FaceID-Cloud

Face Identification Leveraging Utility and Cloud Computing

Ricardo Caldeira
ricardo.caldeira@ist.utl.pt

Instituto Superior Técnico
INESC-ID

Abstract. Detection and identification of human faces has been an intense area of study in the past decades, with many strategies being proposed with very impressive results. Yet, it is a computationally intensive task, specially in videos that can reach a considerable size. Hence, it is important to develop new solutions to improve the performance of face identification methods. At the moment, one of the most promising IT technologies that can be used to achieve this result is Cloud Computing, allowing for scalable and flexible systems to be built with an easy on-demand access to virtual resources in a pay-as-you-go utility model. The purpose of this work is to study the best way to integrate a face identification method with a cloud infrastructure, and propose a system that leverages cloud resources to greatly improve facial identification performance in huge video databases.

Keywords: Video Face Identification, Cloud Computing, Data-Driven Scheduling, Utility Computing, Scalability, Virtualization, Service-Oriented Architecture

1 Introduction

Over the last couple of decades the world has been witnessing the evolution and expansion of the Information Technology (IT) area at an unbelievably fast rate, specially since the appearance of the Internet and the World Wide Web. This lead to the appearance of several new paradigms that revolutionized the way information is processed, among which there is Cloud Computing. Even though a newcomer, making its debut only during the last few years, all the major IT companies like Google, Amazon and Microsoft are now fully aware of its power and usefulness for IT businesses. Given the widespread of cloud technologies, its potentially limitless processing power and storage, and the facilitated access to these resources on-demand, it makes perfect sense to design new cloud-based systems capable of presenting end-users with out-of-the-box solutions that can be used from virtually any Internet-capable device.

In this context, facial identification of human faces in videos is one of the subjects that could greatly benefit from exploiting the cloud potential, specially when dealing with huge databases of videos that can easily reach the gigabytes. The nature of this problem implies a massive computational effort from the start, since video processing is by itself CPU consuming and the face identification algorithm on top of it only worsens the situation. Yet, it can be seen as belonging to the class of the generically dubbed "embarrassingly parallel" problems, meaning that it is easily divided into sub-problems with no data dependencies between them. Knowing this, the main focus of investigation is on how to bring the two sides together, or in other words, how to implement a parallelized face identification algorithm on top of the distributed environment of Cloud Computing, and it is with this goal in mind that we propose a solution to the problem in the form of the system FaceID-Cloud.

This text will focus on exposing a possible feasible solution for the problem of implementing a facial identification system on top of a cloud infrastructure, describing the architecture and workflow of the planned system, and presenting the challenges found along with our proposed way of addressing them. We will start by clearly specifying the problem and our goals in the Objectives Section (see 2), followed by a survey on the previous works and state-of-the-art technologies related with this project in the Related Work Section (see 3). Next, we describe the architecture of our solution in the Proposed Solution Section (see 4) and the planned methodology to evaluate the system in the Evaluation Methodology Section (see 5). We conclude with a wrap-up of the work along with some final remarks in the Conclusions Section (see 6).

2 Objectives

The purpose of this work is, in its essence, to solve the problem of integrating an existing facial recognition technique with the technologies that are part of the Cloud Computing paradigm and to greatly increase the performance of one such technique when compared with a centralized approach. The resulting system should be able to efficiently process gigabytes of video material using the resources provided by a cloud infrastructure, and identify the people that appear in it. All processed information should be adequately stored and indexed so that useful queries can be made, namely to know which people appear in a video and which videos does a person take part in (other types of queries can be added, but these are the minimum expected from the system). The system should be able to scale horizontally computation- and storage-wise and increase its performance in proportion to the addition of new nodes. Also, it should strive for efficiency in terms of communication steps and message size as to reduce this kind of overheads.

The system should be designed with a private datacenter in mind, making use of the available resources through some kind of cloud technology. As a secondary goal, a possible integration with commercial public cloud services like Amazon EC2[1] should be sought. Also, in order for the system to be used in a real world context, it should provide an interface for anonymous users to make queries about videos on other websites (e.g. YouTube), delivering a list with the people found and possibly some visual and textual information to be integrated in the video itself (e.g. subtitles, captions).

It is not in the scope of this work to improve or develop a new face identification method. Any modifications or improvements should only facilitate the implementation of the system and should not be made on the core of the method, as it is not the focus of this work. Thus, one of the most basic face identification methods was chosen to be used, *eigenfaces* by Turk and Pentland [40], so that the focus is on the middleware design. While not in the main objectives list, as a secondary goal it would be interesting to design the system so that it can accept different face identification methods in a seamless way, if possible.

3 Related Work

In this section a survey on the background subjects of this work is made, providing the theoretical and technological foundations necessary to develop a solution for the problem. Since this work depends on knowledge from very different areas, a division was made into three distinct subjects that will be studied: Cloud Computing, Data-Driven Scheduling and Face Identification.

3.1 Cloud Computing

During the past few years, the IT world witnessed the birth and growth of a new paradigm, commonly called **Cloud Computing**, although it has also been named as Dynamic Computing [32]. It is difficult to assign a precise date for its genesis, since the term "cloud" has already been used in several contexts, describing large ATM networks in the 1990s for instance [49], and is also based upon some already existing technologies like distributed computing, virtualization or utility computing which have been around for several years [41]. However, some [16, 43] claim that the true birth of Cloud Computing happened when IBM and Google announced a partnership in this domain [19], leading to a hype around the subject and lots of popularity.

In the following subsections, a possible definition for Cloud Computing will be presented after its evolution and main characteristics (Section 3.1.1), followed by a description of some concepts related to the cloud paradigm (Sections 3.1.2 and 3.1.3) and the most significant enabling technologies (Section 3.1.4). Finally, some running research challenges (Section 3.1.5) are presented. For completeness, a survey and classification of some existent cloud systems is also given on Tables 3*a* and 3*b* in Appendix A.

3.1.1 Overview of Cloud Computing

Cloud Computing is a much younger paradigm when compared with its older brothers who already reached a matured state, like Grid Computing or Cluster Computing, meaning that it still needs a

[1] http://aws.amazon.com/ec2

standardized and generally accepted definition. In an online article [14], twenty-one IT experts define Cloud Computing according to their vision and experience in the area, but the answers are somewhat disjoint and seem to focus on just certain aspects of the technology, lacking a global analysis of the proposals [41]. Some of the experts and other authors focus on *immediate scalability*, *elasticity* and *dynamic resource provisioning* as the key characteristics for the Cloud (Markus Klemns in [14] and others in [39, 32]), while others disagree with this vision focusing on the type of service as the main concept (Brian de Haaf in [14] and others in [39]). Some other existing perspectives are based on the business model, characterizing Cloud Computing as a *pay-as-you-go based service* and with the potential to reduce costs by the realization of *utility computing* (Jeff Kaplan in [14] and others in [5, 7, 44]). Still, there are simpler views about the cloud claiming that it is merely a metaphor for the Internet, relying on its emerging set of technologies and services to satisfy computing needs of users and organizations [18].

3.1.1.1 From Mainframes to Cloud Computing

Even though Cloud Computing is considered a new paradigm, its roots come from far back in the past. In the 1960's, large mainframes could be accessed by several thin clients to process bulk data, very much like today users can use personal computers, tablets, smartphones and other computing devices to request a service from a cloud provider. Thus, one could say that the hype around the cloud does not have a great reason to exist, as it seems to have just brought back an old and well known concept. Steve Mills, Senior Vice-President and Group Executive at IBM, illustrated this point in a recent interview[2] stating that "We [IBM] have been running multitenancy for decades and decades" and that "Everything goes back to the beginning, the mainframe", when asked about his company's view on Cloud Computing. Although it is true that mainframes and the present cloud systems share some basic characteristics, there are some clear distinctions to be made in respect to computing power, storage capacity and scalability. According to [42], a mainframe offers finite computing power, being a physical machine, while cloud systems offer potentially limitless scalability regarding power and capacity. Also, as opposed to the simple terminals used to access mainframes, today's computers have significant computing power and storage capacity on their own, allowing for a certain degree of local computing and caching support.

In this perspective, some authors describe the technology evolution steps from the mainframe paradigm to the cloud one [42, 32], although the descriptions do not match exactly. It started, of course, with mainframes shared by several users and accessed through terminals. However, this strategy is not financially feasible for a single person, leading to the birth of the personal computer era, defined as the second stage in the evolution by the authors. Nonetheless, there were still considerable costs for companies, since each computer needed its own application interfaces and databases. With the appearance of local networks, the third stage, the client-server model helped organizations to reduce costs by featuring a centralized database access in the servers and the application interfaces in the clients. Yet, this model still has limited resources and can not be applied globally in an efficient and effective way. The fourth stage saw the advent of local networks that could connect to other local networks, giving birth to the World Wide Web and the Internet, providing no single point of failure, no single point of information, no single owner and no single user or service provider. All this stages laid the foundation for the appearance of several paradigms based on distributed information systems, including, of course, Cloud Computing.

3.1.1.2 Economy of Scale

An important concept that helps define Cloud Computing is that of *Economy of Scale* [43, 49]. The rationale behind large data-centers is based on the notion of reductions to the unitary cost as the size of the facility increases, allowing companies to focus on specializing and optimizing their products. Although it is a term spanning several business areas, in respect to the context of this text an economy of scale is usually achieved by building a data-center with a very high number of commodity-class systems, leveraging the lower price and also the lower maintenance costs of this type of hardware. In this way, network equipment, cooling systems, physical space and other resources can be efficiently

[2] http://news.cnet.com/8301-13953_3-9933108-80.html

exploited, where otherwise with fewer high-end systems resource utilization efficiency would severely drop. Since the cost and computing power of a single system is so insignificant when compared to the whole data-center, providers can even cope with several failures without bothering to replace individual nodes. Also, to expand the data-center it suffices to acquire more commodity computers and add them to the existing bundle. By reducing their costs with data-centers, cloud providers can lower the prices for their clients, turning Cloud Computing into a competitive and feasible business.

3.1.1.3 Utility Computing

During the past century, society has become accustomed to utility services such as electric power, water, natural gas, telephone access and many others, including Internet access more recently. The infrastructure supporting these services has evolved to an utility based business model, where customers pay solely for what they use and do not have to own the whole necessary equipment, what would be financially infeasible. This way of thinking is now embedded in our mentality, where it does not make sense to possess an expensive power generator at home, for instance, when there are companies who can supply the same service at a lower cost based on economies of scale. All these services are shaped by a combination of several requirements like ease of use, pay-as-you-go charges, high reliability among others [33].

Given the concept of an utility service described above, the Utility Computing paradigm can be described as the on-demand delivery of infrastructure, applications and business processes in a security-rich, shared, scalable and standards-based computer environment over the Internet. Customers can access IT resources as easily as they get their electricity or water, and are charged in a pay-as-you-go basis [33].

3.1.1.4 Service-Oriented Architecture (SOA)

In the IT business world, SOAs are gaining ever more importance as a universal model to which automation and business logic conforms. Service-orientation aims to cleanly partition and consistently represent available resources, simplifying technical disparities by applying abstraction layers and providing them as services, which in turn lay the foundation for standards for representing logic and information. The main principle is that one should leverage the potential of individual services existing autonomously yet not completely isolated from each other, seeing that they must conform to a set of principles that allow them to evolve independently, while assuring some commonality and standardization. Thus, SOA is a form of technology that adheres to the principles of service-orientation, with the potential to support and promote these principles throughout the business process and automation domains of an enterprise when realized through a Web services platform [12]. Cloud Computing is closely related to SOA in that it aims to supply several different services, providing the necessary technologies and flexible platform for companies to build their SOA solutions in a cost-effective way.

3.1.1.5 Defining Cloud Computing

Before presenting a possible definition for the cloud paradigm, it is important to describe the distinctive technical characteristics which differentiate it from earlier related paradigms. Note that there can be dependencies between the listed characteristics, in the sense that one is achieved by relying on the existence of the other, but it is relevant to make the distinction as they are significant in the classification of cloud systems.

On-demand service provisioning. One of the key features of Cloud Computing is the possibility to obtain and release computing resources on the fly, following the real necessities of an application. This avoids both having to pay for unused resources and the exact opposite, missing resources during periods of peak demand.

Service orientation. As mentioned before in Section 3.1.1.4, Cloud Computing aims to provide different services and act as a foundation for SOA solutions. The business models around the cloud are service-driven, hence a strong emphasis is placed on service management. A description of the most significant cloud service models is given in Section 3.1.2. In order to protect both cloud providers

and customers, a Service Level Agreement (SLA) is negotiated, where the provided services and conditions are defined. SLAs oblige providers to live up to the Quality of Service (QoS) terms they committed to in the SLA and assures customers that the services they paid for will be fully provided. On the other hand, SLA can also protect providers from having to cope with unreasonable requests. Yet, SLA's still constitute one of the hardest challenges to overcome, as cloud providers cannot usually guarantee an agreed QoS, forcing some companies to refrain from migrating their services to the cloud. This problem will be discussed in the Challenges Section (see 3.1.5).

Scalability and Flexibility. A scalable and flexible infrastructure is central to cope with different geographical locations of the resources, disparate hardware performance and software configurations. Also, to support a dynamic service provisioning, the computing platforms should be flexible to adapt to the requirements of a potentially large number of users without violating the QoS conditions of the SLAs [44].

Pay-per-use utility model. Some of the experts in [14] referred a strong connection between Cloud and Utility Computing, in the sense that the former is the realization of the earlier ideals of the latter. An utility service model is usually applied to Cloud Computing featuring a pay-per-use billing, where customers only pay for what they really use or for small predefined quanta (e.g. 1 hour of CPU, 1GB of storage, etc).

Ease of access. In order to comply to the utility computing model present in Cloud Computing, the services offered must be ubiquitously accessible from any connected devices over the Internet, with simple to use and well defined interfaces and with very few requirements from the client system [44].

Virtualization. Virtualization techniques provide the means to achieve most of the above mentioned features, namely by partitioning hardware and thus acting as the base for flexible and scalable computing platforms [44]. Virtualization is, thus, one of the major underlying and enabling aspects of the Cloud Paradigm. This subject will be described in more detail in Section 3.1.4.

Multi-tenancy. Cloud infrastructures must be capable of serving multiple clients, or tenants, under the same hardware or software infrastructure to achieve the goal of cost effectiveness. There are always some issues in need to be addressed in order to achieve the full potential of multi-tenancy, like security isolation, customizing tenant-specific features or efficient resource sharing, which are usually handled by leveraging the virtualization support of the cloud infrastructure.

Given the set of concepts described, a possible definition for the cloud paradigm can now be given, based on the work of [41]: ***Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development tools and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale) and can be accessed by several users simultaneously (multi-tenant), allowing thus for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by cloud providers by means of customized SLAs***. Note that the Cloud concept is still changing and this definition shows how it is conceived today.

3.1.2 Cloud Service Models

Cloud Computing has been widely described has a service-oriented paradigm by several authors [41, 39, 32, 5, 31, 16] and the IT community in general. It is clearly one of the major characteristics that distinguish it from similar paradigms, like Grid Computing. Although the SOA (Software Oriented Architecture) concept is common to both Cloud Computing and Grid Computing, it reaches a more practical state in the former [16].

There are two key properties that enable Cloud Computing to be service-oriented in agreement with the utility philosophy, *abstraction* [41, 16, 43] and *accessibility* [43]. In order to achieve the type of interaction described earlier with utility computing, it is necessary to hide all the complexity of the cloud architecture from the end-user and to present an user-friendly interface for requesting services. In respect to the first requirement, the abstraction property is best described through a layered

diagram which identifies the several parts of the cloud architecture and the relations between them. Each layer is, of course, built in a hierarchical way on top of one another and represents a specific type of service to be provided based on the service provided by the subjacent layer.

Note that each layer can in fact offer a service for end-users even if it is not the top layer, distinct from the common software engineering notion of a system build from several modular layers, where often the top one alone provides some service. Virtualization, described in Section 3.1.4, is a key requisite to attain abstraction without exposing the underlying architecture details to the users. As for the accessibility requirement, the common practice is to provide a standardized API which respects a given protocol (e.g. SOAP, REST) and enables an easy access to the service from anywhere in the world. This easy access to cloud services is likely to be one of the main boosters of its ongoing acceptance in the non-academic community in a relatively short period of time [46].

A description of the several cloud service models is now presented, starting with an hierarchical view for Cloud Computing on Fig. 1 followed by the description of the service model associated with each layer in the hierarchy.

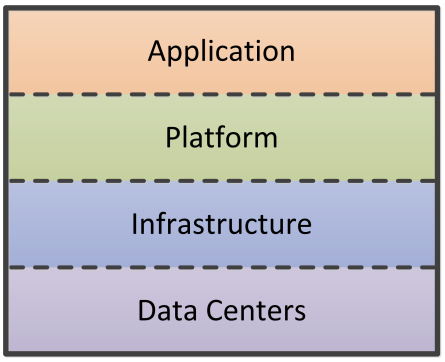

Fig. 1: Cloud Hierarchical Layer View

Data Centers. The Data Centers layer is the foundation for Cloud Computing technologies, providing raw computation, storage and network hardware resources. Usually, they are built in less populated ares with cheaper energy rates and low probability of natural disasters, being constituted by thousands of inter-connected servers [39].

Infrastructure-as-a-Service (IaaS). The Infrastructure layer focuses on leveraging the subjacent data-center's resources and provisioning them as services to consumers by virtualizing storage capacity, computational power, communications and other fundamental computational resources necessary to run software. Companies offering this kind of services are commonly called *Infrastructure Providers* (IPs) [41]. Even though users do not directly control the physical infrastructure, they are able to scale their virtualized resources up and down dynamically according to their needs. This is in line with the utility perspective, where consumers use and pay only for what they do consume. Also, the virtualization of resources is of great value for IPs, since they can focus on maximizing the efficiency of their infrastructure and therefore make it more profitable. An example of an commercial solution in this domain is Amazon's Elastic Compute Cloud (EC2).

Inside the IaaS domain it is still possible to define some sub-categories, although there is less agreement from the IT community in these and the naming is primarily performed by business marketing people trying to differentiate their product's characteristics. First, note that the term *storage* above does not clearly identify the type of service. Is the data organized in some way as in database or is it just stored in bulk as a backup? From this perspective *Database-as-a-service* (DBaaS) and *Storage-as-a-Service* (StaaS) emerge. The former is centered on providing a fine-grained access to data by delivering database functionality as a service, supporting multi-tenancy, automated resource management and other features of traditional database service systems [29]. The latter refers simply to cloud storage leveraging virtualized storage resources, with no complex data querying functionalities provided as in DBaaS. Another sub-category of IaaS derives from the virtualization of communication services, being called *Communications-as-a-Service* (CaaS). In this field, communications platforms like VoIP are offered as a service, freeing companies from the financial burden of building a platform from scratch.

Platform-as-a-Service (PaaS). The second layer, Platform layer, adds an additional abstraction level to the services supplied by the Infrastructure Layer, effectively aggregating them and offering a complete software platform to assist in application design, development, testing, deployment, monitoring. Namely, it provides programming models and APIs for cloud applications, MapReduce [11]

for example. Also, problems like the sizing of the resources demanded by the execution of the services are made transparent to developers [41].

Software-as-a-Service (SaaS). Finally, the Application layer presents software to end-users as an on-demand service, usually in a browser, being a model already embraced by nearly all e-business companies [31]. It is an alternative to locally running applications with some clear advantages over them. First, applications hosted in the cloud do not require users to install and continuously update applications on their own computers, saving time and disk space, and enhancing accessibility and user-friendliness. This also helps software providers, since updates are automatically distributed to users, needing only to alter the few instances running in the cloud infrastructure. Also, the software developers can target their applications to work in a known and controlled environment, raising the quality and security of the products.

3.1.3 Cloud Deployment Models

Another way of classifying a Cloud infrastructure is according to its type of deployment. Currently, there are three main deployment models acknowledged by the IT community, *public*, *private* and *hybrid* clouds [49, 32, 5, 34], although some other models are also considered such as *community* [5, 34] and *virtual private* [48, 49] clouds.

Public or Hosted Clouds. In this deployment model, the cloud infrastructure is owned by organizations selling cloud services who offer their resources to the general public or large industry groups (e.g. Amazon, Google, Microsoft). Public clouds offer several benefits to service consumers since there is no initial investment on infrastructure and the related inherent risks are shifted to infrastructure providers. However, there are some drawbacks, like the lack of fine-grained control over data, network and security settings, hindering the effectiveness in some business scenarios.

Private Clouds. Although the cloud ecosystem has evolved around public clouds, organizations are showing interest in open source cloud computing tools which let them build private clouds using their own or leased infrastructures. Thus, private cloud deployments' primary goal is not to provide services over the Internet, but to give users from the organization a flexible and agile private infrastructure to run service workloads within their administrative domains. This model offers the highest degree of control over the performance, reliability and security of the resources, and improves resource reuse and efficiency, but the advantages of a public cloud are lost, since the organization itself has to perform maintenance and entails up-front costs in case of not owning the infrastructure. It embodies, to some extent, the late adoption of Grid Computing principles by corporate IT.

Hybrid Clouds. Both public and private cloud approaches still have some limitations, namely the lack of control and security in the former and the difficulty of expanding or contracting the resources on-demand in the latter. A common solution is an hybrid deployment model, where unique clouds are merged, bound by technology that enables interoperability amongst them, taking the best features of each deployment model. In an hybrid cloud, an organization builds a private cloud over its own infrastructure, maintaining the advantage of controlling aspects related to performance and security, but with the possibility of exploiting the additional resources from public clouds on an as-needed basis (e.g. when there is a demand peak on the infrastructure of an online retailer in holiday season). On the downside, designing a hybrid cloud requires a careful decision on the best split between public and private cloud components and the cost of enabling interoperability between the two models.

Community Clouds. Community clouds are somewhat of a middle way between public and private clouds. They are not owned by a single organization as in a private cloud, but are also not completely open to the general public as commercial services. The cloud infrastructure is usually provisioned for exclusive use by a specific and limited community of organizations with shared concerns or interests, partially inspired by P2P grid efforts in the past.

Virtual Private Clouds. Another way of dealing with public and private clouds limitations is called Virtual Private Cloud (VPC). In this deployment model, a platform is built on top of a public cloud infrastructure in order to emulate a private cloud, leveraging Virtual Private Network (VPN)

technology. In this way, users are allowed to design their own topology and security settings, having a higher level of control than allowed by simple public clouds, while retaining easy resource scalability. VPC is essentially a more holistic design since it not only virtualizes servers and applications, but also the underlying communication network as well, facilitating the task of migrating services from proprietary infrastructures to the cloud.

3.1.4 Enabling Technologies

There is a set of well understood technologies which constitute the foundation for cloud infrastructures. Without enabling technologies like *virtualization*, *web services*, *distributed storage systems*, *programming models*, just to mention a few, most of the cloud paradigm features would not be attainable. A brief description of these subjects will be not presented.

Virtualization Technology. Virtualization is one of the key components that render cloud computing service models possible. Each layer in the architecture can be provided on-demand as a service through virtualized resources, which not only provide users with access to controlled environment and performance isolation, but allow infrastructure providers to achieve flexibility and scalability of their hardware resources. Also, virtual machines act as a sandbox environment, enforcing a high security level and isolation between instances. In the context of Cloud Computing, virtualization is achieved through special purpose applications called *Virtual Machine Monitors* (VMM), or *hypervisors*, which simulate an hardware environment for deploying guest operating systems, while controlling every aspect regarding resources and interactions. Among the most widely used VMMs are the Xen Hypervisor[3], KVM[4] and VMware ESX[5].

Web Services. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards[6]. The generalized acceptance of Web services in the e-commerce business, namely in the service providing industry, has greatly contributed for the growth and recognition of the Cloud Computing paradigm. Cloud services are usually exposed as Web services, being easily accessible from anywhere in the world and leveraging the standard mechanisms provided to interact with different types of clients. An example of this technology is Amazon Web Services[7].

Distributed Storage System. The underlying distributed storage system where the various cloud layers are built upon is, of course, another important technology necessary for Cloud Computing. Built on top of large distributed data centers, it provides scalability and flexibility to cloud platforms, allowing data to be migrated, merged and managed transparently to end-users for whatever data formats [45]. An example of this technology is the Google File System [15] and Amazon S3[8]. A cloud storage model should not only focus on how to store and manipulate large amounts of data, but also on how to access it in a semantic way. Since the goal of a distributed storage system is to abstract the actual physical location of data, this access should be performed by logical name and not by the physical name. For example, in Amazon S3 each object is assigned a key and is accessed through a meaningful URI, which does not identify a physical location.

Programming Model. In order to comply with the ease of use requirement from the utility computing vision, some Cloud programming models should be proposed for users to adapt more easily to the paradigm, as long as they are not too complex or too innovative. One of the most known programming models is MapReduce [11], which allows the processing and generation of massive data sets across large data centers.

[3] http://xen.org/
[4] http://www.linux-kvm.org/
[5] http://www.vmware.com/
[6] http://www.w3.org/TR/ws-arch/
[7] http://aws.amazon.com/
[8] http://aws.amazon.com/s3/

3.1.5 Challenges

Even though there is a big hype around Cloud Computing, there are still several challenges that need to be overcome. These problems do not prevent the application of cloud technologies – they are being used today – but introduce difficulties when trying to take full advantage of its characteristics.

Cloud Provider Interoperability. Most cloud providers spend time and money developing an API for their cloud infrastructure, giving users a simplified and documented way to access cloud services. However, there is no consensus on a standardized API between providers, as each one has already invested many resources in their solution. This situation forces companies to invest on and develop new applications for one specific cloud platform, as it is not financially feasible to cover the whole spectrum of existing platforms. Customer lock-in may be attractive to cloud providers, but users are vulnerable to price increases, to reliability problems or even to providers going out of business [1].

Coupling Between Components. It is common practice for cloud providers to possess the whole suite of platforms necessary for clients to build full applications. Yet, this usually means that there are dependencies between these modules, where if one is to be used, it needs another one on which it depends to correctly function, bearing a high-coupling. Thus, one is stuck with a full suite of computing services from the same provider with a high cost to change, exactly as the APIs.

SLA Support. Service Level Agreements are the major obstacle for the wide adoption of cloud computing today. Cloud providers are still struggling to achieve the level of guarantees needed for companies to use cloud infrastructures for serious business deployment. Until providers are capable of signing the SLAs and comply with their requirements, most large organizations will not rely on the cloud for mission-critical computing needs since it cannot provide the necessary customization and service guarantees. Also, the business is very dynamic, while SLAs are quite static, meaning that they are not able to adapt to changes in business needs [39].

Security. Most organizations will not have their sensitive corporate data on the cloud, whether because a public cloud system is exposed to more attacks, or some nations' laws prevent customer and copyrighted data outside national boundaries, or even the concern that a country's government can get access to their data via the court system. These are all well justified reasons, but there are no fundamental obstacles on making a cloud environment as secure as a private data center. Technologies such as encrypted storage, VPC and firewalls are capable of providing the necessary security protections needed by the customer organizations.

3.1.6 Cloud Computing in the Proposed System Context

This area has, of course, a major influence in the system to be designed, as it will be implemented on top of a cloud infrastructure. This means that all the advantages of the cloud paradigm will be made available, such as elastic resource provisioning and scalability for example, but also some of the problems. The system is envisioned to be built on top of an IaaS-oriented private cloud infrastructure, but nonetheless able to expand into a hybrid infrastructure if needed or even a full public cloud approach.

3.2 Data-Driven Scheduling

Nowadays, web and scientific applications need to deal with massive amounts of data. Social networks, video-sharing websites, search engines, physics simulators, they all have to handle several petabytes of data per day and the tendency is for this number to rise. Given such scenarios, data placements jobs and decisions are gaining more and more importance when designing a robust workflow planner, where computational jobs have usually been the most used methodology. Workflow planners need to become data-aware in order to tackle this emergent growth of data-intensive applications, taking into account problems like data placement, storage constraints or efficient resource utilization [22, 21].

3.2.1 The Problem

CPU-centric schedulers have been the preferred method of running applications, namely in clusters/grids, as the focus of tasks is usually to leverage all the existing computing power to achieve some result in the shortest time possible, the CPU-time and network access being the dominant bottlenecks. However, the growth and spread of data-intensive applications is outpacing the corresponding increase in the ability of computational systems to transport and process data. Also, techniques which once worked well for CPU-intensive workloads in a local environment can suffer orders of magnitude losses in throughput when applied to data-intensive workloads in remote environments. This situation has been forcing a new perspective, where data access is viewed as the main bottleneck, turning the scheduling of data placement activities in a crucial problem to be solved and carefully coordinated with CPU allocation activities [4, 34].

3.2.2 Early Work on Data-Driven Systems

Even though huge datasets were not a reality outside scientific projects until recently, there were already some concerns about the role of data in computational tasks a few decades ago. In [38], data-flow (data-driven) was perceived as a possible fifth-generation architecture for computer systems, opposing the von Neumann principles which are based on control-flow (CPU driven). This work was focused on the availability and flow of data inside a single machine, where operands would trigger the operation to be performed on them when all the required inputs are available for that operation. In a data-flow architecture each instruction can be seen as having a computing element allocated to it continuously waiting for the arguments to arrive, lying dormant whilst they do not.

Thus, *data-driven* is defined as the term to denote computation organizations where instructions passively wait for some combination of their arguments to become available. This strategy has the advantage that an instruction is executed as soon as its operands are available, leveraging a high degree of implicit paralelism. On the other hand, it can be a restrictive and time wasting model, as an instruction will not execute until it has all its arguments. The introduction of some non-data-driven instructions is possible to overcome this problem and provide some degree of explicit control.

3.2.3 Computational Jobs vs. Data Placement Jobs

There is no efficient solution based exclusively on computational job scheduling or data placement job scheduling when developing a data-intensive application, so it is necessary to bring them together in order to take full advantage of their best features. According to [22], in the world of distributed and parallel computing data placement activities are regarded as second class citizens, meaning that they cannot be queued, scheduled, monitored, managed and check-pointed, just as a computational activity. However, there has to be a different treatment for data placement jobs, as they have very different semantics and characteristics compared to computational jobs. For example, if a large file transfer fails, a full retransmission is not desirable. It should be possible to restart the transfer from the point where it failed or to try again using another protocol, for example, and a traditional computational job scheduler may not be able to deal with such cases. Thus, there should be different schedulers for each type of job, leveraging the different semantics of each one.

3.2.4 Definition

It is possible to establish a definition using the notion of data-driven architecture stated in Section 3.2.2 and the properties described in Section 3.2.3, adapted to the problems of today's data-intensive distributed applications. The principles are essentially the same, only the computing environment and the data dimension are different. In this perspective, ***a data-driven distributed system is an application where the availability of data drives the execution of the individual tasks, so that data placement activities are considered first-class citizens and thus can be scheduled, queued, monitored, managed and check-pointed as if they were computational jobs.*** This means that instead of having computational jobs triggering data requests from a remote location across a throughput-limited network, it is possible to schedule data placement activities which will trigger the tasks needed to be performed on that data.

3.2.5 Data Aware Batch Scheduling

In data-intensive distributed systems, it is often necessary to start a high number of similar tasks, usually called a workload, which will process high volumes of data autonomously. By *similar* it should be understood as capable of running in the same system without manual intervention (e.g. same input/output formats). This process is commonly denoted by *batch scheduling*, being, for example, extensively used in scientific applications where problems are usually highly parallelizable, although a good scheduling must be performed in order to efficiently make use of the available resources. As stated in previous sections, scheduling is widely performed with computation jobs as the main driver, but the constant growth of data is becoming a relevant problem, which means that batch schedulers should take data into consideration when planning. In this section, some relevant topics concerning the properties a data-aware batch scheduler should possess will be presented.

3.2.5.1 Efficient Resource Utilization

One of the main concerns of a data aware batch scheduler is to optimize the utilization of the available resources, making sure that neither the storage space nor the network connections get overloaded. In practice, this is strongly connected with the way parallelism and concurrency are used to maximize the overall throughput of the system, as showed by [21] (parallelism refers to the transfer of a single file using multiple streams while concurrency refers to the transfer of multiple files concurrently). The empirical results of this work showed that the effect of both methods in the efficiency of the system is different if it is tested on a wide area network (WAN) or on a local area network (LAN). In a WAN, the transfer rate of the system increases as expected, but in a LAN it comes to a threshold and additional streams and transfers causes it to decrease. Also, it was observed that CPU utilization at the clients increases with both parallelism and concurrency, while on the server only the concurrency level increases the CPU utilization. With an increased parallelism level, the server CPU utilization starts dropping and keeps this behaviour as long as the parallelism level is increased. It is interesting to note that concurrency and parallelism have completely different effects on the CPU utilization at the server side, probably due to the amortization of the select() system call overhead when monitoring several streams for the same file. Thus, it is important for a scheduler to conciliate both parallelism and concurrency, as the usage of each strategy by itself is not very effective, and is also important to adjust them dynamically, as it helps maximizing the transfer rates and lowers the CPU utilization on the servers.

3.2.5.2 Storage Space Management

When scheduling data transfers to a host, it is important that the necessary space is available at the destination. In an ideal scenario, the destination storage system is capable of space allocations prior to the transfer itself, therefore assuring the scheduler that it can proceed with the transfer. However, this feature is not supported by some storage systems, so the scheduler has to keep track of the size of the data entering or exiting each host. Also, in a storage constrained environment it should apply some strategy that maximizes the most desirable quality of the data (e.g. size, age or some user or system wide utility measure) and takes into account all the jobs executing in the same host and their input and output sizes.

Assuming multiple uniform pipelined jobs to be executed in one host, four possible strategies are described in [4] in order to maximize the throughput of the workload (i.e. to minimize the total time to completion). In these strategies, a set of pipelined jobs is called a *batch*, whereas the read-shared data between jobs is called *batch data*. Also, each pipeline has access to its *private data* that is not shared with other pipelines. Finally, each storage unit allocated to hold a particular data is called a *volume*. A description of the four strategies is now presented, starting with the one with less storage limitations to the one with the most constrained environment:

All Strategy. In this strategy, there are no storage constraints so all the space needed by the workload fits in the available storage. Thus, the planning is straightforward as no possible schedule can result in adverse effects. Also, since the storage space for all jobs executing in a given host can be readily allocated, there are no limits on the concurrency level between the jobs and no refetching of data is needed.

AllBatch Strategy. In the case where all batch data fits in the host storage, but not all private volumes can be allocated, an AllBatch Strategy can be applied, as long as there is at least one pipeline that can simultaneously allocate two volumes for its input and output. This strategy limits the system concurrency and may cause computing capacity underutilization if the number of pipelines that can allocate their private volumes is fewer than the number of available CPUs at the host.

Slice Strategy. When the system is even more storage-constrained compared to the previous two strategies, a Slice Strategy can be used whenever an horizontal slice of the workload can be allocated simultaneously. A slice requires storage space for at least one batch volume (which is used at the same level by all pipelines in the host) and one private volume for each pipeline plus an additional private volume so that at least one of the jobs can access both its input and output storage and allow the workload to make some progress. This strategy ensures that no batch refetch is necessary, since the entire horizontal slice will execute before the workload continues to the next level.

Minimal Strategy. The most constrained approach is the Minimal Strategy, where at least one job needs to have have access to its batch volume and its two private volumes (input and output). This strategy suffers from several problems, like the need to refetch batch volumes and the underutilization of CPU since only one job is executing. It is a similar approach to the AllBatch Strategy, with the difference that only one batch volume can be allocated and refetching needs to be performed.

3.2.5.3 Data Placement

In a data-intensive distributed system, a workflow planner must either take into consideration where data is located and send jobs to a site"close" to it, or it should include stage-in and stage-out steps in the workflow, in order to make sure that data arrives at the execution site before the computation starts. Both strategies are not mutually exclusive, of course.

There are some similarities between data-aware workflow planning and the way microprocessors plan the execution of instructions, where pipelining strategies are employed. In the latter, data retrieval from memory is usually the bottleneck due to slow memory access speed and the latency in the bus and CPU. When applying a pipeline strategy at the instruction level, memory access instructions are buffered and ordered in order to improve the overall throughput. The same strategy can be applied to a distributed workflow planner, where workloads can be viewed as a large pipeline and the bottleneck is the access to remote data due to network latency, bandwidth and communication overhead. Pipelining techniques are used to overlap different types of jobs and to execute them concurrently, yet maintaining the task sequence dependencies, which is what a workflow planner needs to achieve. By following the pipeline strategy, it can order and schedule data placement tasks in a distributed system independently of the computation tasks in order to exploit a greater level of parallelism and reduce CPU underutilization when waiting for data to arrive, while enforcing data dependencies.

3.2.6 Data-Driven Scheduling in the Proposed System Context

The target system of this work will have to deal with large amounts of data, potentially processing gigabytes of video material in a small time interval. This amount of data can hurt the performance of the system if the focus is on computational-oriented tasks and not on data-oriented tasks. Thus, it is important to define data tasks as the basic work unit and guide the computation according to the placement and availability of the data.

3.3 Face Identification

The ability to recognize and distinguish one face from thousands of other different faces has been imprinted on the human brain visual cortex for millennia, so that now people perform this incredible task subconsciously without realizing the difficult process around it, and what it takes for their brain to accomplish an identification with a very high precision. Nonetheless, however easy it seems for a human to recognize a face quite independently of the context, computers still struggle for high confidence intervals when presented with the same face in several different environments. During the last 40 years, the problem of face identification by computers has been the subject of extensive research,

with several techniques being suggested, but humans continue undisputed in this area. During this last decade, the great improvements in computing power have brought new possibilities, enabling techniques that 20 or 30 years ago were simply not feasible, CPU and memory-wise.

In this section, the problem of face identification by computers will be described, starting with an overview and definition of the problem and goals (Section 3.3.1), followed by a description of how humans identify faces (Section 3.3.2). Next, the description of some existing methods to perform recognition (Section 3.3.3). Finally, some issues and approaches specifically related to identifying faces in videos are described (Section 3.3.4).

3.3.1 Defining Face Identification

Solely from the name of the subject, a reader with no background in computer science can quickly understand what it means to identify a face, as it is an action performed naturally everyday. Nonetheless, it is necessary to define clearly what facial identification means for a computer. It is part of a broader subject, which is known as *human face perception* in [17]. A general notion of this area is given, stating that it is a kind of intelligent behaviour for computer to catch and deal with the information on human faces, including human face detection, human face tracking, human face pose detection, human face recognition, facial expression recognition, lipreading and others. Although face identification is a specific sub-area of human face perception, it partially encompasses several of the others areas presented.

Simply put, it tries to solve the problem of, given a still or video image, detect and classify an unknown number (if any) of faces using a database of known faces. Based on this problem, ***face identification could be defined as the detection, tracking and recognition of human faces by a computer system, trying to equal, and eventually surpass, human performance***. It is worth noting that although it is relatively simple to clearly identify the problem and what has to be done to solve it, in [40] it is shown that developing a computational model for face recognition that efficiently and completely solves the problem is immensely difficult, as faces are a complex, multidimensional, and meaningful visual stimuli. Also, faces are not easily mappable, as they are a natural class objects, so the usual models utilized in other computer vision research areas cannot be applied.

3.3.2 Face Identification by Humans

As stated before, one of the goals of face identification systems is to reach and eventually surpass human performance on that same task. Yet, computers are still not at the same performance level as the human visual system when identifying faces even in the midst of very difficult and unconstrained contexts, specially different viewpoints and lighting conditions. Given this, it makes sense to study the way the human visual system performs the identification of faces.

Humans make use of a lot of sensory information to perform recognition (visual, auditory, olfactory, tactile, etc) and in some cases can even rely on the context to identify a person. Yet, most of this information is not easily available to computers, which are frequently only presented with 2D-images. However, computers have a clear advantage on the amount of information that can be stored and processed, even if they cannot use it so effectively. A person is typically able to store, or "remember", a limited set of faces in the order of thousands, while a computer system can potentially store an infinite number.

Some relevant topics regarding the way humans recognize faces are now presented, based on the works of [8, 50, 35].

3.3.2.1 Feature-based vs. Holistic Analysis

Humans are believed to rely heavily on two crucial methods when analyzing a face, *feature-based* and *holistic-based* analysis. These approaches are by no means mutually exclusive, so they are used together when identifying a face. Nevertheless, the question remains as to which of this approaches most influences a recognition.

A feature based approach is performed in a bottom-up fashion, where individual features (e.g. eyes, eyebrows, mouth, etc) are extracted from a face and assigned to a known person which has similar features. On the other hand, an holistic, or configural, approach implies the apprehension and classification of the face as a whole, using elements such as the shape or the global geometric relation between individual features to help in the recognition. Studies suggest the possibility that a global description like this can serve as a front end for a feature-based analysis, being at least as important as the latter. Yet, in the case that dominant features are present, such as big ears or a crooked nose, holistic descriptions may be skipped [8].

A simple example of the importance of holistic analysis is when features on the top half of a face are combined with the bottom half of another face. What happens is that the two distinct identities are very difficult to recognize as whole. However, when the two halves are separated, presumably disrupting the holistic process, the individual identities of each half are easily recognizable. This also suggests that some features alone are sometimes sufficient for facial recognition [35].

Another strong evidence of the importance of a holistic analysis is that an inverted face is much harder to recognize than an normal face. In [2] this is exemplified using the "Thatcher Illusion", where the eyes and mouth of an expressing face are inverted. The result in the face with the normal orientation looks grotesque, but on the inverted version it appears quite normal and it takes some close inspection to notice the change.

Although feature processing is important for facial recognition, given these results it is suggested that facial recognition by the human visual system is dependent on holistic processes involving an interdependency between features and configural information [35].

3.3.2.2 Importance of Facial Features

Even though all features of a human face can be used for identification, there are some who appear to be more significant to this task, without which it seems to be harder to identify someone. Among these most important features are included the face outline, eyes and mouth. The nose is usually not considered with much significance in frontal images, but it has a greater impact in a profile view of a face.

Another feature whose importance is typically undervalued is the eyebrow. It is typically a mean to convey emotions and other nonverbal signals, which the human visual system may be already biased to detect and interpret, and so it may be that this bias is extended to the task of facial identification. It is also a very "stable" feature, as they are hardly occluded as the eyes, tend to contrast with the surrounding skin and are a relatively large facial feature, surviving image degradations. Some interesting results on this subject show that the upper part of the face is more easily recognized than the lower part, presumably by the presence of the eyes and eyebrows.

3.3.2.3 Face Identification as a Dedicated Brain Process

Humans are capable of identifying all kinds of objects from the world, but faces are believed to possess a dedicated process in the brain just to deal with them. Some arguments on that there is indeed a dedicated process are:

- Faces are more easily remembered in an upright orientation when compared to other objects. As stated before, an inverted face is harder to identify than a normal one, presumably by disrupting the holistic analysis process.
- People who suffer from prosopagnosia find it very difficult to recognize faces, even those they were familiar with, but typically they are able to recognize other objects without much difficulty. They can perceive the individual features, but are unable to see them together in order to identify the subject.
- Newborns are believed to be more attracted by faces and face-like patterns than other objects with no patterns or mixed features, suggesting a genetically predisposition to process faces in order to detect and identify parents or caregivers.

3.3.3 Automatic Face Recognition Methods

There has been extensive investigation on how to best recognize a face given a learning database. Researchers from the most different areas have shown interest on the face recognition problem, leading to a very vast and differentiated literature and to the creation of several methods, each having some advantage over the others while loosing in some aspect. In order to achieve a high-level classification of these kind of methods, and since the investigation on this area tends to follow the way the human visual system works, the already described categorization on how humans recognize a face will be "borrowed", namely *feature-based matching methods* and *holistic-based matching methods*. Also, as some methods employ strategies from both approaches, a third category will be used to classify these hybrid systems, named *hybrid-based methods*. This classification is based on the work of [50].

3.3.3.1 Feature-based Matching Methods

One of the proposed ways to distinguish between two faces is through the geometric relations among the several existing facial features, relying on a feature set which is very small when compared to the number of image pixels. The general idea is that by extracting and storing a set of known features from a face image in some useful representation, it is possible to compare this set with that of another image to see if they are sufficiently similar to belong to the same person. Thus, it is necessary that the chosen set of features to extract be sufficient for discrimination even at low-resolution images, so a careful study on the importance of each feature for discrimination must be done. This is typically applied in a supervised learning fashion, so that a learning database has its features extracted and any new face is considered against those extracted features. Only a general description of purely feature-based matching methods is given, whereas further sub-categorization can be found in Table 1 of Appendix A.

A big advantage that feature-based methods have over holistic-based is their ability to adapt to variations in illumination, scale and rotation up to a certain degree, which was the central argument in favor of feature-based methods in [10]. An example of this is the normalization step in [6], which achieves scale and rotation invariance by setting the interocular distance and the direction of the eye-to-eye axis, and achieves robustness against illumination gradients by dividing each pixel by the average intensity. Yet, this type of methods are more likely to have a bad performance when some features are occluded or facial hair and facial expressions are present [13].

An important step in these types of methods is the extraction of the features. Derived from the configuration properties of a face, constraints such as bilateral symmetry or the fact that every face has two eyes can be exploited in order to facilitate feature extraction. An early work on this subject was performed by Kanade [20], where an algorithm for automatic feature extraction is suggested. It starts by calculating a binary representation of the image and then runs a pipelined analysis program which will try to extract the position of individual features from the binary representation, like the top of the head or the eyes. From this analysis, a feature vector is obtained containing ratios of distances between features, which can then be used to compare between two faces.

The next step after the feature extraction is the recognition of the face, which is performed by building a selection model from the training data. It is important that the model does not adapt specifically to selecting on the training data, but that it generalizes to unseen patterns [10]. The classification of a face is performed by evaluating the distance between the feature vector from the face to be classified to all the vectors in the learning database, using a nearest neighbour approach or other classification rule (Bayes for example [6]). Different similarity matching methods can be used, but the most common is the Euclidean distance. Some examples of other methods are the mixture-distance [10], cosine-distance, Mahalanobis-distance [27] or weighted-mean absolute square distance [13].

3.3.3.2 Holistic-based Matching Methods

Another common approach to perform facial identification is to use the whole face region as an input to the analysis process and apply a classifier to the image information and not to facial features information. Unlike feature-based methods, which extract predefined features from the face resulting

in small sized vectors, in a holistic-based approach the high-dimensionality of the data is one of the major problems, as every pixel in the image is considered as a separate dimension, bearing implications in computation efficiency when processing the data. Besides, low-dimensional representations are highly desirable for large databases, fast adaptation and good generalization [13]. Thus, several methods for dimensionality reduction and data representation were proposed and applied in holistic-based systems. Some examples can be seen in Table 1 in Appendix A. Given that the system to be implemented for the dissertation project is intended to implement an holistic-based method using the eigenfaces approach of Turk and Pentland [40], more emphasis will be given to the description of this work, namely its utilization of Principal Component Analysis (PCA) to achieve dimensionality reduction and the classification strategies used to perform identification.

Eigenfaces Method. This method categorizes facial recognition as a two-dimensional problem, despite the fact that a face is a complex 3-D object and would typically require a more detailed model for its analysis. Some constraints can be applied by taking advantage of the fact that faces are usually in its normal upright position and can be described with a small set of of 2-D characteristic views. As this is an holistic-based method, it is susceptible to variations in illumination, rotation and scale, as described above, so the goal is to develop a computational model which is fast, reasonably simple and accurate in constrained environments such as an office or a household.

The foundation of the eigenfaces method lies in an information theory approach of coding and decoding face images, emphasizing the significant local and global "features" of the face. Although they share the same name, these "features", or characteristics, may not be directly related with the intuitive notion of facial features existing in feature-based methods. Similarly, yet distinctly, they represent the characteristics of the *image itself* which enable a better discrimination of the given face. Hence its susceptibility to variations on the background or the light. The general idea is that the relevant characteristics should be extracted from the image, encoded in some efficient format and compared to the encoded information of other faces in the database. To achieve this result, the authors proposed the use of an encoded representation technique developed by Sirovich and Kirby [36] based on PCA, which can economically represent an image set using a best coordinate system. In the image set, each pixel is considered a dimension by itself, so an image corresponds to one point in a high-dimensional coordinate system, which implies a large computation effort to process. By utilizing PCA, the number of dimensions in an image set can be reduced to a representation they called *eigenpictures*, while Turk and Pentland called them *eigenfaces*, and thus the name of the method. An example of a typical eigenface image can be found in Fig. 2c.

(a) Face example (b) Average face (c) Eigenfaces

Fig. 2: Eigenfaces method training results

A brief description of the mathematical reasoning behind an eigenface is provided for the sake of completeness, while referring to the work of Sirovich and Kirby [36] for a full explanation. In order to efficiently encode and process a distribution of faces, the most important characteristics in terms of data variation must be calculated. These are the components which will better represent the data and allow for the most accurate reconstruction. In practice, this resumes to finding the eigenvectors, or eigenfaces, of the covariance matrix of the set of images. Note that each image in the database can be reconstructed from a linear combination of the eigenvectors, thus an encoded representation of an image corresponds to the vector containing the coefficients of that linear combination, or, in the nomenclature of Turk and Pentland, the *weights* of each image.

There can be as much eigenvectors as images in a data set of size M, but the goal of PCA is to choose a subset containing only the "best" M' eigenfaces, those with the higher eigenvalues and that best represent the data variation. The number of eigenvectors to store is typically much smaller than the total number of images ($M' << M$) and can be set with an heuristic. Each eigenface represents a coordinate in the new much smaller coordinate system, spanning a subspace of the original coordinate system which the authors named *face space*.

After calculating the eigenfaces for an image set, the process of recognizing a new face can be resumed as: project the image onto each of the eigenfaces to calculate its weights; determine if there is indeed a face in the image by checking if the weights are sufficiently close to the face space; classify the weight pattern as a known or unknown person by comparing with the weights stored in the database according to some distance measure (Euclidean distance in this case); and optionally, include the new face in the database and retrain the system.

Apart from the task of identifying a face, the eigenfaces strategy can also be used to detect faces in images. Faces present much more inter-class than intra-class difference, meaning that it is easier to tell whether an object is a face than to differentiate between two faces. Knowing this, one can identify faces as the regions of the given images that, when projected, bear most similarity to the face space.

3.3.3.3 Hybrid Matching Methods

In a hybrid approach, the best of the two methods described above (holistic and feature-based) is conciliated to create a system that not only processes faces as a whole, but also takes facial features into consideration to create a more robust system. An example of a system in this category will be presented.

In an extension to the original eigenfaces system [40], Pentland et al. [30] developed a system that incorporates both holistic and feature-based methods. In this work it was pointed that the basic eigenfaces method is not prepared for real-world applications, as it was only tested on a database of a few hundred images under constrained conditions. The scalability of the system using just the normal eigenfaces strategy was tested on a large database with 3000 people, with more than one image for each individual and several different views, bearing surprisingly good results. Nonetheless, the system still depends on the problems that affect holistic approaches (i.e. illumination, glasses, etc). The authors then extended the eigenface technique to the description and coding of facial features, yielding eigeneyes, eigennoses and eigenmouths. The evaluation showed that an hybrid stategy can outperform both holistic and feature-based strategies used exclusively.

In order to detect facial features, a similar approach to the whole face detection previously used [40] was applied, making use of a metric similar to the distance-from-face-space to extract them. Refer to [30] for the complete description of the feature extraction.

Possessing both eigenfaces and eigenfeatures, a modular approach can be employed that makes use of both information. Several strategies were studied on the best way of merging the information, such as cumulative scores taking equal contributions by each facial feature; weighting schemes based on psychophysical data; and a sequential classifier, where eigenfaces are used to narrow the scope of database search to a region of the face space, followed by the use of eigenfeatures to perform the final classification [30].

3.3.4 Video Face Identification

So far, all the identified methods for facial recognition were developed with still images in mind. The next step in the evolution of face identification systems is to detect and identify faces in videos, which provide not only a whole new set of challenges, but also properties that still images do not possess and that can be useful if exploited (e.g. motion). Among the most relevant challenges there are the low-quality of the videos, either by outdoor captures or other bad conditions for video capture, and the small size of face regions when compared with still-images. Video face identification can be seen as an extension of still-image face identification, as the earliest attempts were focused

on first detecting and segmenting faces from a video and then applying still-image techniques. Yet, in [50] it is claimed that true video-based face recognition uses both spatial and temporal information.

An improvement over the previous strategy is to add tracking capabilities to the system [50]. A tracking-enabled system can analyze and estimate the motion of a face and recover some 3-D spacial information about it, such as rotations and translations. This enables the synthesization of a virtual frontal view from the various frames by estimating pose and depth, which can then be processed by a regular still-image based technique. Also, since there can be a high number of frames in a video, the recognition rate can be improved by applying a voting strategy, where the recognition results from each frame will vote for the most likely virtual face representation.

According to [50], the next step in this area is the implementation of multimodal systems, which make use of several information sources to reach an identification. The reasoning behind this concept is that humans make use of more information than the face, like voice or body motion, and the usage of multimodal cues can offer a solution which would not be achieved by using face images alone.

Finally, recent approaches in video-based face identification try to make use of one of the most important characteristics of video over still-images, the temporal semantics hidden in the video face sequence. By exploiting this property and its relation with the spatial information considered in face tracking, a system can effectively capture the dynamics of pose changes and reduce occlusion effects. The research tendency is for the appearance of more systems based on this strategy. Some examples of systems in the three described categories can be found in Table 2 in Appendix A.

3.3.5 Face Identification in the Proposed System Context

Bridging the gap between the subject of human face identification and the system to be developed, the goal, as already stated, is to provide a fast and scalable system to identify human faces in videos. Among the several existing approaches, the system will use the eigenfaces strategy of Turk and Pentland [40] applied to image sequences, but it should be designed in a modular way in order to leave room for easily integrating future improvements and different recognition strategies (e.g. eigenfeatures). There will not be much effort in the direction of improving the existing algorithm or developing a new one, as it is not the goal of this work. Yet, some amendments can be introduced in order to better integrate the recognition module in the system.

4 Proposed Solution

Given the theoretical and technological background obtained from the previous study, we are now able to present our solution to the problem of facial identification in videos leveraging cloud resources. We will start by an overview of the software architecture of the system and a description of its main components, followed by the workflow steps in the most basic use-case and some relevant topics that do not fit in the architecture section.

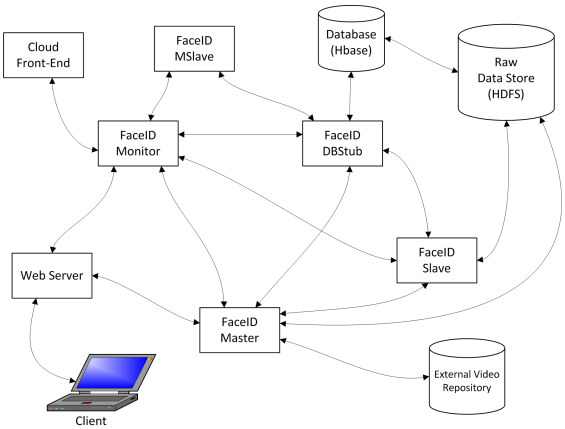

Fig. 3: Component Interaction Diagram

Overview. We start by presenting a high-level view of the system's components and their relations in Fig. 3, describing each one and their interactions in more detail afterwards. In this diagram, each software component is represented either by a rectangle or a cylinder and is not necessarily assigned to a single VM instance in exclusivity, meaning that several components can coexist in the same node. A typical use-case would be for a client to contact the web-server by supplying a new video for the system to process, hoping to get some information about the people that appear in the video. Information could be presented as subtitles stating the moments where some appears, or as a list of names and images identifying the participants.

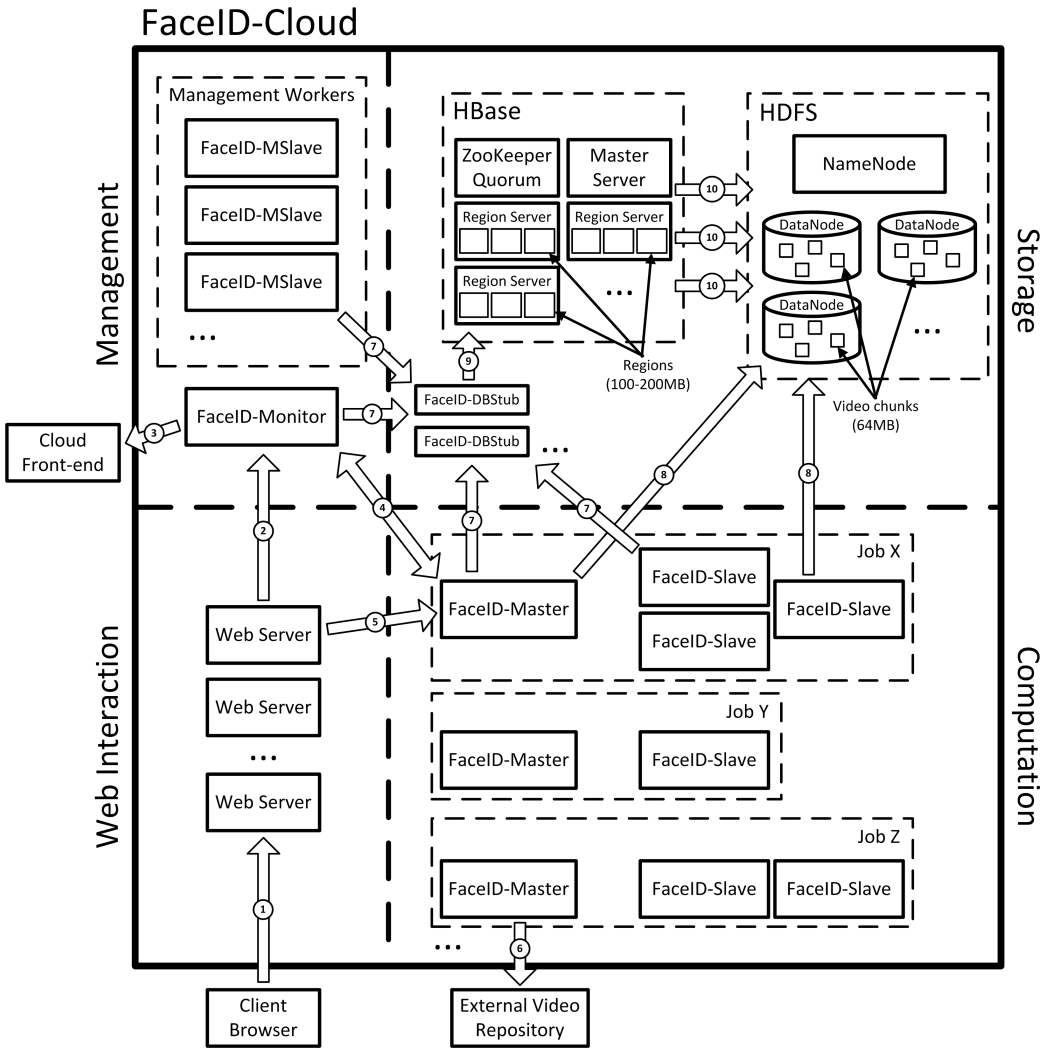

Fig. 4: System Architecture. Interactions: **1.** Client submits new video or query and receives results; **2.** Web server submits new job and gets an assigned master from monitor; **3.** Monitor can allocate and deallocate components through the Cloud Front-end; **4.** Master can ask for more slaves from the monitor; **5.** Web server submits detailed job information to the master and receives the computation results; **6.** Master can fetch videos from external sites; **7.** All accesses to the database are mediated by stubs; **8.** Video chunks are stored and read in HDFS by masters and slaves, respectively; **9.** Stubs perform reads and writes in HBase; **10.** HBase depends on HDFS to physically store regions

4.1 Job Description

A job in this system corresponds simply to the processing of one video. Each job is constituted by several smaller tasks derived from partitioning the video into small chunks with a fixed size of 64MB (the standard size of the filesystem block), where a chunk corresponds to the smallest work unit and cannot be split. Thus, if a node cannot process a whole chunk, that task has to be repeated. Each task is independent from all the others, as no temporal relation between frames is being exploited, and so they can be assigned without any constraints but data location.

Each job has its own entry in the database system where the master can fill in all the necessary information for its completion, including the distribution of the load between slaves. The outcome of a job is obtained by merging the results from each slave and eliminating redundancies and errors.

4.2 System Topology

In Fig. 4, a detailed view of the system's topology is provided, highlighting a clear separation in four areas: Web Interaction, Management, Storage and Computation. This separation is important both to clearly identify the responsibility of each area and to achieve loose coupling between them. We will now proceed with a description of the system's components.

Management. Due to the elastic nature of the virtualized resources, it emerges that a centralized way to manage resources is needed, or at least semi-centralized way accounting for a larger systems spawning different geographic regions. It should be simple for a client to launch a new task by contacting a centralized known component.

Thus, we introduce the component **FaceID-Monitor**, referenced simply by *monitor* for the rest of the text. This component is primarily responsible for managing the resources, allocating and deallocating on-demand by interfacing with the infrastructure's front-end component. It should contain information about all the running jobs and the resources assigned to them.

Another responsibility of the monitor is to assign and manage priorities between jobs, which will condition the order of execution in case of a loaded system. There are three main rules to be followed when assigning priorities to jobs, which are presented by order of importance (i.e. rules appearing first on the list are applied first and win when in conflict with others):

1. Jobs derived from real-time user requests take precedence over background batch workloads, unless otherwise specified by an administrator.
2. Small jobs take precedence over large jobs, where a small job corresponds to videos that fit in less than one chunk, in an attempt to answer the most requests in the shortest time. This rule does not hold if a large job has already been postponed for more than an heuristically calculated threshold (e.g. time-waiting over video size).
3. Finally, the ground rule is that older jobs take precedence over newer jobs, with a timestamp assigned by the monitor.

We are aware that the monitor represents a single point of failure for the system. Nonetheless, in case of failure, all the running jobs should terminate successfully, despite the impossibility of accepting new jobs until the monitor is running again. Three types of failures can happen that would stop the monitor component: node failure, cloud middleware failure and the monitor application itself crashing. We delegate the responsibility to recover from the first two situations to the cloud infrastructure and expect to have a complete uncorrupt version of the monitor instance available after the recovery. We then focus on recovering a crash from the monitor application. What we propose to solve this problem is a simple small application running on the same instance as the monitor process that would receive complaints from other components when the monitor does not answer requests. After some predefined number of complaints, the application would kill the monitor process (in case it is still running) and restart it. To try to partially recover the lost state at the time of failure, the monitor should keep a very small log in its local storage containing snapshots of the in-memory state of the system. The intervals between the snapshots should be enough to cover most of the state, but should not be too small so that disk I/O does not become a bottleneck.

Finally, the diagram in Fig. 4 shows another type of component in the management area, **FaceID-MSlave** (referenced as *m-slaves* for the rest of the document), which are a special type of slaves whose master is the monitor itself. The monitor can use m-slaves to perform heavier management subtasks such as online training of the eigenfaces data or statistics gathering.

Storage. In this system there are essentially two types of data that need to be stored: raw video chunks and processed information, such as knowledge about people and videos, and eigenfaces related data. To address this need, two different strategies are proposed according to the type of data.

To store video chunks, it suffices to have a large distributed data store following a write-once-read-many access model for the stored files. Any regular distributed file system can be used to fill this task, but since scalability is part of the main goals of this work, we plan to use the Hadoop Distributed

File System (HDFS)[9] that is part of the Hadoop framework ecosystem and capable of scaling up to thousands of datanodes. HDFS will be referred simply as *datastore* for the rest of this text.

Information obtained from the processing of videos should be more carefully stored in a database system, as it has to be organized and indexed to allow fast queries to be performed. Two options were considered for the database system, relational databases and key/value databases. Relational databases, although the dominant model in enterprise applications, have known scalability issues when dealing with a huge number of nodes and are typically not suited for a cloud-based application. Key/value databases, although lacking built-in data integrity and some query capabilities available in relational databases, are much less complex and tend to scale very well for thousands of nodes. For this reason, and also to leverage the choice of HDFS as the file system, we propose the use of HBase[10] as the database system. HBase is a column-oriented database that runs on top of HDFS and is capable of random, realtime read/write access to data, so it fits very well in the requirements. HBase will be referred as *database* for the rest of this text.

The data storage part of the system will run in separate nodes which will not take part in computation jobs, being always available and having sufficient space to accommodate all the data at any given moment. It is important to make this separation between storage and computation since they possess different allocation strategies. Storage has an amount of nodes that will typically not be deallocated for a long time, while new nodes can still be added as data needs increase. The computation nodes can go from none to the maximum allowed by the cloud infrastructure, and so are much more elastic.

Finally, there is still one issue that must be address concerning the lack of some querying functionality by HBase. In a key/value database, joins must be performed by the client application, meaning that one needs at least the double of database round-trips to perform an operation concerning two tables. To partially reduce this inefficiency, we propose to add a stub application to the nodes running HBase, called **FaceID-DBStub** or *stub*, which will perform improvements to queries transparently to database clients, such as an in-memory cache for "hot" cells or others (e.g. index). This stub will leave room for future improvements and even facilitate a database migration to a new platform.

Computation. To handle the jobs that are delivered to the system, the monitor can instantiate two types of components, **FaceID-Master** and **FaceID-Slave** (treated simply as *master* and *slave* for the rest of the document). These are the components responsible for the processing of video material and application of face identification algorithms. A description of their functionality will now be given.

A master component is instantiated whenever there is no other free master that can handle a new job that arrived. All the responsibility of the job is transfered to the master, so that the monitor becomes free to answer other requests. The master is prepared to retrieve the video to be analyzed and stores it in the raw datastore already in the form of chunks. This component can ask the monitor for slaves to perform the job according to the size of the video, but the number of actual allocated slaves will be determined by the monitor according to the system load. In a normal state the master will get all the slaves it asks. Finally, the master's main function is to distribute the load evenly among the slaves it controls and build a merged list of people from the individual results.

The slave components are instantiated on-demand when a master requests them from the monitor for a new job to which they become assigned, although they can be directly transferred from a job that just ended to a new one and avoid a deallocation. The slaves' function is simply to apply the face identification algorithm on the video chunks assigned to them by the master using the eigenfaces data on the database, and to store the results in the database.

Web Interaction. In this area there is only one main component, **Web Server**, providing a user interface to interact with the system. This component acts a middle-man between users and the system, simply contacting the system monitor when submitting new jobs and displaying the results to

[9] http://hadoop.apache.org/hdfs/
[10] http://hbase.apache.org/

the user. The Web Server is part of the system and also runs inside a VM instance, or multiple VMs in case of more than one server.

Support Components. There are some components represented in the diagram on Fig. 3 which are not part of the core of the system, but give support to its execution. The first of these components is the **Cloud Front-end**, which makes the bridge with the cloud infrastructure where the system runs and enables the monitor to allocate and deallocate VM instances on-demand. It certainly has many other roles, but in respect to the system itself this is the only function needed. Finally, the **External Video Repository** represents all external sources of video material where the system can download video according to the location presented by the user.

4.3 Workflow

To better describe the execution of a job in the system, the steps taken for its completion will be presented:

1. A client contacts the Web Server and provides a URL/URI for the video.
2. The Web Server contacts the system monitor in order to submit the new job.
3. The monitor checks the system state for masters waiting to be deallocated or on the last phase of a previous job. If such a master exists, the job will be assigned to it. If not, a new master component is instantiated by communicating with the Cloud Front-end.
4. The Web Server is given information about the master and sends it the job information.
5. If the video is already known to the system, the last step is executed. Else, the master downloads the video from the given URL/URI, stores it in chunks of 64MB in the raw datastore and fills the new information on the database to start the processing. According to the size of the video, some slaves will be requested from the system monitor.
6. Upon receiving the slaves information, the master distributes the load between them and waits for termination. Dynamic load balance may be performed by the master if some slaves end their work earlier.
7. Each slave fills in its processed information in the database.
8. The master merges the results from every slave and creates a final list of people that were identified in the video, storing this information in the database.
9. The answer is retrieved from the database and sent to the Web Server that presents it to the client.

This workflow describes the basic interaction with the system, but some other typical interaction could include queries for videos where a person appears. In both cases, in order to avoid some expensive database access, results would only be partially shown (e.g. 10 at a time), starting with the ones with highest confidence level. Other results would be requested on-demand by the user, but hopefully most times the first 10 will be enough.

4.4 Data Model

Since the chosen database is column-oriented, the data model will be very simple and adapted to this type of database, containing only 5 tables:

- **Videos:** general information about videos, chunk location and people identified in frames.
- **People:** general information about people, the videos on where a person appears, feature weight vector for every face image of a person, and a mapping of faces to frames.
- **EigenfacesData:** eigenface related data, containing the results from the training process.
- **Jobs:** information about running and past jobs, containing also the people found by the slaves.
- **User:** information about users that use the face identification service in the website.

This is only an overview of the data model, whereas the full specification can be found in Table 4 of Appendix A.

4.5 Training

In the original eigenfaces algorithm, the training of the system is performed offline in a limited training set, prior to the execution on test data. However, our system is to deal with an increasing amount of people, whose types of faces will not certainly be covered by the training data (namely among different ethnic groups), and so it should be able to perform online training when "needed". It is hard to assert when a re-train of the system is required, but we propose the usage of several heuristics: average identification confidence level; precision of the system based on user-feedback; number of new persons added to the database; time since last training. Also, we want the training to be performed online and without fully stopping the system, trying to keep accepting new jobs with minimal disturbance. To achieve this, the system monitor allocates m-slaves that will perform the training and store the results on a new row in the EigenfacesData table. When the training is complete, all new jobs will be directed to use the new eigenfaces database, while all the already executing jobs will finish their work with the old information, leading to a smooth transition between database versions.

4.6 Implementation Details

We now give some detailed information on the proposed technologies to be used in the system, starting with a layered view of each component/group of components on Fig. 5.

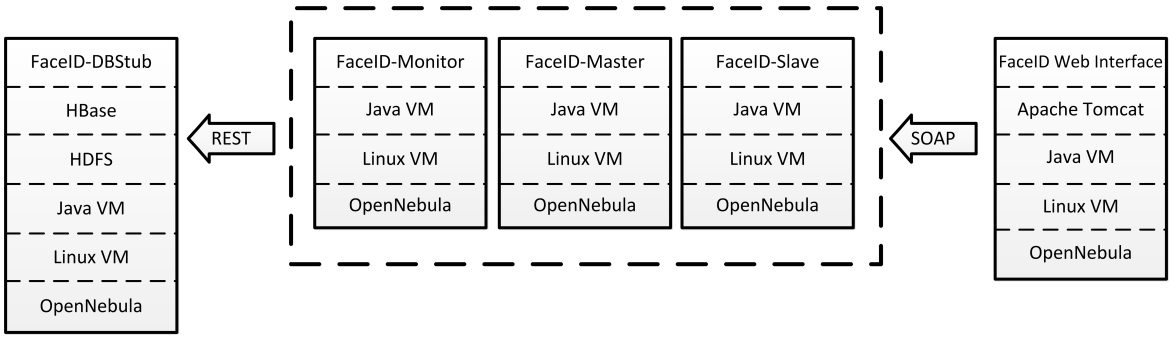

Fig. 5: Component Software Stack Layered View

The cloud technology onto which the system shall be deployed is OpenNebula[11], an open-source solution that enables IaaS on top of a private datacenter. It supports a relevant subset of Amazon's EC2 query interface and has a large community support. The VM instances will run a Linux based distribution, as it is more easily tuned and configurable. The Java VM on top of the operating system is necessary since HDFS, and HBase consequently, is java-based, so we intend to develop the whole system in the Java programming language. Communication with the Storage area is intended to be performed using a REST API and not with a Java API, so that storage clients are not forced to use Java. REST allows for a better performance by reducing the message sizes when compared to SOAP, for example. For the web interface, Apache Tomcat[12] has all the functionality needed by the system. To communicate with the monitor, the web server shall in its turn use SOAP, providing stateful operations and asynchronous processing.

5 Evaluation Methodology

In order to test the performance and stability of the proposed solution, some evaluation methodologies are now described. As one of the goals is to improve the performance of a face identification technique in a centralized environment, one of the metrics to evaluate is the time comparison between the centralized and distributed versions when processing a single video. Other evaluation metrics are related to the behaviour of the system under different conditions to see how the performance is affected and how the system reacts. This quantitative evaluation includes the following topics:

[11] http://opennebula.org/
[12] http://tomcat.apache.org/

- CPU utilization at each component
- Overall number/size of messages exchanged and bandwidth transferred
- Number of requests answered with different system loads
- Time to process different sized videos
- Cold and hot cache scenarios querying times
- System responsiveness while training
- Enforcement of priorities
- Failure recovering
- Amount of repeated (wasted) processing

Finally, some qualitative evaluation metrics will also be applied based on user feedback, such as:

- Precision of the system (true and false positives)
- Relevance of first presented results
- More subjective waiting time evaluation (e.g. fast, slow)

6 Conclusions

This "newcomer" named Cloud Computing really is changing both service provider and user perspectives about Internet and its potential to simplify the delivery of IT services and its consumption by the general public. Major IT companies such as Amazon, IBM and Microsoft are already eagerly moving into this paradigm, setting the pace for all the other providers. Cloud based systems are definitely here to stay and all the efforts should be put into developing new cloud-based solutions which take advantage of the vast amount of on-demand resources available. Face identification is clearly one of the subjects that can benefit from being transferred to the cloud, as it is computationally expensive and can be applied in countless fields.

Our main goal was to design a system that can horizontally scale performance- and storage-wise in a graceful way with the addition of new nodes, making use of the elastic nature of virtual cloud clusters to speedup the recognition process in large video databases. To achieve this, we separated computation and storage into two different parts, allowing computation nodes to be allocated and deallocated on-demand, without interfering with a stable set of nodes dedicated to storage which can easily be extended on a more static way. It is important to note that this is a data-oriented system, since each job is represented by a video chunk, conditioning computational tasks to the availability of data. The load-balancing and job execution is addressed in two levels by the introduction of a system monitor and master components which aims to drive an efficient use of the resources and maintain the responsiveness of the system. To improve the overall face identification precision rate overtime, we proposed an online training strategy that tries not to disrupt the system while rebuilding the eigenfaces database.

From the survey of the state of the art technologies and background theory on the areas of Cloud Computing, Data-Driven Scheduling and Face Identification, we think that our solution has a solid foundation and deals with most of the existing challenges. Also, we think that, given a good implementation, this system can definitely be used in a real-world scenario, processing both videos in real-time by user request and large video databases on an institutional repository.

References

1. Armbrust, M., Joseph, A.D., Katz, R.H., Patterson, D.A.: Above the Clouds : A Berkeley View of Cloud Computing. Tech. rep., University of California at Berkeley (2009)
2. Bartlett, J.C., Searcy, J.: Inversion and configuration of faces. Cognitive psychology 25(3), 281–316 (Jul 1993), http://www.ncbi.nlm.nih.gov/pubmed/8354050
3. Belhumeur, P., Hespanha, J., Kriegman, D.: Eigenfaces vs. Fisherfaces: recognition using class specific linear projection. IEEE Transactions on Pattern Analysis and Machine Intelligence 19(7), 711–720 (Jul 1997), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=598228
4. Bent, J., Denehy, T.E., Livny, M., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Data-driven batch scheduling. In: Proceedings of the second international workshop on Data-aware distributed computing - DADC '09. pp. 1–10. ACM Press, New York, New York, USA (2009), http://portal.acm.org/citation.cfm?doid=1552280.1552281
5. Bojanova, I., Samba, A.: Analysis of Cloud Computing Delivery Architecture Models. In: Workshops of International Conference on Advanced Information Networking and Applications. pp. 453–458. IEEE (Mar 2011), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5763543
6. Brunelli, R., Poggio, T.: Face recognition: features versus templates. IEEE Transactions on Pattern Analysis and Machine Intelligence 15(10), 1042–1052 (1993), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=254061

7. Buyya, R., Yeo, C.S., Venugopal, S.: Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In: 10th IEEE International Conference on High Performance Computing and Communications. pp. 5–13. Ieee (Sep 2008), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4637675
8. Chellappa, R., Wilson, C., Sirohey, S.: Human and machine recognition of faces: a survey. Proceedings of the IEEE 83(5), 705–741 (May 1995), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=381842
9. Choudhury, T., Clarkson, B., Jebara, T., Pentland, A.: Multimodal Person Recognition using Unconstrained Audio and Video. Tech. rep. (1999)
10. Cox, I.J., Yianilos, N.: Feature-Based Face Recognition Using Mixt ure-Distance Joumaiia Ghosn. Tech. rep. (1996)
11. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. 6th Symposium on Operating Systems Design & Implementation 51(1), 107–113 (2008)
12. Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR (2005)
13. Etemad, K., Chellappa, R.: Discriminant analysis for recognition of human face images. Journal of the Optical Society of America A 14(8), 1724 (Aug 1997), http://www.opticsinfobase.org/abstract.cfm?URI=josaa-14-8-1724
14. Geelan, J.: Twenty-One Experts Define Cloud Computing (2009), http://virtualization.sys-con.com/node/612375
15. Ghemawat, S., Gobioff, H., Leung, S.t.: The Google File System. Tech. rep., Google (2003)
16. Gong, C., Liu, J., Zhang, Q., Chen, H., Gong, Z.: The Characteristics of Cloud Computing. In: 39th International Conference on Parallel Processing Workshops. pp. 275–279. IEEE (Sep 2010), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5599083
17. Hongxun, Y., Wen, G., Mingbao, L., Lizhuang, Z.: Eigen features technique and its application. In: 5th International Conference in Signal Processing Proceedings. vol. 2, pp. 1153–1158 (2000)
18. Hutchinson, C., Ward, J., Information, C.: Navigating the Application Architecture. IT Professional, vol. 11 no. 2 (April), 18–22 (2009)
19. IBM, Google: Google and IBM Announce University Initiative to Address Internet-Scale Computing Challenges (2007), http://www-03.ibm.com/press/us/en/pressrelease/22414.wss
20. Kanade, T.: Computer Recognition of Human Faces (1973)
21. Kosar, T., Balman, M.: A new paradigm: Data-aware scheduling in grid computing. Future Generation Computer Systems 25(4), 406–413 (Apr 2009), http://linkinghub.elsevier.com/retrieve/pii/S0167739X08001520
22. Kosar, T., Livny, M., Street, W.D., Wi, M.: Stork : Making Data Placement a First Class Citizen in the Grid. In: Proceedings of the 24th International Conference on Distributed Computing Systems. IEEE (2004)
23. Lawrence, S., Giles, C.L., Tsoi, a.C., Back, a.D.: Face recognition: a convolutional neural-network approach. IEEE transactions on neural networks 8(1), 98–113 (Jan 1997), http://www.ncbi.nlm.nih.gov/pubmed/18255614
24. Lin, S.H., Kung, S.Y., Lin, L.J.: Face recognition/detection by probabilistic decision-based neural network. IEEE transactions on neural networks 8(1), 114–32 (Jan 1997), http://www.ncbi.nlm.nih.gov/pubmed/18255615
25. Liu, C., Wechsler, H.: Evolutionary pursuit and its application to face recognition. IEEE Transacations on Pattern Analysis and Machine Intelligence 22(6), 570–582 (2000), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=862196
26. Liu, X., Chen, T., Engineering, C.: Video-Based Face Recognition Using Adaptive Hidden Markov Models. In: Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (2003)
27. Navarrete, P., Ruiz-del solar, J.: Analysis and Comparison of Eigenspace-based Face Recognition Approaches. Tech. rep. (2002)
28. Nefian, A., Hayes III, M.: Hidden Markov models for face recognition. In: Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on. vol. 5, pp. 2721–2724. IEEE (1998), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=678085
29. Oracle: Database as a Service : Reference Architecture An Overview. Tech. Rep. September (2011)
30. Pentland, A., Moghaddam, B., Starner, T.: View-based and modular eigenspaces for face recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. vol. 02139, pp. 84–91. IEEE Comput. Soc. Press (1994), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=323814
31. Prodan, R., Ostermann, S.: A survey and taxonomy of infrastructure as a service and web hosting cloud providers. In: 10th IEEE/ACM International Conference on Grid Computing. pp. 17–25 (2009)
32. Rajan, S., Jairath, A.: Cloud Computing: The Fifth Generation of Computing. In: International Conference on Communication Systems and Network Technologies. pp. 665–667. IEEE (Jun 2011), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5966533
33. Rappa, M.a.: The utility business model and the future of computing services. IBM Systems Journal 43(1), 32–42 (2004), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5386779
34. Sakr, S., Liu, A., Batista, D.M., Alomari, M.: A Survey of Large Scale Data Management Approaches in Cloud Environments. IEEE Communications Surveys & Tutorials 13, 311–336 (2011), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5742778
35. Sinha, P., Balas, B., Ostrovsky, Y., Russell, R.: Face Recognition by Humans: Nineteen Results All Computer Vision Researchers Should Know About. Proceedings of the IEEE 94, 1948–1962 (Nov 2006), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4052483
36. Sirovich, L., Kirby, M.: Low-dimensional Procedure for the Characterization of Human Faces. Journal of the Optical Society of America A 4, 519 (1987)
37. Stewart, M., Lades, H.M., Sejnowski, T.J.: Independent component representations for face recognition. In: Proceedings of the SPIE Symposium on Electronic Imaging: Science and Technology, Conference on Human Vision and Electronic Imaging III. California (1998)
38. Treleaven, P.C., Brownbridge, D.R., Hopkins, R.P.: Data-Driven and Demand-Driven Computer Architecture. Computing Surveys 14(March) (1982)
39. Tsai, W.T., Sun, X., Balasooriya, J.: Service-Oriented Cloud Computing Architecture. In: Seventh International Conference on Information Technology. pp. 684–689. IEEE (2010), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5501650
40. Turk, M., Pentland, A.: Face recognition using eigenfaces. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. pp. 586–591. IEEE Comput. Sco. Press (1991), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=139758
41. Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A Break in the Clouds: Towards a Cloud Definition. ACM SIGCOMM Computer Communication Review 39(1), 50–55 (Dec 2008), http://dl.acm.org/citation.cfm?id=1496100
42. Voas, J., Zhang, J.: Cloud Computing : New Wine or Just a New Bottle ? IT Professional 11(2), 15–17 (2009)
43. Vouk, M.: Cloud computingIssues, research and implementations. Journal of Computing and Information Technology 16(4), 235–246 (2008)
44. Wang, L., Laszewski, G., Younge, A., He, X., Kunze, M., Tao, J., Fu, C.: Cloud Computing: a Perspective Study. New Generation Computing 28(2), 137–146 (Jun 2010), http://www.springerlink.com/index/10.1007/s00354-008-0081-5
45. Wang, L., Tao, J., Kunze, M., Castellanos, A.C., Kramer, D., Karl, W.: Scientific Cloud Computing: Early Definition and Experience. In: 10th IEEE International Conference on High Performance Computing and Communications. pp. 825–830. Ieee (Sep 2008), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4637787
46. Weinhardt, C., Anandasivam, A., Blau, B., Stöß er, J.: Business Models in the Service World. IT Professional 11(April, no.2), 28–33 (2009)
47. Wiskott, L., Fellous, J.M., Kuiger, N., von der Malsburg, C.: Face recognition by elastic bunch graph matching. IEEE Transactions on Pattern Analysis and Machine Intelligence 19(7), 775–779 (Jul 1997), http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=598235
48. Wood, T., Shenoy, P., Gerber, A., Ramakrishnan, K.: The Case for Enterprise-Ready Virtual Private Clouds. Tech. rep. (2009)
49. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications 1(1), 7–18 (Apr 2010), http://www.springerlink.com/index/10.1007/s13174-010-0007-6
50. Zhao, W., Chellappa, R., Phillips, P., Rosenfeld, A.: Face recognition: A literature survey. Acm Computing Surveys 35(4), 399–458 (2003)

A Appendix: Additional Tables

Approach	Sub-class	Example systems
Holistic-based	Principal Component Analysis	Turk and Pentland[40]
	Fisher's Linear Discriminant	Belhumeur et al.[3]
	Linear Discriminant Analysis	Etemad and Chellappa[13]
	Independent Component Analysis	Stewart et al.[37]
	Evolutionary Pursuit	Liu and Wechsler[25]
Feature-based	Purely Geometric Methods	Kanade[20]; Cox et al.[10]
	Dynamic Link Architecture	Wiskott et al.[47]
	Hidden Markov Model	Nefian and Hayes[28]
	Convolution Neural Network	Lawrence et al.[23]
Hybrid-based	Modular Eigenfaces	Pentland et al.[30]
	Probabilistic Decision Based Neural Networks	Lin et al.[24]

Table 1: Classification of Face Identification systems

Approach	Example systems
Still-image methods	Turk and Pentland[40]; Pentland et al.[30]; Lin et al.[24]
Multimodal methods	Choudhury et al.[9]
Spatio-temporal methods	Liu and Chen[26]

Table 2: Classification of Video Face Identification systems

Feature	Amazon Web Services	Microsoft Windows Azure	Google App Engine	Eucalyptus	OpenNebula
Computing Architecture	Elastic resources (EC2), multiple instance types operating systems and software packages; elastic IPs; availability zones; automatic load balancing	Automatic management of VM instances and load balancing	Automatic scaling and load balancing; task queues	Cloud requests serviced asynchronously; elastic IPs; automatic scaling and load balancing	Dynamic resizing and partitioning; centralized management; utilizes existing heterogeneous resources
Service Model	IaaS; PaaS (Beanstalk)	PaaS	PaaS	IaaS	IaaS
Deployment Model	Public; Virtual Private Cloud	Public; Hybrid	Public; Virtual Private Cloud (SDC)	Public; Private; Hybrid; Community	Public; Private; Hybrid
Virtualization Technology	Xen	Windows Azure Hypervisor (Hyper-V)	No hypervisor	Hypervisor-agnostic architecture (Xen, KVM compatibe at the moment)	Hypervisor-agnostic architecture (Xen, KVM and VMare compatible at the moment)
Storage System	"Bulk" storage (S3); query-capable non-relational database (SimpleDB)	Blobs, Tables, SQL Azure	BigTable	Walrus (S3 compatible)	SQLite; supports most file systems through textual configuration files
API	SOAP; REST	Windows Azure Service Management API	APIs provided for main services (e.g. datastore)	AWS-compatible	XML-RPC; EC2 Query subset; OGF OCCI
Programming Model	Amazon Machine Images; Amazon Elastic MapReduce	Windows Azure Hosted Services Application Model	Google App Engine SDK	Unknown	Unknown
Service Level Agreement	99.95% uptime, client elegible to Service Credit otherwise	99.95% uptime for two or more instances in different domains, 99.9% for all other services	Up to 99.95% uptime with different client Financial Credit compensations	Best-effort basis only on community deployment	Not applicable (open-source)
Pricing	Per hour (computation); per GB (network, storage)	Per instance (computation); per GB (network, storage); 6-month plans	Per hour (computation); Per GB (network, storage); per operation count (datastore)	Not applicable	Not applicable
Security	Firewall between instance groups; VPC; identity and access management; security groups	Confidentiality, integrity, availability, accountability	Multi-layered security strategy; multiple levels of data storage, access, and transfer	Similar to EC2, except VPC	Auth subsystem for users and groups
Component Coupling	High coupling (e.g. EC2 / S3)	High coupling	Access to some services is code-based (APIs), so allows other substitute components in principle (medium coupling)	Substitute components need to implement Web Services interface (medium coupling)	Low coupling; in general, substitute components can be attached with minor effort

Table 3*a*: Cloud Computing systems classification

Feature	Nimbus	Flexiscale	IBM SmartCloud Services	Force.com	OpenStack
Computing Architecture	Fast-propagation; non-invasive site-scheduler; auto-configurable clusters	Highly automated and rapid provisioning of resources; allows multi-tier architecture; designed to deliver a guaranteed QoS level	Fast deployment and dynamic provisioning of resources accross heterogeneous environments; Integrated into the IBM software and hardware ecosystem	Metadata-driven software architecture enabling multitenant sofware applications	Shared-nothing design; Multiple network models; Floating IPs; Distributed scheduler; Asynchronous architecture
Service Model	IaaS; some PaaS features (Nimbus Platform)	IaaS	IaaS; PaaS; SaaS	PaaS; SaaS	IaaS
Deployment Model	Private; Hybrid	Public	Public; Private; Hybrid; Virtual Private Cloud	Public	Public; Private
Virtualization Technology	Xen; KVM	Xen	Heterogenous hypervisor support; POWER Hypervisor	Unknown	Hypervisor-agnostic (Hyper-V, Citrix XenServer, Xen, KVM, VMWare ESX, LXC, QEMU, UML
Storage System	Posix filesystem backend storage system	Virtual disks on highly redundant SAN disk array	IBM Storwize; IBM XIV; IBM SONAS	Virtual relational database structures	AoE over LVM; S3 compatible; Several supplementary block storage options provided
API	WSRF; EC2 SOAP and Query subset; Cumulus	Extility (SOAP)	RESTful API; Java API	SOAP	Openstack API; EC2 and S3 support
Programming Model	Nimbus Platform	Unknown	IBM SmartCloud Application Services	Force.com Platform	Unknown
Service Level Agreement	Not applicable (open-source)	100% uptime, client credit if not achieved	Up to 99.9% uptime	99.9% uptime	Not applicable (open-source)
Pricing	Not applicable	Unit pricing system for all services	Per hour (computation); per GB (network, storage)	Applications per user per month	Not Applicable
Security	X509 Credentials; group policies	Port based firewall; VLANs and private virtual disks	Firewall; IP-filtering; VPN; patched and scanned images; hypervisor isolation; access control	Access control (digital and physical); firewall and edge routers; intrusion detection sensors; third party scan of the network	Rate limiting and autentication; Security Groups; Role Based Access Control; Federated Auth with Zones
Component Coupling	Low coupling; new components easily integrated	High coupling	High coupling	High coupling	Low coupling; Modular design can integrate with legacy or third-party technologies

Table 3*b*: Cloud Computing systems classification

Table	Row Key	Column Family			
Videos	<hash(video_name)>	info:	chunks:	people:	
		timestamp type → upload|site author_id location → URL video_id	<chunk_id>→ URI	<person_id>→ <frame_id>	
People	<hash(person_id)>	info:	videos:	eigeninfo:	faces:
		stdpicture → image person_id *name* *familyname* *nickname*	<video_id>→ frame_id	<face_id>→ weights	<face_id>→ frame_id
EigenfacesData	<timestamp>	info:	eigenfaces:		
		date ntraining neigens	avgface → image eigenValMat projectedTrainFaceMat <eigen_id>→ eigenface		
Jobs	<hash(video_name)>	info:	results:		
		timestamp nslaves videosize nchunks	<person_id>→ confidence <slave_id>_<unk_id>→ weights		
User	<login_name>	info:	search_history		
		name type → admin|regular *registerdate* *email* login passwdhash	<person_id> <video_id>		

Table 4: Detailed Data Model Specification - tables, rows and column families are specified. Inside each column family there can be any number of columns holding one value each. The elements of this Table follow this scheme: elements enclosed with "<>" represent the possibility of multiple different values; the first element of each line in a cell corresponding to a column family is a column name; elements that follow "→" are values stored in the respective column on that line (values are not specified whenever they are not relevant or the column name is sufficient); elements in italic are optional and provide extra information.

B Appendix: Planning