Efficient and versatile data analytics for deep networks

D. Garcia-Gasulla[1], J. Moreno-Vázquez[1], J.A. Espinosa-Oviedo[1], J. Conejero[1], G. Vargas-Solar[3], R.M. Badia[1], U. Cortés[12], T. Suzumura[14]

[1] Barcelona Super Computing Centre
{dario.garcia, jonatan.moreno, javiera.espinosa, francisco.conejero, rosa.m.badia}@bsc.es
[2] Universitat Politècnica de Catalunya, Spain
ia@cs.upc.edu
[3] CNRS, LIG-LAFMIA, Saint Martin d'Hères, France
genoveva.vargas@imag.fr
[4] IBM T.J Watson Research Center, USA
suzumura@acm.org

Abstract. Deep networks (DN) perform cognitive tasks related with image and text at human-level. To extract and exploit the knowledge coded within these networks we propose a framework which combines state-of-the-art technology in parallelization, storage and analysis. Our goal, to make DN models available to all data scientists.

Keywords: deep learning, data analytics, parallelism, data storage

1 Introduction

Deep learning networks (DN) learn data representations through the millions of features composing them [13]. This provides a trained DN with a rich representation language to, for example, perform object classification at a human-level. In the case of images, each feature learnt by a convolutional neural network provides a significant piece of visual information on the image, even if these features are not optimal for discrimination (only the top layer features are). In a sense, by considering all feature values for a given image, one is in fact looking at everything the network *sees* in an image.

The goal of the Tiramisu environment (see Figure 1) is extracting and exploiting the knowledge coded within DN. Therefore, using a pre-trained network (*e.g.,* GoogLeNet on ImageNet data [18]), and extracting the features through a deep learning toolkit ((DLT) *e.g.,* Caffe[12]), Tiramisu builds alternative data representations and runs various analytics methods on top of them [9]. This paper presents the Tiramisu architecture, focusing on its unique requirements in terms of parallelism and data management and storage.

The remainder of the paper is organized as follows. Section 2 describes and compares our proposal with works related to data storage, data analytics and

parallel runtime environments. Of course, few works address the use of data analytics for DN reasoning about in memory data organization for ensuring efficient executions. We argue about the interest of this challenge. Section 3 describes our approach for ensuring an efficient execution of the DN by managing data: modelling them according to different data structures, indexing and collocating them looking for performance in parallel executions, storing partial and final results. Section 4 reports the first experiments that we performed using the last version of Tiramisu and the PyCOMPS[19] execution runtime on top of the MareNostrum infrastructure. Section 5 concludes the paper and discusses future work.

2 Related work

The emergence of Big Data some years ago denoted the challenge of dealing with huge collections of heterogeneous data continuously produced and to be exploited through data analytics processes. First approaches have addressed data volume and processing scalability challenges. Solutions can be described as balancing delivery of physical services such as:

1. hardware (computing, storage and memory),
2. communication (bandwidth and reliability) and scheduling,
3. greedy analytics and mining processes with high in-memory and computing cycles requirements.

Due to the democratization Big Data management and processing is no longer only associated to scientific applications with prediction, analytics requirements, artificial intelligence algorithms requirements also call for Big Data aware management related to the understanding and automatic control of complex systems, to the decision making in critical and non-critical situations.

2.1 Data storage and distribution

Data management in many data analytics workflows is guided by the RUM conjecture (Read, Update, Memory (or storage) overhead) [4]. Several platforms address some aspect of the problem like Big Data stacks [7, 1]; data processing environments (*e.g.,* Hadoop, Spark, CaffeonSpark); data stores dealing with the CAP (consistency, atomicity and partition tolerance) theorem (*e.g.,* NoSQL's); and distributed file systems (*e.g.,* HDFS). The principle is to define API's (application programming interface) to be used by programs to interact with distributed data management layers that can cope with distributed and parallel architectures.

In the distributed systems domain objects persistence has been an important issue addressed already by consolidated middleware such as JBOSS and PiJAMA. The new exascale requirements introduced by greedy processes often related to Big Data processing has introduced objects persistence again. In order for exascale and/or Big Data systems to deliver the needed IO performance, new

storage devices such as NVRAM or Storage Class Memories (SCM) need to be included into the storage/memory hierarchy. Given that the nature of these new devices will be closer to memory than to storage (low latencies, high bandwidth, and byte-addressable interface) using them as block devices for a file system does not seem to be the best option. DataClay[5], proposes object storage to enable both the programmer, and DataClay, to take full advantage of the coming high-performance and byte-addressable storage devices. Today, given the lack of such devices, DataClay performs a mapping of such abstractions to key-value stores such as Kinetic drives from Seagate[6].

Data structures and associated functions are sometimes more important for some requirements rather than non functional properties like RUM or CAP. Non-relational databases have emerged as solutions when dealing with huge data sets and massive query work load. These systems have been redesigned to achieve scalability and availability at the cost of providing only a reduced set of low-level data management functions, thus forcing the client application to take care of complex logic. Existing approaches like Hecuba [2], Model2Roo [6] provide tools and interfaces, to ensure an efficient and global interaction with non-relational technologies.

The large spectrum of data persistence and management solutions are adapted for addressing workloads associated with Big Data volumes; and either simple read write operations or with more complex data processing tasks. The challenge today is choosing the right data management combination of tools for variable application requirements and architecture characteristics. Plasticity of solutions is from our point of view the most important property of such tools combination. This paper focuses the evolving persistence needs of data centric workflows.

2.2 Data analytics for data science

Methods for querying and mining Big Data are fundamentally different from traditional statistical analysis on small samples. Big Data are often noisy, dynamic, heterogeneous, inter-related and untrustworthy. Nevertheless, even noisy Big Data could be more valuable than tiny samples. Indeed, general statistics obtained from frequent patterns and correlation analysis usually overpower individual fluctuations and often disclose more reliable hidden patterns and knowledge.

Big Data forces to view data mathematically (e.g., measures, values distribution) first and establish a context for it later. For instance, how can researchers use statistical tools and computer technologies to identify meaningful patterns of information? How shall significant data correlations be interpreted? What is the role of traditional forms of scientific theorizing and analytic models in assessing data? What you really want to be doing is looking at the whole data set in ways

[5] Toni Cortes, Anna Queralt, Jonathan Mart, and Jesus Labarta, DataClay: towards usable and shareable storage, http://www.exascale.org/bdec/sites/www.exascale.org.bdec/files/whitepapers/

[6] http://www.seagate.com

that tell you things and answers questions that you are not asking. All these questions call for well-adapted infrastructures that can efficiently organize data, evaluate and optimize queries, and execute algorithms that require important computing and memory resources.

Big Data are enabling the next generation of interactive data analysis with real-time answers. In the future, queries towards Big Data will be automatically generated for content creation on websites, to populate hot-lists or recommendations, and to provide an ad hoc analysis of data sets to decide whether to keep or to discard them [11]. Scaling complex query processing techniques to terabytes while enabling interactive response times is a major open research problem today.

Analytical pipelines can often involve multiple steps, with built-in assumptions. By studying how best to capture, store, and query provenance, it is possible to create an infrastructure to interpret analytical results and to repeat the analysis with different assumptions, parameters, or data sets. Frequently, it is data exploration and visualization that allow Big Data to unleash its true impact. Visualization can help to produce and comprehend insights from Big Data. Visual.ly, Tableau, Vizify, D3.js, R, are simple and powerful tools for quickly discovering new things in increasingly large datasets.

2.3 Parallel runtime environments

Today maybe because of the emergence of Big Data and greedy algorithms and applications requiring computing resources, parallel architectures have come back in the arena. There are different kinds of computing, memory and storage resources providers that adopt their own method for delivering such resources for executing programs. According to [14] there are three categories of resources provision: PaaS frameworks, programming models for computing intensive workloads and programming models for Big Data.

Platform-as-a-Service (PaaS) offer APIs to write applications. For example, in the Microsoft Azure Cloud programming model applications are structured according to roles, which use APIs to communicate (queues) and to access persistent storage (blobs and tables). Microsoft Generic Worker proposes a mechanism to develop a Worker Role that eases the porting of legacy code in the Azure platform [17]. Google App Engine provides libraries to invoke external services and queue units of work (tasks) for execution; furthermore, it allows to run applications programmed in the MapReduce model. Data transfer and synchronization are handled automatically by the runtime. Environments for computing workload intensive applications use in general the bag of tasks execution model conceiving an application as composed of independent parallel tasks. For example, the Cloud BigJob, Amazon EC2, Eucalyptus and Nimbus Clouds and ProActive that offers a resource manager developed to mix Cloud and Grid resources [15, 3]. MapReduce programming is maybe the most prominent programming model for data intensive applications. Mapreduce based runtime environments provide good performance on cloud architectures above all on data analytics

tasks working on large data collections. Microsoft Daytona[7] proposes an iterative MapReduce runtime for Windows Azure to support data analytics and machine learning algorithms. Twister [10] is an enhanced MapReduce runtime with an extended programming model for iterative MapReduce computations. Hadoop [5] is the most popular open source implementation of MapReduce on top of the Hadoop Distributed File System (HDFS). The use of Hadoop avoids the lock into a specific platform allowing to execute the same MapReduce application on any Hadoop compliant service, as the Amazon Elastic MapReduce[8].

2.4 Discussion

With the new computing, storage and memory requirements the use of high performance architectures providing such resources has increased and is somehow democratized particularly with the emergence of the cloud. Data management requirements have to be revisited they vary from simple storage coupled with read/write requirements with reasonable data processing performance needs to real critical applications dealing with huge data collections to be processed by greedy algorithms. In such settings it is possible to exploit parallelism for processing data by increasing availability and storage reliability thanks to duplication and collocation. For data centric parallel tasks, strategies for dealing with data persistence, efficient read and write in memory operations, data preparation (transformation, cleaning, collocation) must be integrated within the runtime environments. These environments must provide interfaces, either API or annotations to enable programmers to transparently tag their data with their associated properties. It will be up to the runtime environment to provide data management services that can ensure efficient data I/O thanks to well adapted operations with underlying processes for providing the right data in the right moment and in the right place.

The challenge is to have tools that can change their preferences towards RUM and provide elastic strategies for implementing these operations. Such strategies should evolve as data acquire different structures and semantics as a result of the data processing operations applied on them.

3 Deep data storage and process management

Figure 1 illustrates the general architecture of the Tiramisu framework. The Tiramisu data-flow originates on a DLT. Given a set of images Tiramisu obtains a large set of values for each one (millions of floats), corresponding to the neural activations taking place within the deep learning model. These values are processed, and represented in various internal Tiramisu models (vectors, graphs).

[7] http://research.microsoft.com/en-us/projects/daytona/ Last time visited 29.03.2016

[8] Amazon elastic map reduce. http://aws.amazon.com/documentation/elasticmapreduce/. Last visited on 30.04.2016.

Once the representation models have been built, Tiramisu can call upon analytics which may or may not be native to Tiramisu. Given the huge data Tiramisu operates with, both parallelism and storage become cornerstones of its design.

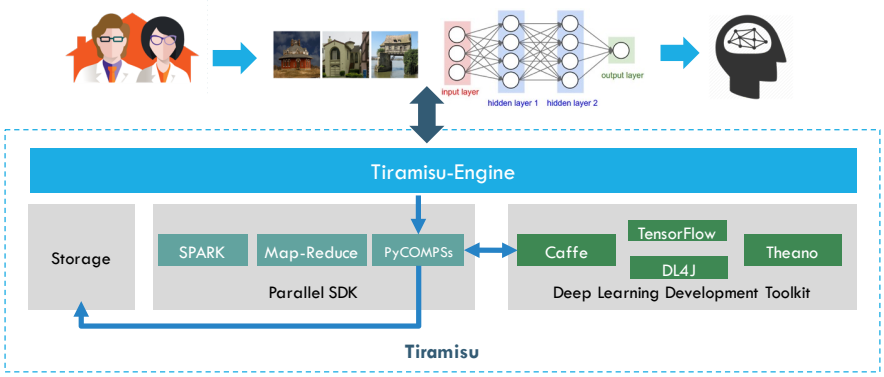

Fig. 1: Tiramisu knowledge extraction environment

Storage of data is key for performance, as large volumes of data are to be accessed, operated with and persisted on each execution. Depending on the specific representation of the data, a different type of persistence may be used, which must also be consistent with the set of tools providing parallel analytic services.

The challenge for Tiramisu is to provide an architecture that (i) connects to each of those components, and (ii) accesses the most appropriate on each specific context.

3.1 Parallelism and storage requirements in Tiramisu

The classification approach in Tiramisu is performed by a data centric workflow. It consists in a sequence of parallel analytics operations that process a data set of images and generate subsequent data collections with incremental content and semantics. Figure 2 illustrates the general reverse engineering principle of Tiramisu.

The three main phases of the workflow are: image feature extraction, image class aggregation, and analytics. The workflow is based in three types of data collections: a set of tagged images stemming from an existing benchmark, a graph representing a thesaurus; multidimensional vectors associated to metadata attribute value structures, and tagged matrices. The first phase uses an initial image collection and executes a set of parallel jobs each one performing an image feature extraction, as images are processed by a DN. Although typically this process is done with the purpose of classification (as the output of the DN is processed through a classifier), the Tiramisu workflow focuses on what the DN percieves in the image, extracting neuron activations at various

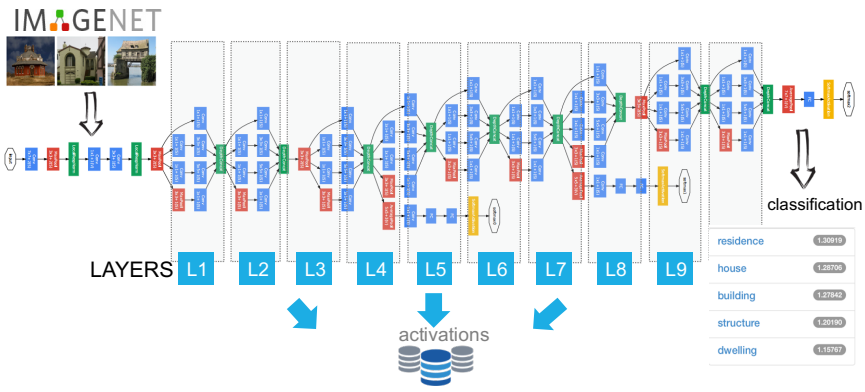

Fig. 2: Tiramisu reverse engineering principle

levels as the image is processed. Such information is captured in a collection of multi-dimensional vectors tagged with meta-data describing the conditions in which such data was produced (e.g., DN model, number and size of layers, image properties, etc.). Meta-data being codified by attribute - value structures. For example the concept associated to the image (*this is the image of a church*).

Single images are imperfect representations of abstract entities. A single image is limited by perspective, illumination, context and many other variables. To obtain abstract concept representations Tiramisu aggregates single images representations into image class representations. All images belonging to the same class (being labeled as such) will be aggregated to build a unique class structure. This aggregation is done at feature level (neuron by neuron) through an arithmetic mean, and results in class vectors of equal size and structure as the image vectors used to compute them.

In order to understand the way the DN percieves images and entities, several operations are performed on the resultant vector embedding space. These operations seek syntactic and semantic regularities (see Figure 3). Regularities emerge as a result of performing arithmetic operations on the continuous space defined by these vectors. This is also a parallel task that leads to new sets of synthetically produced vectors that describe possible neuron activation patterns that could be activated when analyzing images of similar type (e.g., the images of a church and a mosque other examples are shown in Figure 3).

Besides arithmetics, the Tiramisu workflow also allows to performs data analytics based on similarity matrices obtained from the vector embedding space. Matrices that are to be explored and exploited by data scientists. Given the vector-space representation we extract the distance between every pair of image classes captured [8]. We do so using the cosine similarity, and obtain, for every image class, a sorted list of the rest of classes based on their distance, a ranking. To validate this process, an analogous process is done through WordNet, using the image class labels to nd their closest classes in the lexical database. Thus we have, for each image class, two rankings; one according to the vector-

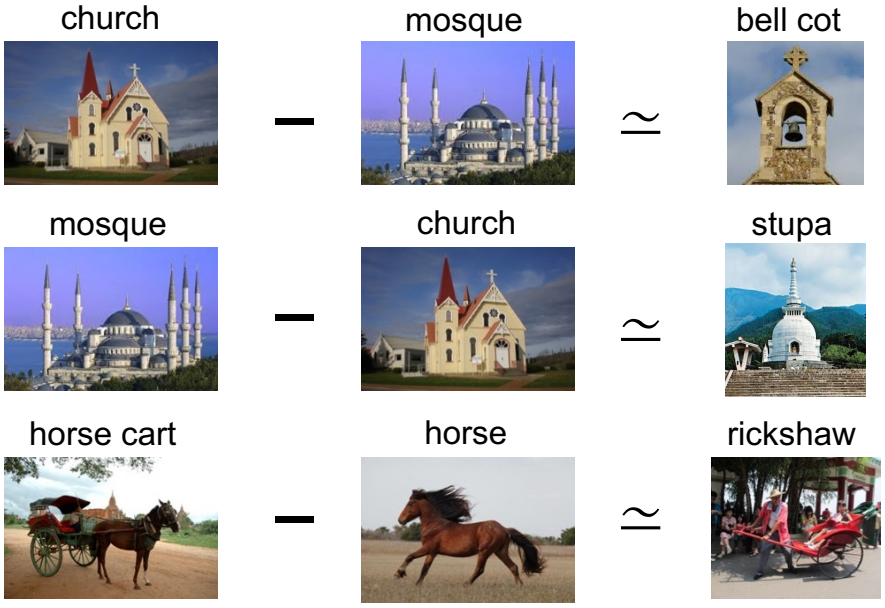

Fig. 3: Patterns analysis operations

space and one according to WordNet. For each pair we compute Spearmans ρ, which provides a measure of ranking correlation. We perform this study for the rankings of the 1,000 image classes, resulting in a distribution of correlations. This distribution tells how close both semantic spaces are. These are also greedy operations executed as parallel tasks that produce complex data sets.

The parallelism and storage requirements in Tiramisu depend on the analytics to be executed. On one hand, the parallel execution of various DLT instances to process various inputs must be handled by a general purpose parallel programming model (*e.g.,* PyCOMPSs) [19]. Other data analytics processes, such as vector distances or graph clustering, may be provided by third parties which already implement those methods (*e.g.,* Spark, ScaleGraph). The challenge for the Tiramisu architecture is therefore to connect to each of those components, and to access the most appropriate on each specific context.

An example of application which may benefit from the parallel execution of DLT instances is the vector visualization process integrated in Tiramisu. Since class vectors correspond to activations from the neural network, for a given vector the reverse process can be performed, in order to generate a visual representation of the vector. When dealing with thousands of vectors, this vector-to-image process can be executed in parallel and independently for each vector through DLT calls. A sample of the visualizations than can be obtained is shown in Figure 4

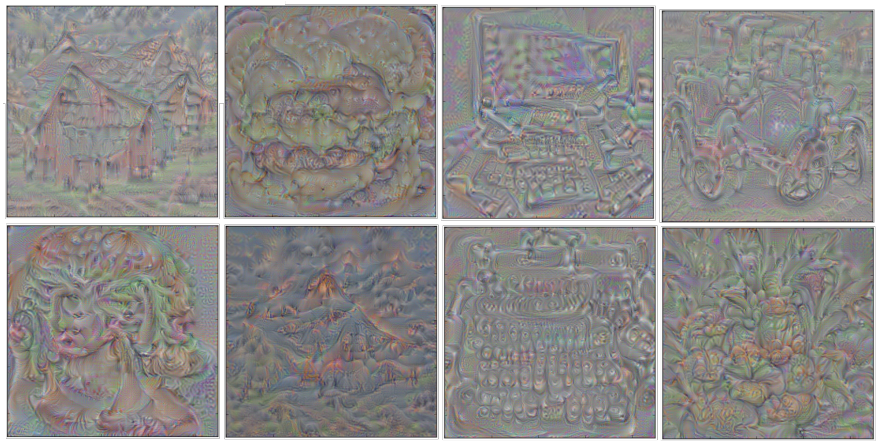

Fig. 4: Sample of visual reconstructions of vector classes. Top to bottom, left to right, classes belong to concepts: barn, cheeseburger, laptop, Model T, wig, volcano, typewriter and pineapple. Best seen in color.

3.2 Managing data for Tiramisu

Such operations require prepared input data collections (tagged and indexed), clever data collocation across processing nodes and results storage.

Figure 5 shows the data management aspects for the first processing phase that starts from an image data collections and produces so called ImageVectors. A typical Tiramisu object containing an ImageVector contains a set of meta-data (*e.g.,* source image), and a set of values (*e.g.,* millions of floats) representing the neurons activated at every level of the DN. With respect to this phase images are processed by PyCOMPS process that execute an extended Caffe executor with monitors to harvest information about activated neurons. Besides data collocation there is no read or location strategy (persistence) to contribute to the scale up of such task. At the end of the execution partial results are gathered by the master node and stored. As shown in the figure results are indexed according to the WordNet classes they belong to. Indeed, every ImageVector is associated to an image associated to a Wordnet concept. So they are grouped accordingly and stored in a store that provides tools for sharding[9] data according to the index.

Figure 6 shows the data processing flow during the arithmetic operation tasks. An indexed data collection is taken as input and sharded and eventually duplicated across the nodes of the architecture according to the index groups. Therefore every operation is performed on each of the groups. Having the Im-

[9] Sharding in the database jargon means creating fragments of a given data set according to a specific criterion. Shards are not necessarily independent, that is, data can be duplicated across shards.

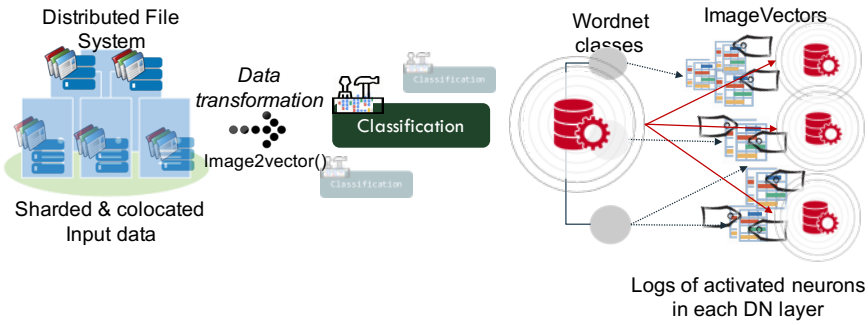

Fig. 5: Fragmenting images data sets for delivering classification input

ageVectors indexed collection reduces data preparation and distribution time which is penalizing for this process.

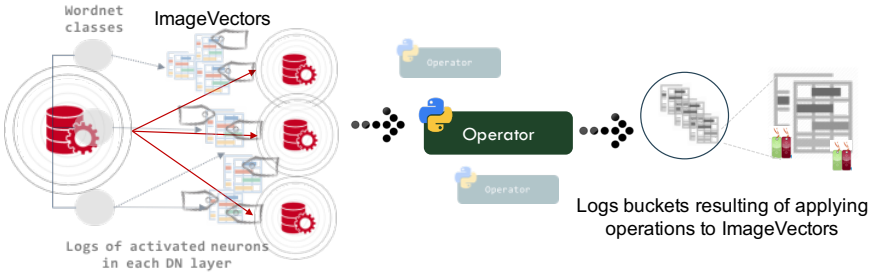

Fig. 6: Indexing image vectors for computing arithmetic operations

The results of the arithmetic operations generate ClassImageVectors that correspond to images that could have been classified by DN and that are mapped to WordNet synsets. Such ClassImageVectors are indexed together with the initial groups, and then stored in the shards managed by the NoSQL system. They guide the distribution of the analytics process of the last phase of the Tiramisu workflow shown in Figure 7.

The last analytics phase of the Tiramisu workflow is the most challenging one in terms of data management. As shown in Figure 7, the ImageVectors are processed for generating matrices that show the similarity among the originally classified images represented by ImageVectors and synthetic ones ClassImageVectors. These matrices are composed with respect to some representative attributes chosen by a data scientist. In order to enable the exploration of data analytics results, we use a column oriented store that provides multidimensional records and associated operations for querying them.

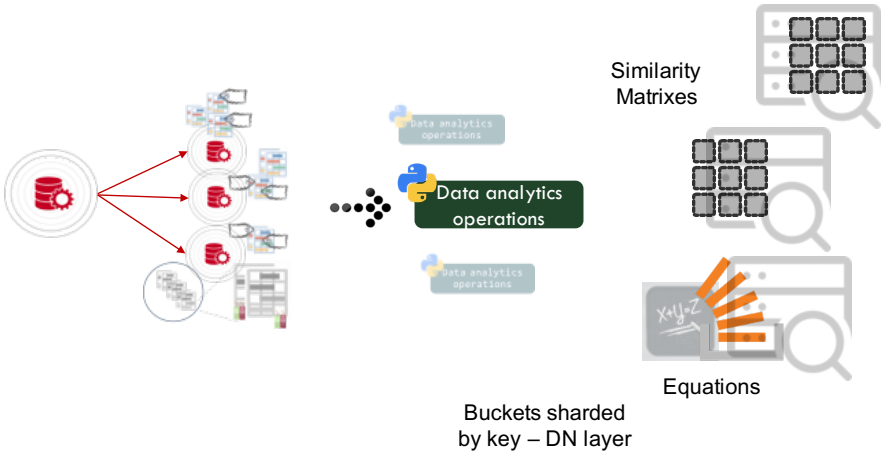

Fig. 7: Querying and executing analytics process

Depending on the specific data representation, different types of persistence are used, and they are consistent with the set of tools providing parallel analytic services. Indeed, well adapted data look-up, querying and exploration tools ensure transparent access to data scientists.

4 Experimentation

We conducted experiments for including elastic data persistence strategies (storage, indexing, look up) of different structures as the Tirsamisu enriches data collections for providing its data science environment.

Experimental setting We used the GoogLeNet deep convolutional neural network architecture [18], a very deep network (22 layers) which obtained the best results at the ILSVRC14 visual recognition challenge [16]. We used the GoogLeNet pretrained model available in the Caffe deep learning framework [12] trained using the 1.2 Million images of the test set. This model was trained to discriminate among the 1000 leaf-node categories of the ImageNet hierarchy. GoogLeNet is based on so called inception modules. The available model contains 9 of those.

For executing our parallel tasks we used the PyCOMPSs runtime system which automatically finds the data dependencies between tasks based on the direction of their parameters. With this information, it dynamically builds a task dependency graph, executed in the target infrastructure. The experiments described were done on the MareNostrum supercomputer, using two Intel Sandy Bridge-EP E5-2670/1600 20M 8-core at 2.6 GHz and64 GB of RAM. We used 20,000 images of the ImageNet validation set. All code used to process the activation files is available at https://github.com/dariogarcia/tiramisu, including the python scripts used to build figures and graphs.

Introducing persistence into the analytics phases To build the vector representations we used the output of those modules, particularly the three 1x1, 3x3 and 5x5 convolution layers. When an image is run through the trained network for its classification, these 27 different layers combined produce over 1 Million activations, expressing the presence and relevance of 1 Million different visual patterns in the input image. We treat each of those activations as an independent variable describing distinct visual information of the image. Thus, our image high-dimensional, sparse vector representation is composed by over 1 Million continuous variables. Besides images, the only source of information available for each of the 1,000 contained classes is their label, which is mapped to a WordNet concept[10]. Thus, WordNet captures part of the semantics of the classes represented as vectors.

We developed a *"sur mesure"* data management approach trying to ease data access according to the RUM conjecture. Our objective of this experiment was to embed data management within the Python analytics programs of Tiramisu trying to reduce communication overhead and ensuring the plasticity of the Read strategies (from disk and memory), since there were few updates in the first and third workflow phases. In this experiment we concentrated on persistence of the first and second phases. We are currently working with the solution Hecuba for dealing with the third phase in order to profit from a column oriented data store. Figure 8 gives a general overview of the data we had to manipulate across the Tiramisu workflow. As show in the UML class diagram we suppose for the time being a transformation process between the data seen by the application and the stored data. In this experiment we mainly relayed on the sharding and indexing facility of MongoDB and we decided to accept the transformation overhead only in the case of this experiment. Our objective being to measure how data indexing and sharding, coupled with read strategies could help to scale up the Tiramisu workflow tasks.

We used MongoDB for generating an index and sharding the data according to the approach introduced in Section 3. ImageVectors were transformed into complex documents that were sharded according to their associated semantic classes. The storage process was embedded withing the Python scripts of Tiramisu with the hypothesis of integrating *"sur mesure"* persistence services for the problem. The resulting strategy was to reduce the initial lookup of ImageVectors with cost O(n) to O(log(n)).

5 Conclusions and perspectives

This paper introduced the Tiramisu data oriented workflow as a sequence of parallel analytics operations for processing large data sets of images. The workflow generates data collections with incremental content (float vectors) and semantics (meta-data) under a write once, read many approach. From our first experiments

[10] WordNet, a lexical database composed groups of synonym words named synsets associate terms through semantic relations such as hypernym/hyponym

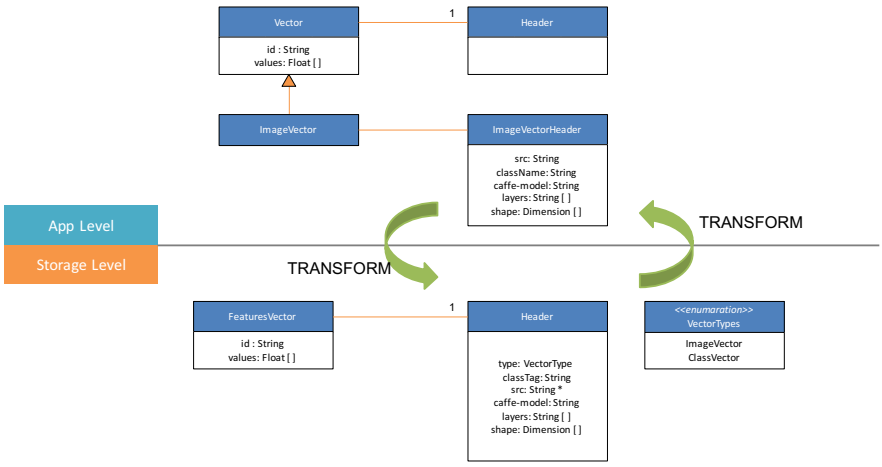

Fig. 8: Overview of the Tiramisu workflow data

we learnt that operations for knowledge extraction require: clever data collocation across processing nodes and results storage. Depending on the specific data representation, different types of persistence may be used, which must be consistent with the set of tools providing parallel analytic services. The results of this first experiment show that developing the right RUM strategies we will be able to provide an intelligent data management service that can make the analytics processes more efficient. The final step will be to integrate the data management techniques within the PyCOMPS persistence model to avoid impedance and gain en simplicity, transparency of use and flexibility.

Acknowledgements

This work was partially supported by the IBM/BSC Technology Center for Supercomputing (Joint Study Agreement, No. W156463) and the French Council of Scientific Research through its UMI 3175.).

References

1. A. Alexandrov, R. Bergmann, S. Ewen, J.C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, et al. The stratosphere platform for big data analytics. *The VLDB Journal*, 23(6):939–964, 2014.
2. Guillem Alomar, Yolanda Becerra Fontal, and Jordi Torres Viñals. Hecuba: Nosql made easy. In *BSC Doctoral Symposium (2nd: 2015: Barcelona)*, pages 136–137. Barcelona Supercomputing Center, 2015.
3. Brian Amedro, Françoise Baude, Denis Caromel, Christian Delbé, Imen Filali, Fabrice Huet, Elton Mathias, and Oleg Smirnov. An efficient framework for running applications on clusters, grids, and clouds. In *Cloud Computing*, pages 163–178. Springer, 2010.

4. M. Athanassoulis, M. Kester, L. Maas, R. Stoica, S. Idreos, A. Ailamaki, and M. Callaghan. Designing access methods: The rum conjecture. In *International Conference on Extending Database Technology (EDBT)*, 2016.
5. Dhruba Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 11(2007):21, 2007.
6. Juan Carlos Castrejón, Rosa López-Landa, and Rafael Lozano. Model2roo: A model driven approach for web application development based on the eclipse modeling framework and spring roo. In *Electrical Communications and Computers (CONIELECOMP), 2011 21st International Conference on*, pages 82–87. IEEE, 2011.
7. Matthew Franklin. The berkeley data analytics stack: Present and future. In *Big Data, 2013 IEEE International Conference on*, pages 2–3. IEEE, 2013.
8. D Garcia-Gasulla, J Béjar, U Cortés, E Ayguadé, and J Labarta. Extracting visual patterns from deep learning representations. *arXiv preprint arXiv:1507.08818*, 2015.
9. D Garcia-Gasulla, J Béjar, U Cortés, E Ayguadé, and J Labarta. A visual embedding for the unsupervised extraction of abstract semantics. *arXiv preprint arXiv:1507.08818*, 2015.
10. Thilina Gunarathne, Bingjing Zhang, Tak-Lon Wu, and Judy Qiu. Scalable parallel computing on clouds using twister4azure iterative mapreduce. *Future Generation Computer Systems*, 29(4):1035–1048, 2013.
11. Stratos Idreos, Ioannis Alagiannis, Ryan Johnson, and Anastasia Ailamaki. Here are my data files. here are my queries. where are my results? In *Proceedings of 5th Biennial Conference on Innovative Data Systems Research*, number EPFL-CONF-161489, 2011.
12. Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014.
13. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
14. Francesc Lordan, Enric Tejedor, Jorge Ejarque, Roger Rafanell, Javier Alvarez, Fabrizio Marozzo, Daniele Lezzi, Raül Sirvent, Domenico Talia, and Rosa M Badia. Servicess: An interoperable programming framework for the cloud. *Journal of grid computing*, 12(1):67–91, 2014.
15. Junjie Peng, Xuejun Zhang, Zhou Lei, Bofeng Zhang, Wu Zhang, and Qing Li. Comparison of several cloud computing platforms. In *Information Science and Engineering (ISISE), 2009 Second International Symposium on*, pages 23–27. IEEE, 2009.
16. Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
17. Yogesh Simmhan, Catharine Van Ingen, Girish Subramanian, and Jie Li. Bridging the gap between desktop and the cloud for escience applications. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 474–481. IEEE, 2010.
18. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

19. Enric Tejedor, Yolanda Becerra, Guillem Alomar, Anna Queralt, Rosa M Badia, Jordi Torres, Toni Cortes, and Jesús Labarta. Pycompss: Parallel computational workflows in python. *International Journal of High Performance Computing Applications*, 2015.