Social Semantic Emotion Analysis for Innovative Multilingual Big Data Analytics Markets

D3.5 Acceleration of Large-scale Emotion Analysis Methods, final version

Project ref. no	H2020 644632
Project acronym	MixedEmotions
Start date of project (dur.)	01 April 2015 (24 Months)
Project full name	Social Semantic Emotion Analysis for Innovative Multilingual Big Data Analytics Markets
Document name	D3.5_Acceleration_FinalVersion
Security (distribution level)	PU
Document due Date	31 December 2016 (Month 21)
Responsible for deliverable	Paradigma Tecnológico
Reply to	cnavarro@paradigmatecnologico.com
Deliverable number	D3.5

Deliverable name	Acceleration of Large-scale Emotion Analysis Methods, final version
Type	Other
Version	Final
WP / Task responsible	WP3 / Paradigma Tecnológico
Contributors	José Víctor Marcos Martín, Carlos Navarro De Martino
EC Project Officer	Martina Eydner

Index

Executive Summary

This document describes the final phase of task T3.2, which is devoted to the optimization of the modules composing the MixedEmotions platform. In this final phase, several strategies have been considered taking into consideration the desired change to a microservices architecture, chosen to provide an easier integration framework.

In the platform, software replication can be employed to obtain a parallelization scheme. This approach consists of the deployment of a given software module in different machines from a cluster[1] or other deployment. As a result, these instances can be used simultaneously, in a parallel approach. This parallelization will be achieved using a microservice architecture, as explained in deliverable D3.2.

This document defines a set of strategies for the efficient use of the tools provided by the MixedEmotions platform, for managing the distribution of the data between the machines in the cluster, and for the identification of sets of subtasks from large processing tasks.

1 Here and thorough the whole document cluster is considered a group of machines, either virtual, physical or a combination of both, deployed in conjunction.

1. Introduction

This document is the fourth deliverable of WP3 “Big Data Platform Architecture and Data Analysis Acceleration”. It describes the strategies adopted for the integration and acceleration of the different modules composing the MixedEmotions platform. The results presented here result from the experiences obtained during the implementation of the strategies presented in the initial version of this document.

The main change from the initial version of this document is that the final architecture is based on microservices, and as a result, most of the modules are distributed. Spark is no longer part of the orchestrator, although its analysis will remain here, as the platform still uses Mesos. Final users could easily develop their own Spark orchestrators and deploy them on Mesos.

Briefly, the purposes of this task are the following:

- Analyzing the algorithms and data flow needed for the efficient processing, emotion analysis and data management of very large multimodal datasets.

- Proposing suitable architectures, breakdown of tasks, and flow of data for acceleration of the time and memory critical algorithms.

- Designing the architectures according to computational requirements and dynamic allocation of resources.

- Checking the behaviour of the algorithms of interest on such architectures.

In the rest of the document, we provide an exhaustive description of the approaches adopted towards the completion of this task, with a focus on the objectives initially posed.

2. Analysis of the algorithms

2.1. Acceleration through parallelization

The MixedEmotions platform is intended for the processing of large volumes of multimodal data in order to enable the automatic and objective recognition of entities, sentiments and emotions. The outcomes derived from the platform can optimize the decision making process of a company as they provide more accurate knowledge of their customers and partners.

In this scenario, one of the functionalities implemented in the MixedEmotions platform is the possibility of analyzing data from different sources and representing such a volume that cannot be easily handled by conventional applications. Therefore, the main approach to accelerate the execution of data processing in MixedEmotions will be **parallelization**.

The adopted strategy follows the divide-and-conquer pattern. It is employed in many sequential algorithms. According to this approach, a problem is solved by splitting it into a number of smaller subproblems, solving them independently, and merging the subsolutions into a solution for the whole problem. The subproblems can be solved directly, or they can in turn be solved using the same divide-and-conquer strategy, leading to an overall recursive program structure. Figure 1 illustrates the execution pattern when compared to the conventional sequential execution.

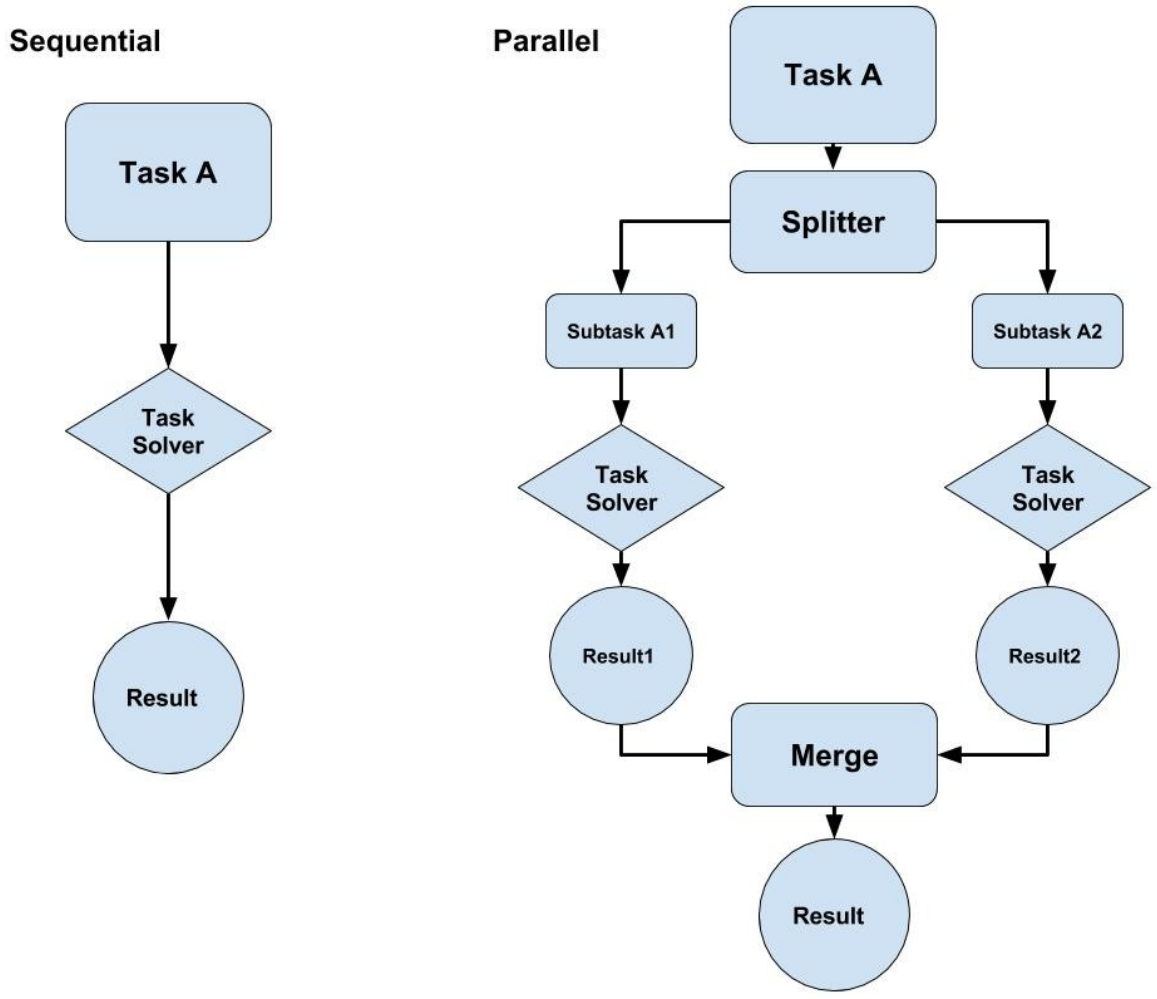

Figure 1. Scheme of the parallelization strategy adopted for the acceleration of the MixedEmotions modules.

As can be observed, the cost of the parallelization strategy is increased complexity. New agents must be included in our system. They correspond to the splitter, which is in charge of dividing the initial task into smaller ones, and the merger, which combines the results from different subtasks to get the global result.

In addition, parallelization will enable the scalability of the solution since distributed and parallelizable technologies suitably adapt to use cases with growing demand for processing needs.

The parallelization of MixedEmotions functionalities involves three main technologies to be included in the architecture of the platform:

- Distributed data storage system. In order to process data in a parallel way, they must be conveniently stored in distributed and scalable storage systems. These technologies are usually accompanied by easy-to-use software which enables connecting capabilities. Hence, read/write operations from/to the database can be accomplished in an easy and efficient way, avoiding undesired bottlenecks. Example of distributed storage systems are HDFS, Cassandra, MongoDB and Elasticsearch.

- Distributed data processing. A specific framework for distributed data processing is adopted. This software is responsible for the definition of subtasks from a given large processing task (task splitting) and the coordination of the results for each of these subtasks (merging results). These actions correspond to the elements included in the system due to parallelization, as detailed in the previous scheme.

- Parallelization of services. Similar to the previous point, this technology consists in having the subtasks divided into services, with multiple instances of each service being distributed alongside each other on a cluster. This is the main focus of the platform, following its microservice architecture, although the platform is flexible enough to support the other two strategies, given the appropriate tools.

Once the inclusion of distributed technologies in the platform is assumed, strategies for the acceleration of the modules will be developed to get the most out of these technologies. The following points of the document describe the actions and strategies designed for the achievement of this goal.

2.2. Initial assessment of the MixedEmotions modules

As an initial step for this task, we analyzed each of the modules that compose the MixedEmotions platform. Two types of modules were identified according to their data processing functionality:

1) Distributed processing modules. The distributed processing modules provide advanced data processing capabilities in order to identify sentiments, recognize emotions and analyze network connections. These modules are equally applied to a large number of entries and the computation of each entry does not depend on the results from the others. Therefore, processing can be directly carried out in a parallel manner.

   The input to distributed processing modules consists of a large amount of small sized items, and thus they suitably adapt to parallel processing. In addition, the exact size of the dataset to be processed is known in advance, which makes the parallelization approach feasible.

2) Non-distributed modules. These modules enable concrete actions on the data, which are generally carried out at the beginning of the processing pipeline. Due to their nature (i.e., the technology on which the module is based or the infrastructure required for its deployment), the parallelization of these modules is not always recommended. On one hand there are the ingestion modules, such as the Twitter Crawler that cannot be easily replicated. First, because API restrictions will make replicated instances useless, as a single instance can make more queries in an hour than its hourly quota. There is also Kibi, which is the analytics and

visualization tool, that could be replicated. However the only advantage of having a replica would be to have a backup instance. Lastly there is the video emotion extraction module that can be replicated in an usual way. However, as it is a very processing intensive module, it is designed to use GPUs for processing.

2.3. Approaches for MixedEmotions modules acceleration

The proposed division of the modules has directly influenced the definition of the platform architecture, as explained in deliverable D3.1. Briefly, non-distributed modules are available as standalone applications. Hence, they are hosted by a single machine, running in a local mode. On the other hand, distributed processing modules are available as parallelizable tools. Then, the user is the one to decide between using them for scalable and distributed processing or running them in a local mode. The adopted orchestrator determines which one of the two modes will be used.

The following subtasks are identified in order to accomplish the integration and acceleration of the modules in the platform:

- In the case of non-distributed modules, since they will run in a local mode, the integration will involve the definition of the format of both their inputs and outputs. In addition, the communication of these modules with other resources of the platform needs to be addressed. For instance, the capability for writing the output data from the module in the storage system of the platform is one of the integration and acceleration actions to be addressed. For the integration of these non-distributed modules, each of them has been thoroughly analysed. For instance, after the analysis of the social media crawlers, they were conveniently modified to provide the output in accordance with the format expected by the rest of the modules in the platform. In this case, a data model was defined to store the items to be processed. These are persisted in files, with each line corresponding to an ingested item. This will be the case for some external modules such as the recommendation engine or, in general, modules without a Docker image.

- For distributed processing algorithms, we identified two different options for their integration and acceleration. First, we considered the implementation of the modules on a distributed processing framework like Spark. This would involve a new implementation of the module, which may require a great effort in some cases. The second option would be to enforce a parallel approach by deploying the module (i.e., the program or the library that provides the functionality) in multiple machines of the cluster. The second option, deploying multiple instances of the services following a microservice architecture was the prefered solution. As a result, the processing capability of the initial standalone software is parallelized. The main challenge in this task was the integration of different technologies. This will be the strategy for most of the modules of the platform, including some external services, and mainly by using Docker images.

In the MixedEmotions platform Mesos will be the tool of choice for deploying the Docker modules. More information about Mesos in this platform can be found at document D3.2, MixedEmotions architecture, final version. To give a brief explanation, Mesos is a resource manager that encapsulates the complexity of managing the resources of a cluster. When deploying a new Docker container, Mesos will look for the necessary resources and deploy it in the machine (virtual or physical) that has the most resources available. Mesos has rapidly become the de facto resource manager in many open source Big Data environments, especially those that use the Mesosphere stack. The only real alternative to Mesos at the moment is the relatively new Docker Swarm.

3. Influence of acceleration strategies on the design of the platform architecture

The final version of the platform architecture described in deliverable D3.2 “Architecture Specification and Platform Implementation, final version” shows which algorithms are used. However, the work done in the first version of this document (D3.4) is maintained, as the algorithms are still valid and could be used in the platform, even if not all components (mainly Spark) are directly present in the final architecture.

The definition of the platform architecture aims at optimally accommodating each of the modules provided by the project partners. It also aims at improving the performance of the modules when large amounts of multimodal data need to be processed. The main decisions on the design of the platform architecture directly influencing the acceleration of the modules and processing algorithms are the following:

- Microservice architecture. The main rationale behind adopting a microservice architecture as the architecture of the MixedEmotions platform was the ease of integration of different modules created by different partners using different libraries and technologies. However, it also allows for data acceleration and processing of large volumes of data via parallelization. Textual data is usually large in number, but small on size, meaning that usually large texts are processed in a phrase by phrase basis or the data simply consists of a large number of smaller texts. That means that it is easy to split the processing and make it in parallel, where each piece of data is processed by a different instance of a service and then results are put together at the end.

- Distributed storage. In order to accelerate the modules in the MixedEmotions platform, a distributed storage system has been proposed. This approach provides the platform with the scalability required to manage large amounts of data. Furthermore, distributed storage is aligned with parallel processing, since data can be simultaneously read/write from several nodes in the cluster. Specifically, Elasticsearch has been chosen for the implementation of distributed data storage. If the final user prefers, HDFS could also be used.

- Spark-based orchestrator. The final version of the architecture has removed the Spark orchestrator. However, a final user could develop their own orchestrator and deploy it using Mesos. As described before, distributed processing algorithms can be used in a parallel way using Spark. Spark would enable different data entries to be processed simultaneously by a given module, which has been previously deployed in several machines of the cluster. That would be more beneficial for modules developed using the Spark framework following a map reduce paradigm.

4. Strategies for data processing acceleration

In this section, possible strategies of acceleration are discussed for data analysis in the MixedEmotions platform. Two main approaches to improve the performance of the processing algorithms have been considered. First, the implementation in a parallel framework like Spark. Second, the parallelization of standalone modules. These are described in the following sections. Additionally, as a common strategy to ensure the interoperability of the modules and improve its efficiency, a common format to represent the information in the platform has been defined.

4.1. Common strategies

As an initial step to ensure the interoperability of the modules composing the MixedEmotions platform, a common output format has been defined. The format must be adopted in order to exchange information with other modules or elements of the platform. To accommodate both structured and semi-structured data, JSON was chosen as the most suitable information format.

The JSON format defined for MixedEmotions extends on JSON-LD (JSON for Linked Data[2]) and incorporates metadata standards for natural language processing (NIF[3]) and representing subjective opinion (Marl[4]) and emotion (Onyx[5]). It provides a standard way to represent detected emotions, subjective opinion and real world entities as well as link these findings to existing knowledge bases such as DBPedia (a public knowledge base build from information in WikiPedia).

In MixedEmotions, specific tools have been created to easily manipulate this JSON-LD format in various programming languages.

4.2. Parallelization of processing modules

The first approach for the acceleration of the algorithms is based on data parallelization using microservices. This is a valid strategy when the processing modules are stateless, each module is light enough to not require all the processing capabilities of a given machine, and each of the entries to be processed are independent of each other. For example, when extracting emotions for a given text, analysis can be carried out in parallel for each sentence, as sentiment in each sentence is independent of the previous sentences. On the other hand, if for example, a topic analysis needs the whole text to work properly, the whole text cannot be divided.

The following strategy is used in a parallel scenario. First, the module is deployed in multiple machines of the cluster (in MixedEmotions that is done using Marathon, Mesos and Docker Images).

2 http://json-ld.org/
3 http://persistence.uni-leipzig.org/nlp2rdf/
4 http://www.gsi.dit.upm.es/ontologies/marl/
5 http://www.gsi.dit.upm.es/ontologies/onyx/

Data to be processed is then distributed between these modules. The modules then process the entries simultaneously. As a result, each copy of the module works locally in each machine but globally a parallel process is obtained.

The key point in this accelerating strategy is the software responsible for distributing the original collection of data entries between the cluster machines. For this purpose, it is necessary that the orchestrator is able to process in parallel, using separate threads or a similar technology. In the MixedEmotions provided orchestrator that is done using the Future library of Scala. That library encapsulates the creation of threads in a simple way. Its parallelization capabilities are proportional to the number of CPUs of the host machine.

4.3. Implementation of processing modules in Spark

Although the decision was taken to move the platform from Spark to a microservices implementation, it is of interest to consider the performance advantages and implementation overheads of Spark . Final users are encouraged to write their own orchestrators and modules and could develop them in Spark if they felt it advantageous. Those modules will integrate seamlessly with the platform as Mesos, that can be used as the framework for running Spark, is present on the platform.

The most efficient way to accelerate processing algorithms is to program them according to the properties of the framework adopted for parallel data processing. A example of this kind framework is Spark, which enables efficient parallel data processing based on in-memory storage. Spark defines the Resilient Distributed Dataset (RDD), which is an abstraction to refer to a collection of elements, all of them of the same type, distributed among the nodes of the cluster.

Programming an algorithm in Spark involves handling RDDs. For instance, the RDD may contain texts from which extract the sentiments or emotions expressed by their author. For the development of Spark programs, the distributed nature of the RDDs must be taken into account. The elements of the RDD are stored in different physical machines and there is no order for these elements. Therefore, transformations of these elements must be equally applied to each of them. Similarly, computations on the elements of the RDD must involve associative and commutative operators. For the implementation of the modules in Spark, the content of these resources is distributed by using a RDD.

RDD are an abstraction that helps programmers to easily implement the map-reduce paradigm. This strategy consists of doing tasks in parallel (map) and then aggregate the results of those tasks (reduce). Programming with RDDs is an advanced question and thus falls outside the scope of this document. However, it is important to at least know that there are two kinds of operations that can be performed with RDDs. First, there are the transformation operation. These operations correspond to the map portion of the paradigm and are performed in a parallel way. They are usually much lighter in computation time. Then, there are the action operations, which correspond to the reduce part of the paradigm and usually will imply the use of an internal shuffle operation. The shuffle operation reorders the data in the cluster and requires much more computing power. It is also worth noting that even the operations of the same type can require different amounts of time to execute. The skilled Spark programmer is able to minimize the use of these heavier operations.

4.4. Comparison between strategies

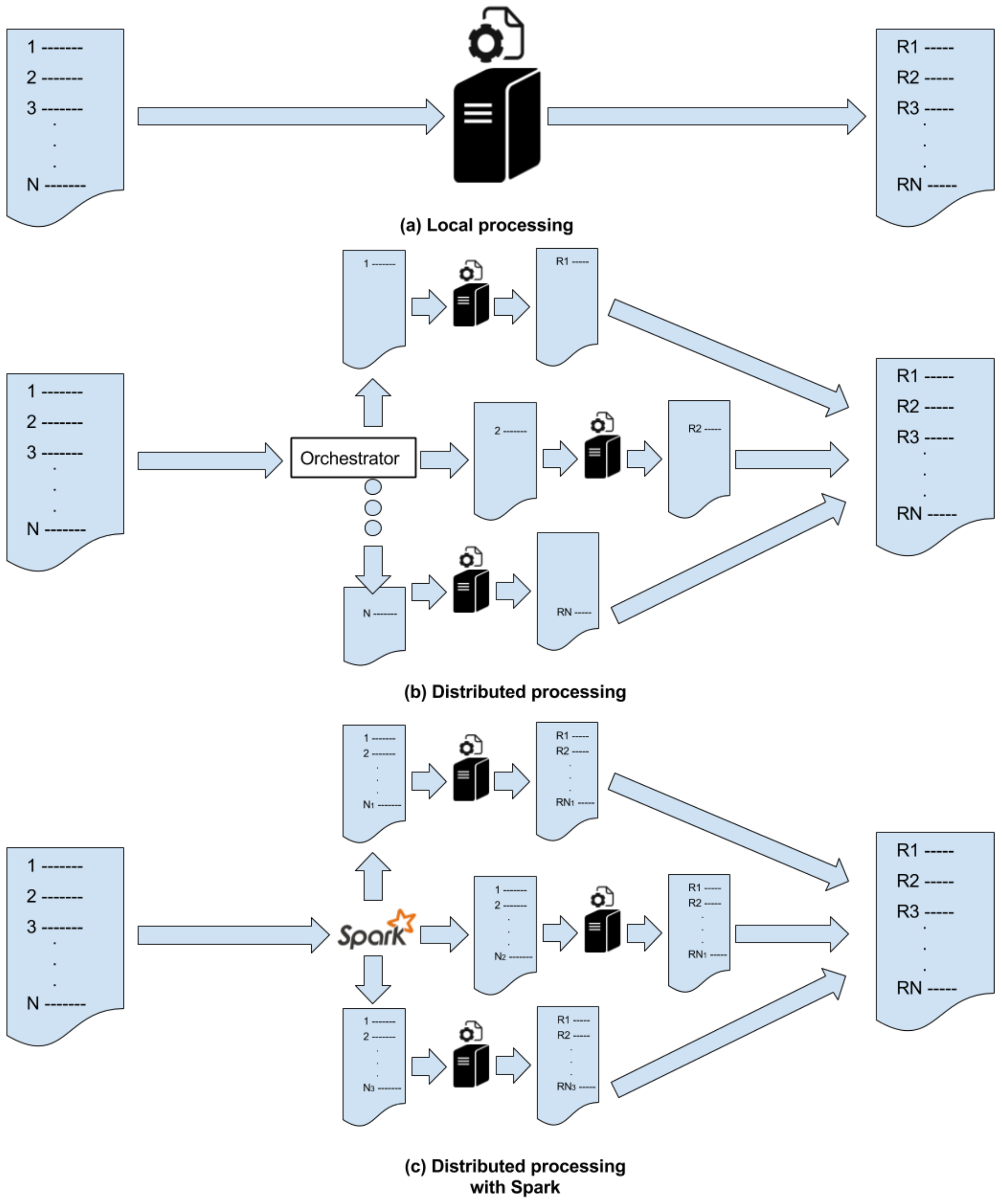

Figure 2. Comparison between strategies.

In this graph the difference between methods of processing is shown. The regular parallelization implies the creation of a thread for each input and then execute the various stages of processing in different instances of the services. A specialized distributed processing framework in spark is more optimized. It will divide the input in a number of partitions depending on the machines of the cluster, then process each of those partitions in their corresponding node and finally will merge the results in the end. This will reduce the network and management latency. However, it implies more complexity when developing the modules and the orchestrators, as both should be created using the Spark framework. Spark also introduces some overhead at the initial stages of creating RDDs, and this means that for small volumes of data it will have a worse performance than a local processing solution. It also requires the modules to be written in certain programming languages and to be completely reworked to fit into its structure and adds more complexity when large files are needed to perform the processing, such as ontologies or trained models. Docker microservices are much easier to maintain, as they can hold any program inside (regardless of programming language) and even the ontologies and training models. In case of an update of the models, the cluster operator will only need to redeploy the corresponding services and, if done correctly, will not even lose availability of the services.

5. Acceleration using specific hardware

Some computational tasks can greatly benefit from specific hardware. Processing throughput measured with respect to equipment cost or power consumption can be improved by more than an order of magnitude. Specifically, the emotion extraction from video module in MixedEmotions rely on convolutional neural networks (head localization, facial expression estimation, identification, age and gender estimation)[6]. The computational cost of convolutional networks is dominated by large matrix multiplications and linear 2D convolution operations which exhibit highly regular structure and are ideal for Single Instruction Multiple Data (SIMD) processors such as Graphical Processing Units (GPU).

We have selected NVIDIA CUDA® Deep Neural Network library (cuDNN) to accelerate the convolutional neural network modules. The library provides efficient implementation of 2D multi-channel convolution, batch normalization, pooling, and other operations used in our networks for recent NVIDIA graphic cards. The obtained speedups are significant. For example, our person identification network takes 194 ms to process a single video frame on a machine with dual 8-core Xeon E5-2630 v3 CPUs with standard version of Caffe deep learning framework[7], 31 ms with specially optimized version of Caffe for Intel processors[8], and only 4 ms on NVIDIA GTX 1080 with cuDNN v5.1. The importance of the speedup is emphasized by the fact that the single GPU costs half of what the two Xeon CPUs cost.

This kind of strategy can greatly increase the performance of certain modules if dedicated servers with the required hardware are available. However this technique is only viable for certain modules deployed in a static manner, which is not compatible with the microservice architecture, which relies in replication and the use of small, non physical, cloud machines. In conclusion, this technique is to be used for certain modules when the hardware is available and not for the platform as a whole.

6 See deliverable D4.6 - Emotion Recognition from Image and Video Content, final version
7 http://caffe.berkeleyvision.org/
8 https://github.com/intel/caffe

6. Conclusions

This deliverable covers the study of strategies and actions taken in regard of the optimization of the software modules that compose the MixedEmotions platform. The experiences obtained from the development of the platform resulted in a change of the platform architecture. It was decided that the trade-off between the performance improvement derived from parallelization using Spark and the greatly increased implementation costs of its use was not worth it. As a result, Spark has been removed from the platform. However, it is still an option for final users of the platform.

We found that ease of integration and expandability were much more important than a small performance improvement. Moreover, for small amounts of data, Spark greatly decreases the efficiency, and thus was not providing the flexibility we were looking for for the platform. Also, though hardware acceleration could be used for certain computationally intensive modules, such as the video processing module, that technique is not suitable for the platform as a whole as it hinders parallelization and scalability.

In the final microservice architecture, parallelization enables the scalability of the platform, while reducing the latency if the processing task must deal with large volumes of data. This approach is flexible and efficient enough for the general needs and offering of the platform. We recommend this approach where scalability and timely analysis are required.