

# Cloudifier Virtual Apps

## Virtual desktop predictive analytics apps environment based on GPU computing framework

Andrei Ionut DAMIAN (*Author*)

Cloudifier SRL  
Bucharest, Romania  
damian@cloudifier.net

Alexandru PURDILA (*Author*)

Cloudifier SRL  
Bucharest, Romania  
alex@cloudifier.net

Nicolae TAPUS (*Author*)

University “Politehnica” Bucharest  
Bucharest, Romania  
ntapus@cs.pub.ro

**Abstract** — The need for systems capable of conducting inferential analysis and predictive analytics is ubiquitous in a global information society. With the recent advances in the areas of predictive machine learning models and massive parallel computing a new set of resources is now potentially available for the computer science community in order to research and develop new truly intelligent and innovative applications. In the current paper we present the principles, architecture and current experimentation results for an online platform capable of both hosting and generating *intelligent* applications – applications with predictive analytics capabilities.

**Keywords**— *artificial intelligence, machine learning, virtual desktop, predictive analytics*

### I. INTRODUCTION

The entire field of Artificial Intelligence tends to become more and more about improving predictions and predictive models as Goldfarb et al [1] argue in the paper “Managing the Machines”. Predictive modelling can now be used for extremely varied tasks ranging from constructing expert system chat-bots [2] based on Long Short Term Memory deep recurrent neural networks that would use natural language processing capability (reading and writing) up to melanoma recognition in dermoscopy images [3] based on deep convolutional networks.

In our paper, we will propose several approaches:

(i) a web-services based model for the deployment of model training and prediction on GPU massive parallel computing backend. The GPU based parallel computing environment will enable to construct and optimize a GPU-optimized set of machine learning algorithm implementations.

(ii) employment of automatic software design, development and deployment for user experience modules based on a pipeline shallow machine learning model [4]

(iii) a unified architecture and underlying framework for generation and hosting predictive analytics applications.

Within our paper we will argue that our proposed research and experimentation will fuel the development and implementation of a production-grade system capable of addressing in the same time the needs of several categories of customers such as:

- users in need of a online application marketplace fully oriented on predictive analytics applications;
- software developers in need of a legacy apps automatic software translation tools including machine learning templates;
- scientific users both from academic and commercial environment looking for an online application framework and execution environment in order to conduct predictive experiments in areas such as healthcare, finance and environment;

### II. RELATED WORK

Our work relates to the most recent advances both in the field of High Performance Computing and Machine Learning, particularly in the areas of machine learning model deployment on massive parallel processing infrastructures based on numerical computing cores.

#### A. GPU based High Performance Computing

The usage of GPU based computing is currently mostly applied in deep learning – both research experiments and production development. However, little to no attention is given to the employment of GPU parallel computing resources in shallow machine learning techniques such as decision trees, multi-variate linear and logistic regression, Naive-Bayes models. Nonetheless the current advances and the state-of-the-art prediction scores achieved by the deep learning scientific experiments give little room to further scientific development and advances of known shallow models techniques. Even more, companies such as Microsoft, Google or IBM currently propose online framework-engines that allow on-demand web-based prediction model creation and experimentation [5] that are

entirely used for off-line training and prediction experiments, running on CPU based infrastructures. Taking these presented aspects into account we are proposing a set of new modern approaches that will potentially empower shallow machine learning models with the capability using GPU based parallel computing resources. Our approach fully relates with the current state-of-the-art in the area of GPU-based tensor computation frameworks currently employed in Deep Learning frameworks [6] [7].

Another of the supporting aspects that we will use in our approach is based on the most important features of shallow machine learning models: the ability of the shallow-model to self-explain itself. This ability of shallow machine learning models such as linear models or decision trees models is certainly available at the expense of prediction power shown by deep models (deep learning models with massive hidden weight spaces).

### B. Architectural roots

The proposed architecture is based on our previous work conducted in the area of automatic software application migration and machine-learning assisted automatic programming [4]. Based on our automatic software migration architecture [4] combined with the latest advances in large-scale machine learning on heterogeneous distributed systems [6] we will be able to achieve the following desiderates during the production phase of our project:

- Create a scalable architecture for a user-friendly online environment where external users will be able to quickly generate and customize application starting from existing legacy apps;
- Combine the power of the automatic application generation features with template-based models in order to create a business horizontal application repository that will include apps such as (a) multi-vertical customer churn prediction models; (b) multi-vertical customer behavior inference model; (c) business flow generated data exploration stub apps; (d) machine learning stub/template applications
- Finally, we are able to create a research applications repository that will continuously grow by accommodating machine learning augmented apps in areas such as medical diagnostics inference/prediction, financial predictive analytics, environment inferential models. The proposed scientific apps will be developed within the “Cloudifier” community and will fully utilize the scalable GPU-augmented machine learning template models;

## III. PROPOSED ARCHITECTURE

The main points of our proposed architecture are the following:

- A. General execution framework based on REST services with HPC backend powered by GPU computing;
- B. Machine learning augmented applications generated within a provisioned Cloud infrastructure by automated software migration [4];
- C. Virtual desktop predictive analytics apps repository consisting of template-developed business applications and scientific experiments;

### A. REST based massive parallel computation services

Based on the fact that one of our paper areas of innovation is the augmentation of shallow machine learning models with parallel processing techniques based on GPU processing, we are proposing a REST based service for parallel computation on GPU backend. At present time, based on our knowledge, the advancement of GPU based hardware infrastructures capable of delivering massive parallel computing for machine learning tasks at very low costs has only been exploited by the area of deep learning and deep neural networks in particular. Our proposal addresses this gap between the available computation resources for deep learning and the ones available for scientific shallow machine learning.

In past years various papers have been arguing that shallow models can achieve similar results with deep models if sufficient data is provided, such as the famous quote defending the power of data by Google’s Research Director Peter Norvig: “We don’t have better algorithms. We just have more data.” [8]. In our proposed approach for advancing shallow machine learning by GPU parallel computing augmentation we will argue that shallow model’s pipelines can achieve comparable results with deep learning models for various applications. Although most of the state-of-the-art research in the area of machine learning is focused on deep learning models, the classic machine learning models with shallow hidden layer structure have two main characteristics that deep models lack: self-explain ability and algorithmic robustness for online and real-time environments.

Currently several programming approach options are available for programming GPU parallel computation engines both for scientific purposes and for engineering/production and probably the most widely used are OpenCL [9] and CUDA [10]. Basically, both mentioned models use the same technique with only slight differences. The two most fundamental aspect in regard to the computational approach consists in the two central entities consisting in *kernel* – that is the function containing the actual code to be run in parallel - and the *devices* – the actual computational devices that will run the *kernel* in parallel. Traditionally the *kernel* is a function described by the the following generic algorithm:

---

#### Algorithm 1 *Kernel(Data, BlockInformation)*

---

cID  $\leftarrow$  get coordinates tuple from *BlockInformation*  
 Modify *Data*[cID] with hard coded <operation>

**Return**

---

Note that in above example *BlockInformation* defines the actual data segment that a particular *device* is processing within *Data* and  $\langle operation \rangle$  denotes the hard-coded (compiled) operation that the *kernel* is computing for the particular block of *Data* it receives. For the particular case of matrix dot product we can have each *device* compute with its *kernel* the dot product of two vectors (the row vector of 1<sup>st</sup> matrix and the column vector of 2<sup>nd</sup> matrix).

$$A * B = A^T B = \sum a_i b_i \quad (1)$$

Finally, we have the following two code snippets for computing (1) in OpenCL and CUDA based *kernels*:

---

**Code 1 OpenCL Kernel for Matrix-Matrix dot product**

---

```
__kernel void OpenCLMatrixDotKernel(const int
A_ROWS,
const int BC_COLUMNS,
const int A_COLUMNS,
const __global float* A,
const __global float* B,
__global float* C)
{
    const int Row = get_global_id(0);
    const int Col = get_global_id(1);

    float acc = 0.0f;
    for (int c=0; c<A_COLUMNS; c++) {
        acc += A[c*A_ROWS + Row] *
B[Col*A_COLUMNS + c];
    }
    C[Col*A_ROWS + Row] = acc;
}
```

---



---

**Code 2 CUDA Kernel for Matrix-Matrix dot product**

---

```
__global__ void CUDAMatrixDotKernel(Matrix A,
Matrix B, Matrix C)
{
    float acc = 0;
    int row = blockIdx.y * blockDim.y +
threadIdx.y;
    int col = blockIdx.x * blockDim.x +
threadIdx.x;
    if(row > A.height || col > B.width)
        return;
    for (int e = 0; e < A.width; ++e)
        acc += A.elements[row * A.width +
e] *
B.elements[e * B.width +
col];

    C.elements[row * C.width + col] = acc;
}
```

---

The above *OpenCL* and *CUDA* kernels are basically the work-horses of the most basic and well-known machine learning algorithms such as liner and logistic regression, SoftMax regression, matrix-factorization based recommender systems and others. The main purpose of above examples is to present the low-level approach for computing usual predictive analytics tasks such as the inference within a LogReg model given by equation (2) where  $\hat{y}$  represent the true-label

probability for a binary classification task,  $\sigma$  represents the sigmoid function,  $X$  represents the multi-variate observations dataset and finally  $\theta$  represents the learned coefficients vector of the models predictor variables.

$$\hat{y} = \sigma(X \cdot \theta) \quad (2)$$

For large scale datasets such as those met in BigData analytics such operations are extremely time and resource consuming. Nevertheless, the employment of the GPU kernel based computation parallelism will enable this operation to be executed in a time dramatically lower than with classical CPU approach.

The implementation of the *kernel* code will be left for a lower level tensor computation framework, namely TensorFlow [6], a Open Source framework already established as a industry state-of-the-art standard. TensorFlow is a programming framework for expressing machine learning algorithms combined with a backend implementation for executing such algorithms in various heterogenous parallel computing environments. One of the most important features of TensorFlow is the automatic deployment of tensor graphs calculation on multiple GPUs, although without high-level resource allocation mechanics.

As previously mentioned, in the current paper we will present a REST [11] approach for cloud computing based provisioning of parallel computation services. This approach will enable the implementation and consumption of a web based service that will have the following generic algorithm based on FIFO allocation.

---

**Algorithm 2 ParallelProcessingREST(Request)**

---

For each Kernel, Data pair in Request

*Kernel*  $\Leftarrow$  get **high-end kernel** source code from *Request* (TensorFlow [6] code)

*Data*  $\Leftarrow$  get dataset from *Request* by value or by reference from external source

*Available*  $\Leftarrow$  get available devices from existing cluster based on adaptive **Algorithm 3**

If *Available* is valid then

Off-load *Kernel* and *Data* on

*Available* devices

Get off-load *Result* and append to

*Response*

Else

*Response*  $\Leftarrow$  Overhead time

exceeds computing time

**Return Response**

---

The actual GPU execution strategy of the off-loaded kernels will be based on an adaptive algorithm that will take into consideration server-load, task complexity and GPU load/preparation overhead.

---

**Algorithm 3 ResourceAllocation(Kernel, Data)**

---

*cFLOPS*  $\Leftarrow$  evaluate needed FLOPS for *high-level Kernel* and *Data*

```

Resources  $\Leftarrow$  Compute needed devices based on
cFLOPS
ComputeTime  $\Leftarrow$  Aproximate total execution time
based on Resources, cFLOPS
WaitTime  $\Leftarrow$  Off-load overhead time + time until
devices are available
If ComputeTime > WaitTime
    Result  $\Leftarrow$  Queue position or actual allocated
    devices
else
    Result  $\Leftarrow$  allocation invalidated

```

**Return** Result

Finally, the entire predictive analytics REST webservice and app repository is based on a simple algorithm (Algorithm 4) for servicing the following requests:

- Training new models based on post data or externally available data
- Prediction/inference based on trained models (via a model handle obtained after training)

---

**Algorithm 4** GetRequest(Handle, Data, Model)

---

```

If Handle exists then
    If Data denotes model status request:
        Result  $\Leftarrow$  Model status (Working, Trained,
        InferenceReady)
    Else
        Prepare Request  $\Leftarrow$  Handle, Data, Model
        Call ParallelProcessingREST (Request)
        Update Handle
        Result  $\Leftarrow$  Handle
Else
    If
        Select high-level Kernel model  $\Leftarrow$  Model
        Call ParallelProcessingREST
        Generate Handle
        Result  $\Leftarrow$  Handle

```

**Return** Result

---

The actual messages that are passed between clients and the REST server are simple JSONs in the following format:

```

{
    HANDLE: ModelHandle, # not NULL if requesting
    results

    DATA: DataDict, # data dictionary containing
    predictor variables, targets (if available) with
    actual data or link to external data repository

    MODEL: ModelName # a model (specific or
    generic) to be trained (if model HANDLE is not
    yet available)
}

```

For a clearer understanding of the mechanics behind the proposed architecture we will present the full algorithm cycle for the case of requesting the training of a recommender system (Use case algorithm 1).

---

**Use case algorithm 1** Request(TransactionalDataDict, "Recommender")

---

```

Model  $\Leftarrow$  Determine the "Recommender" type of model
ParsedData  $\Leftarrow$  Parse the transactional data dict and
download data from external source if needed
HighLevelKernel  $\Leftarrow$  construct computational graph for
Model based on tensor computation/optimization
framework
TrainingJob  $\Leftarrow$  Allocate resources and request training of
Model based on ParsedData and HighLevelKernel
ModelTrainingHandle  $\Leftarrow$  obtain a handle of TrainingJob

```

**Return** ModelTrainingHandle

---

### B. User Apps – migration and fast-development repository

The user created apps repository will consist of both automated generated apps and template-based apps. The generated apps will actually be the results of our automatic software migration engine [4], fully hosted within the execution environment proposed by our proposal.

The template-based apps purpose is to construct a continuously growing repository of online "skeleton" apps that have been designed for a particular horizontal purpose within the area of predictive analytics. The user will be able to configure a certain template or propose a new template for online publication and crowd-testing. The proposed template apps cover the following types of predictive analytics models:

- Multi-variate machine learning regression model templates for product/sales recommendations
- Multi-variate machine learning classification model templates for customer churn prediction
- Unsupervised machine learning clustering model templates for customer segmentation

Finally, both the automated apps and the template based app creation features of our user apps repository will facilitate the fast-development and deployment of customer predictive analytics applications.

### C. Business Apps and Research experiments

The business apps and research experiments repositories will host various predictive analytics online applications that will be able to run the proposed machine learning models within the proposed GPU processing backend. These repositories will contain applications that have been designed/developed using the API provided by the our cloud machine learning execution framework.

## IV. CONCLUSIONS AND FUTURE DEVELOPMENTS

It is our objective to transform the experimental outcomes of the presented paper and the underlining architecture into a full community and crowd-sourced repository of both scientific and business-oriented machine learning experiments. This objective will effectively transform the proposed applications repository in a incubatory for intelligent predictive analytics applications. In order to achieve this, we will address two different types of communities: the auto-generated applications community (User Apps) and the externally developed and uploaded applications community (Experiments).

The main difference between the User Apps repository and the Business and Research Experiments repository is that in the first case the platform users are fully relying on our cloud architecture for the entire development lifecycle of the predictive analytics app. In the second case, the platform users are externally developing the predictive analytics apps based on the resources provided by our cloud framework.

At present time, we are conducting further research and experimentation activities in the areas of automatic legacy software migration for the User Apps repository and also in the area of development and deployment of predictive analytics experiments.

Automatic legacy software migration and provisioning is currently experimented within a cloud infrastructure. An experimental “dataset” of desktop-based ready-to-migrate applications has been constructed consisting of applications written in FoxPro, Visual Basic, Visual C++. For this particular set of applications, we are using our automated migration software based on [4] and the existing source code of the legacy where available. In this particular area we are currently conducting research and experimentation both with functional legacy applications and with partially functional applications (or even user interface scraps). The experiment validation is based on the actual legacy application source code (where available). The source code is necessary for the actual validation of resulting automated migration inferential process.

The second area of experimentation is the actual development and deployment of predictive analytics solutions for business applications. Currently customer behavior predictive analytics experiments are conducted within our proposed GPU-augmented machine learning framework based on our REST machine learning model execution framework and TensorFlow [6] backend. Cross-validation and benchmarking of results is conducted using the well-known scientific and general machine learning frameworks such as: (a) Python Sci-Kit Machine Learning package [12] and (b) R package “caret” (CRAN hosted) [13]. This second area of experimentation is currently limited to translation and REST provisioning of machine learning algorithms such as:

- GPU optimized SoftMax, linear/logistic regression for large datasets
- GPU optimized recommender systems based on matrix factorization techniques

As an actual result example, we will describe the machine learning recommender system inference performance. Our current experimental environment based on a single Nvidia

PASCAL P5000 GPU card and 24GB RAM. Our target was to produce recommender system inferences for the following experiment:

- 2,000,000 customers transaction database
- more than 10,000,000 transactions in total
- more than 50,000 different products
- product-per-customer average of 5 products

Finally, we managed to produce the full-batch inference with a huge speed-up compared with a classic architecture with 32 CPU and 64GB RAM. For this particular benchmark, we used a classic matrix factorization based recommender system with BoW-based engineered predictor variables and generated 2,000,000 different shallow models (1 model for each customer). We ran the same approach within the 32 CPU/64 GB RAM server using Python SKLearn and R caret packages with parallel computing for the inference process on all available CPU cores.

	Single customer inference (ms)	Batch inference (ms)
<i>Cloudifier</i>	2,601	15,324
<i>SKLearn linear models</i>	1,120	27,438
<i>Caret linear models</i>	1,307	30,732

Table 1: GPU vs CPU performance comparison

As it can be observed from Table 1 for the single case inference where only one customer inferential model is employed the classic CPU approach wins mostly due to the overhead of GPU device placement of the operations. However, for 2,000,000 models batch-inference, the GPU-optimized approach clearly leverages the 2500+ numerical computations cores and manages to surpass the CPU based parallel computing approach.

Finally, we will mention that our current focus at present time is to research new GPU optimization and implementation approaches for other shallow machine learning algorithms than we currently have implemented. All these methods will be freely provisioned and available for academic purposes though the proposed REST framework.

## V. BIBLIOGRAPHY

- [1] A. Agrawal, J. Gans and A. Goldfarb, "Managing the Machines," *HBR*, 2016.
- [2] O. Vinyals and Q. Le, "A Neural Conversational Model," in *ICML Deep Learning Workshop 2015*, 2015.
- [3] C. V, Q. B. Nguyen and S. Pankanti, "Deep Learning Ensembles for Melanoma Recognition in Dermoscopy Images," *Journal of Research and Development*, 2016.

- [4] A. Damian and N. Tapus, "Model Architecture for Automatic Translation and Migration of Legacy Applications to Cloud Computing Environments," in *CSCS*, Bucharest, 2017.
- [5] M. Bihis and S. Roychowdhury, "A generalized flow for multi-class and binary classification tasks: An Azure ML approach," in *Big Data (Big Data), 2015 IEEE International Conference on*, 2015.
- [6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia and R. Jozefowicz, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *Corenell University Library*; arXiv:1603.04467, 2016.
- [7] J. Bergstra, O. Breuleux, P. L. R. Pascanu, O. Delalleau, G. Desjardins, I. Goodfellow, A. Bergeron, Y. Bengio and P. Kaelbling, "Theano: A CPU and GPU Math Compiler in Python," in *PROC. OF THE 9th PYTHON IN SCIENCE CONF. (SCIPY 2010)*, Austin, Texas, 2010.
- [8] A. Halevy, P. Norvig and F. Pereira, "The Unreasonable Effectiveness of Data," *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8-12, 2009.
- [9] J. Stone, D. Gohara and G. Shi, "OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems," *Computing in Science & Engineering*, vol. 12, no. 3, 2010.
- [10] J. Ghorpade, J. Parande and M. Kulkarni, "GPGPU PROCESSING IN CUDA ARCHITECTURE," *Advanced Computing: An International Journal ( ACIJ )*, vol. 3, 2012.
- [11] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," *University of California, Irvine*, 2000.
- [12] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort and J. G. Saclay, "API design for machine learning software: experiences from the scikit-learn project," *European Conference on Machine Learning and Principles and Practices of Knowledge Discovery in Databases (2013)*, 2013.
- [13] M. Kuhn, "Caret package," *Journal of Statistical Software*, 2008.