JAVA with Wildfly, and other using Python with Restful support. The Restful Web Service was created to provide an easy access over the Web.

In the host machine we have the Listener which is responsible by receive the requests from the clients using the HorusRemote service. After receive a request the Listener send it to the Command Interpreter which parse the command and start a new thread to execute it.

The Command Interpret dispatch the process in the Kernel to search the location of the data on it's tables, and redirect the process to the correspondent CSD unit. To do the redirection the Kernel access the follow base tables: (1) users table for access control; (2) table space table to identify the location of the data on the CSD unit; and (3) the table meta data which has information about the table data file.

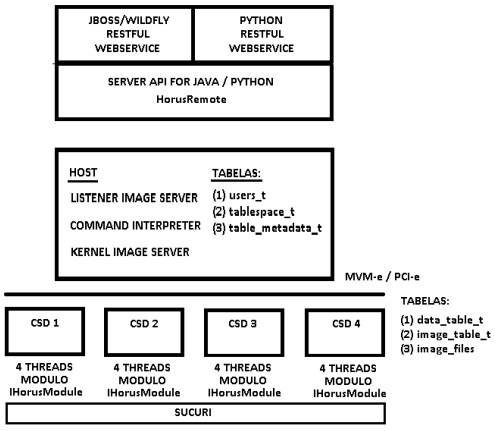

Figure 7: Architecture Data Flow.

In each CSD unit is running up to 4 threads which are responsible by search the data on the data and image tables, and dispatch the module responsible by fetch the data and process it before return the data to the host machine.

Each process module implement the interface IHorusModule show at Listing 1. It's used for training, classification, or process the image.

B. Sucuri Support

The Sucuri Framework was created with the objective to provide a distributed Framework based on the program data flow.

Sucuri uses share memory for local process communication and MPI support for communications between process executing on different hosts.

One of the main features of the data flow model is solve synchronization, concurrency and communications problems existent on any distributed Framework on a efficiently way.

The Figure 8a and Figure 8b show a sample of program code and the correspondent program data flow which show the program dependencies.

```
01 | a = f()
02 | b = g()
03 | c = h(a,b)
04 | d = h(a,c)
05 | e = w(c,b)
06 | x = t(d,e)
```

Figure 8a: Sample of program code sequence.

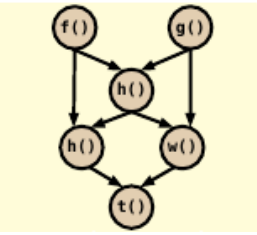

Figure 8b: Sample of program data Flow.

The Sucuri code is created following the steps:

Line 1: Import Sucuri Library

Line2: Is created an empty Sucuri graph

Line 3: Is created the Sucuri scheduler, passing the graph and the maximum number of worker threads.

Line4 to 9: The data flow nodes are inserted at the graph.

Line 10 to 23: The data flow edges are inserted at the graph.

Line 24: The Sucuri scheduler is started.

This same code could be executed on a standalone way or on a cluster of computers. To execute the Framework on a cluster is necessary enable the MPI support at the scheduler.

The Sucuri Framework is applied in 2 different layers of the Image Server.

The first layer is applied to provide a distributed request of data to the Image Server, with the objective to create a concurrent experiment environment. The program at this layer is implemented using the Image Server API for Python with a instance of HorusRemote object.

The other layer is applied with the objective of explore the In-Situ computing support of the Sucuri Framework, and to integrate with the Image Server, the Sucuri process implement the interface IHorusModule. The use of Sucuri Framework permit a more easily application development on the distributed environment of the CSD units (Figure 9).

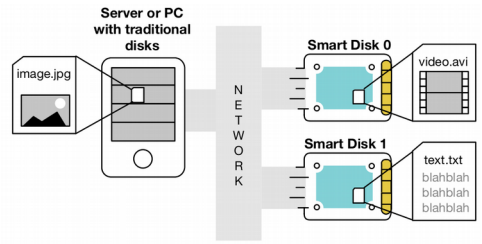

Figure 9: Sucuri In-Situ support for CSD communication.

C. Experiments Data Base

The first experiments was realized using 23 GB true-color images with resolution of 25 meters which was provided by the IBGE - Brazilian Geographic and Statistics Institute.

The median size of each compressed image is 50 MB, and that size grow to 600 MB when the image is decompressed for processing.

The data base was created with the identification of map areas without vegetation, and the data was selected from the base using a query which select 190 images, 4,7% of the total data base.

In the experiments the storage units was responsible to run the query and select the data in the local data tables, and then execute three different image processing before return the results to the host.

The first processing was convert the original image from GeoTIFF to JPEG, following by the re-projection of the image from UTM to Geographic Coordinate System, and finally, simplify the image size to ¼ of the original size to return the data to the host.

IV. Experiments Results and Analysis

To compare the results with the results obtained in the article “POLARDB Meets Computational Storage: Efficiently Support Analytic Workloads in Cloud-Native Relational Database”, written by C. Wei, L. Yang, C. Zhushi, Z. Ning, L. Wei, W. Wenjie, O. Linqiang,W. Peng, W. Yijing, K. Ray, L. Zhenjun, Z. Feng, Z. Tong, and presented at 18th USENIX Conference on File and Storage Technologies, which occur in 2020, related at reference [2]. We conduct the experiments as following.

A. Target Experiment

Our target experiment is execute up to 6 different queries which will be executed 5 times each one, independently after the initialization of the Image Server, to guaranty which no cache will be applied to any queries.

The queries recover respectably the following volume of data from the server: TS-1: 1.25%; TS-2: 15.17%; TS-3: 30.34%; TS-4: 54.04%; TS-5: 63.22%; TS-6: 100.00%.

The experiments will be executed using two different computer configurations: (a) with one Newport CSD device with 4 simultaneous threads in execution; and (b) with two Newport CSD device with 4 simultaneous threads in execution.

The base system used in our analysis will run with same number of threads on multicore server, using conventional SSD device.

B. Experiment Realized

Our initial experiment was executed using just one query, TS-1 with 4,7% of total data, 190 images. The experiment process the selected images, converting from GeoTIFF to JPEG, re-projecting them from UTM to Geographic Coordinate Systems, and simplifying them to ¼ of the original size, before return the results to the host.

The experiments was executed using one basic computer with 2 cores, simulating the following storage units configuration: (a) host only processing (SEQ), our analysis base system, (b) the host with 2 working process simulating 2 storage units (U2), and (c) the host with 4 working process simulating 4 storage units (U4).

In this first experiment we didn’t take in consideration the time to transmit the data from the working process to the host, but on the target experiment with have intention in calculate the gain with the reduction of the data transmitted.

The Tables 1a and 1b, show the processing time and the speedup obtained with the partition of the data processing over more than one working process executing on the same host.

	TOTAL PROCESSING TIME (s)		
	SEQ	U2	U4
TS1-C	4057.2	2616.2	2403.8
TS1-R	3062.1	3335.7	2736.1
TS1-S	2062.8	1637.7	1396.4

Table 1a: Experiment processing time on the same computer.

	PROCESSING SPEEDUP – REFERENCE SEQ		
	SEQ	U2	U4
TS1-C	1.00	1.55	1.69
TS1-R	1.00	0.92	1.12
TS1-S	1.00	1.26	1.48

Table 1b: Experiment Speedud on the same computer.

At the Figures 10a and 10b, we can observe a speedup of 1.55 (U2) and 1.69 (U4) for image conversion, and 1.26 (U2) and 1.48 (U4) for image simplification, when compare with the base system (SEQ). The result for re-projection have a speedup of 1.12 (U4) over the base system (SEQ).

This significant reduction on the total processing time when the number of working process grow was obtained using a basic computer with 2 cores and without any SSD device installed.

The results will be significantly better if we use independent CSD units, which capability for in-situ processing and analysis of the images before return the data to the host, re-ducting the data size and improve the data transmission to the host.

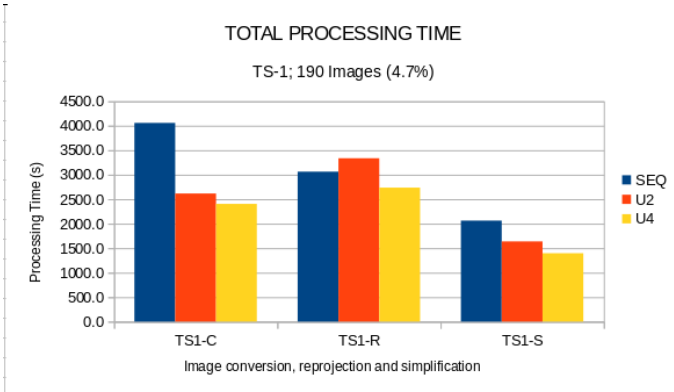

Table 10a: Experiment processing time on the same computer.

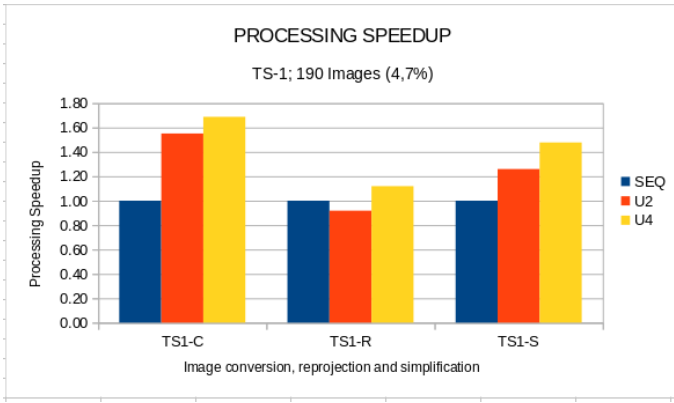

Table 10b: Experiment Speedud on the same computer.

V. Conclusions

Our first experiment was conducted using a basic computer with 2 cores and without any CSD device installed, and we can observe a speedup of 1.69 (U4) for image conversion, 1.12 (U4) for image re-projection, and 1.48 (U4) for image simplification when compare with the base system (SEQ).

For our experiment we use two and four working units, simulating CSDs devices on the same computer, and with this configuration we could observe the potential gain which we can obtain with the partition of the processing over different working process.

This gain would be improved using computation storage units, CSDs devices, to process the data and return the processed information to the host, re-ducting the overall size of the data transmission.

In our future work we will use similar configuration of that which was presented at the article “POLARDB Meets Computational Storage: Efficiently Support Analytic Workloads in Cloud-Native Relational Database” [2], to compare the gain obtained with data base and image processing on computation store.

References

[1] Anonymous; “I/O and Energy Efficient Large-scale Image Similarity Search Using Computational Storage Devices”, USENIX FAST;

[1] D. Jaeyong, C. F. Victor, B. Hossein, T. Mahdi, R. Siavash, H. Ali, Heydarigorji, S. Diego, F. G. Bruno, S. Leandro, K. S. Min, M. V. L. Priscila, M. G. F. França, A. Vladimir; “Cost-effective, Energy-efficient, and Scalable Storage Computing for Large-scale AI Applications”; ACM Transactions on Storage, Vol. 16, No. 4, Article 21. Publication date: October 2020.

[2] C. Wei, L. Yang, C. Zhushi, Z. Ning, L. Wei, W. Wenjie, O. Linqiang,W. Peng, W. Yijing, K. Ray, L. Zhenjun, Z. Feng, Z. Tong; “POLARDB Meets Computational Storage: Efficiently Support Analytic Workloads in Cloud-Native Relational Database”; 18th USENIX Conference on File and Storage Technologies; pages 29-41; 2020.

[3] T. Surat, M. Bradley, H. T. Kung; “Distributed Deep Neural Networks over the Cloud, the Edge and End Devices”; Naval Postgraduate School Agreements No. N00244-15-0050 and No. N00244-16-1-0018. 2015.

[4] Z. Li, W. Hao, T. Radu, and H. C. David Du; “Distributing Deep Neural Networks with Containerized Partitions at the Edge”; 2019.

[5] C. Sandeep, C. Eyal, P. Evgenya, C. Thianshu, K. Sachin; “Neural Networks Meet Physical Networks: Distributed Inference Between Edge Devices and the Cloud”; 2018.

[6] H. Ali, T. Mahdi, R. Siavash, B. Hossein; “STANNIS: Low-Power Acceleration of Deep Neural Network Training Using Computational Storage Devices”; 2020.

[7] L. Chun-Yi, B. K. Jagadish, J. Myoungsoo, T. K. Mahmut; “PEN: Design and Evaluation of Partial-Erase for 3D NAND-BasedHigh Density SSDs”; ISCA 2020.

[8] T. Arash, G. L. Juan, S. Mohammad, G. Saugata, M. Onur; “MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices”; ISCA 2020.

[9] A. Bulent, B. Bart, R. John, K. Mathias, M. Ashutosh, B. A. Craig, S. Bedri, B. Alper, J. Christian, J. S. Willian, M. Haren, and W. Charlie; “Data Compression Accelerator on IBM POWER9 and z15 Processors”; ISCA 2020.

[10] B. Eunjin, K. Dongup, and K. Jangwoo; “A Multi-Neural Network Acceleration Architecture”; ISCA 2020.

[11] B. Summet, G. Jayesh, S. Zeev, R. Lihu, Y. Adi, S. Sreenivas; “Focused Value Prediction”; ISCA 2020.

[12] K. Jeremy, A. Willian, B. Willian, B. Nadya, B. Robert, B. Chansup, C. Gary, G. Kenneth, H. Matthew, K. Jonathan, M. Andrew, M. Peter, P. Andrew, R. Albert, R. Antonio, Y. Charles; “Dynamic Distributed Dimensional Data Model (D4M) Database and Computation System”; 2012.

[13] C. M. Rodrigo, O. T. Giovane, L. P. Maurício, L. P. Laércio, T. C. Amarildo, M. G. F. Felipe; “Value Reuse Potential in ARM Architectures”; IEEE 28th International Symposium on Computer Architecture and High Performance Computing; 2016.

[14] L. P. Maurício, R. C. Bruce, T. C. Amarildo, M. G. F. Felipe, O. A. N. Philippe, “A Speculative Trace Reuse Architecture with Reduced Hardware Requirements”, 2006.

[15] D. G. Massimo, G. Maurizio; “WiSARD rp for Change Detection in Video Sequences”; ESANN 2017.

[16] D. G. Massimo, G. Maurízio; “Change Detection with Weightless Neural Networks”; CVPR 2014.

[17] C. Chunlei, C. Li, Z. Lei, Z. Xiaoyun, G. Zhiyong; “RCDFNN: Robust Change Detection Based on Convolutional Fusion Neural Network”; 2018.

[18] Y. S. Abu-Mostafa, M. Magdon-Ismail, H. Lin Authors, “e-Chapter 8 – Support Vector Machines” in Learning From Data a Short Course, Country: Passadena, CA, USA, 2012, ch. 8.

[19] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011, https://scikit-learn.org/stable/user_guide.html.

[20] C. G. B. Caio; “An Adaptation of the Data Flow Library Sucuri Static Scheduler for In-Situ Computing”; Dissertation presented to COPPE/UFRJ in March, 2018.