FaceID-Cloud

Face Identification Leveraging Utility and Cloud Computing

Ricardo Caldeira and Luís Veiga

[1] Instituto Superior Técnico - UTL
[2] INESC-ID Lisboa
ricardo.caldeira@ist.utl.pt luis.veiga@inesc-id.pt

Abstract. Detection and identification of human faces, mostly in photos, 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 achieve scalability in face identification by integrating it in a cloud infrastructure. We propose a system that leverages cloud resources to greatly improve facial identification performance in large video databases, such as those found in social application (e.g Facebook[3], YouTube[4]).

Keywords: Video Face Identification, Cloud Computing, 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 a very fast rate, specially since the appearance of the Internet and the World Wide Web. This led 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 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 is important 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 large databases of videos that can easily reach the terabytes. 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

[3] http://wwwfacebook.com
[4] http://www.youtube.com

generically dubbed "embarrassingly parallel" problems, meaning that it is easily divided into sub-problems with few or 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 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 start with a survey on previous works and state-of-the-art technologies related with this project in the Related Work (see Section 2). Next, we describe the architecture of our solution and present some implementation details in the Architecture (see Section 3). Finally, we present some test results and conclude with a wrap-up of the work along with some final remarks in the Conclusions (see Section 6).

2 Related Work

In this section, a brief overview is made on the cloud computing paradigm and some of the most representative systems in this area. We also explore some face recognition methods, focusing on one of the earliest approaches by Turk and Pentland [4] using eigenfaces for recognition, which our system will use. Some systems in this area will also be described, focusing specially on the performance of database size vs. time.

Cloud Computing. During the past few years, the IT world witnessed the birth and growth of a new paradigm, commonly called **Cloud Computing**. 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 [6], and is also based upon some already existing technologies like distributed computing, virtualization or utility computing, which have been around for several years already[5]. There are some properties that are commonly attributed to this concept:

- **On-demand service provisioning.** The possibility of obtaining and releasing resources on the fly, following the real necessities of an application.
- **Service Orientation.** Most business models around the cloud are service-driven, hence a strong emphasis is placed on Service-Oriented Architectures (SOA), allowing business solutions to create, organize and reuse its computing components. Cloud Computing provides a flexible platform for enterprises to build their SOA solutions, complementing each other [3].
- **Scalability and Flexibility.** In order to support a dynamic service provisioning, cloud infrastructures must be able to cope with very distinct conditions (e.g hardware, software, location) and adapt to different requirements from a large number of users.
- **Pay-per-use utility model.** An utility service model is usually applied to Cloud Computing platforms, featuring a pay-per-use billing, where customers pay solely for what they use, similar to common existing utilities such as water supply or eletric power.
- **Virtualization.** Virtualization techniques provide the means to achieve most of the above mentioned characteristics, namely by partitioning hardware and thus acting as the base for flexible and scalable computing platforms.

Other important characteristics of cloud systems are the service and deployment models. On the former there are three main accepted types: **Infrastructure-as-a-Service (IaaS)** providing virtualized hardware (CPU, disk, network); **Platform-as-a-Service (PaaS)** providing a complete software platform to develop and deploy applications; **Software-as-a-Service (SaaS)** where complete applications are provided to end-users.
About the deployment models, there are also three most used types: **Public** clouds are owned by organizations selling services and offered to the general public or large industry groups; **Private** clouds allow companies to build clouds on top of their own or leased infrastructures; **Hybrid** clouds try to leverage the advantages of both Public and Private clouds [6].

We now expose some of the most representative cloud systems available:

- **Amazon Web Services.** AWS is one of the most known cloud solutions available, providing mainly IaaS and also PaaS with a 99.95% uptime on a public deploy model. Its main features are elastic resource provisioning and load balancing, multiple operating systems and software packages, several storage solutions, availability zones, all accessible through SOAP/REST APIs. Yet, VMs are managed by developers.
- **Microsoft Windows Azure.** Windows Azure provides essentially a complete PaaS service to developers, with development tools and automatic deployment and management of VMs, and uptime guarantees starting on 99.9%.
- **Eucalyptus.** It is an open-source cloud solution, allowing all types of deployment and providing IaaS. It it very similar to AWS, featuring a compatible API and some features such as elastic IPs, automatic scaling and load balancing. Some components present some coupling, so its not easily tuned or extended to a company's needs (e.g. different file system).
- **OpenNebula.** Another open-source cloud infrastructure, similar to Eucalyptus. Unlike the latter, OpenNebula does not focus on AWS features, although providing a subset of its Query API. Besides dynamic resizing and partitioning of heterogeneous resources, it has a centralized management which brings both advantages and disadvantages. It has good support for extension, with low coupling between components.
- **OpenStack.** The youngest open-source cloud platform (of this group) providing IaaS services on public and private deployments and supported by more than 150 companies. It features a shared-nothing design, multiple network models, distributed scheduler and asynchronous architecture, with several storage options and EC2/S3 compatible APIs. It has a modular design with low-coupling between components and can integrate with legacy or third-party technologies.

Face Recognition. During the last 40 years, the problem of face recognition by computers has been the subject of extensive research, with several techniques being suggested. Yet, humans still have better results in this area. During the last decade, the great improvements in computing power have brought new possibilities, enabling techniques that 30 years ago were simply not feasible, CPU- and memory-wise. Simply put, face recognition, or identification, tries to solve the problem of, given a still or video picture, detecting and classifying existing human faces using a database of known faces. A few representative systems in this area are now described, bearing in

mind that the recognition results are **not** directly comparable since the tests were performed on different datasets under distinct conditions. These systems represent 3 types of strategies to deal with the problem (holistic-based, feature-based and hybrid) and so cannot be seen as the state-of-the-art on this area:

- **Eigenfaces.** Proposed by Turk and Pentland [4], its core idea is to handle face images as a whole (holistic approach), extracting relevant information to classify unkown faces with a model built from an existing training face database. A dimensionality reduction is applied (each pixel is considered a dimension) using Principal Component Analysis (PCA) to improve time performance. From this reduction result the eigenvectors of the set of face images, called "eigenfaces" by the authors. Eigenfaces enable the projection of face images on the new dimensions as a small weight vector, which can be used for classification using some distance measure (e.g Euclidean distance). According to the authors, this algorithm reached 96% accuracy over lighting variations, 85% over orientation variation and 64% over size variation.
- **Hidden Markov Model.** Another approach was proposed by Nefian and Hayes [1] using Hidden Markov Models (HMM). The rationale is to map different face features (e.g eyes, mouth) to states on an HMM (as opposed to an holistic approach) and build a face model from training data, which is applied for classification. The authors argue that this approach outperforms the original Eigenfaces algorithm by 11%, reaching 84% on their testing database.
- **View-based Modular Eigenfaces.** There are also systems that use both holistic- and feature-based approaches, like the one proposed by Pentland et al. [2]. In this system, the concept of "eigenfaces" is extended to individual facial features, resulting in "eigenfeatures". The usage of both kinds of information (eigenfaces and eigenfeatures) reached a 98% recognition rate on the test database.

If we explore the application of face recognition to the area of our work, there are not many known systems that explore the same problem of improving recognition raw performance (database size over time). Nonetheless, we mention some systems based on cloud or grid computing:

- **Faces of Facebook**[5]**.** This work is focused on the raising of new privacy concerns from the convergence of face recognition, cloud computing and online social networks. It tries to recognize persons by searching facebook profiles, powered by cloud infrastructures. Although the goals between this system and ours are different, cloud computing is a central piece on both.
- **Grid-Based Parallel Elastic Graph and Template Matching.** These systems parallelize the Elastic Graph and Template Matching algorithms so that they can run in grids, surpassing the eigenfaces algorithm in time performance when the database has 500 faces or more (Elastic Graph) and when the grid has more than 10 nodes (Template). This result show that scheduling activities and network delay have a big impact on small databases.
- **Other systems.** Other representative examples of face recognition in the Web 2.0 era are the automatic photo tagging features on Facebook[6], Google's Picasa

[5] https://media.blackhat.com/bh-us-11/Acquisti/BH_US_11_Acquisti_Faces_of_Facebook_Slides.pdf
[6] https://blog.facebook.com/blog.php?post=467145887130

Web Album[7] or Apple's iPhoto[8]. Unfortunately, these are proprietary systems and not much information is available.

Although some efforts are being made to bring face recognition and cloud computing together, none of the above mentioned systems are dealing with video face recognition, which is a subject that comes with an increased complexity in terms of the necessary processing power and network utilization. We think that cloud computing is the ideal solution to this problem, as the problem is highly parallelizable, but the sheduling effort and resource management (network, storage) are the most difficult challenges.

Also, as the main purpose of this work is not to advance the state-of-the-art in face recognition accuracy by itself, we deliberately chose one of the simplest approaches for our system (eigenfaces), while leaving "software hooks" for better algorithms to be added.

3 Architecture

We start by presenting a high-level view of the system's components and their relations in Fig. 1, 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. There are six main components in the system: **FaceID-Monitor**, which coordinates the whole system; **FaceID-Master**, which handles new jobs; **FaceID-Slave/MSlave**, which performs processing tasks; **Database**, where all processed data is kept; **Datastore**, where raw data is placed (videos, chunks); **WebServer**, which is merely a front-end for clients to interact with the system.

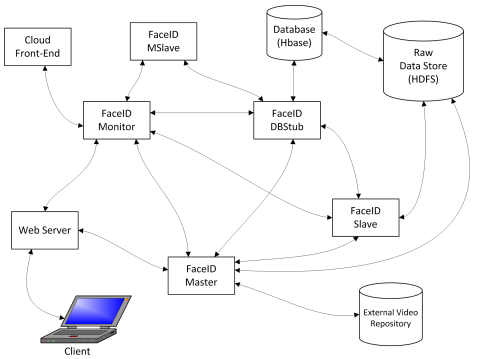


Fig. 1: Component Interaction Diagram

3.1 System Topology

Since there are several different components, with very different types of tasks and interactions, we add another layer of abstraction and group them according to their type of responsibilities: Management, Computation, Storage and Web Interaction (Fig.. This separation is important to clearly identify responsibilities and to achive loose coupling between the components. A brief description is provided:

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 of 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

[7] https://picasaweb.google.com/
[8] http://www.apple.com/ilife/iphoto/

component. To accomplish this we provide the component FaceID-Monitor, responsible for managing all the system's resources, allocating and deallocating VMs on demand by interfacing with the cloud infrastructure's front-end component.

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 recovers. 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 of recovering from the first two situations to the cloud infrastructure, focusing on the third problem. A simple solution is to insert a very small and robust application on the same VM node of the monitor which will receive *suspected* notifications from other components when the monitor is not responding, and will take action in order to put it back on a correct state. The monitor recovers the system state through the database and small local snapshots.

Finally, the diagram in Fig. 2 shows another type of component in the management area, FaceID-MSlave, 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. There are essentially two types of data that need to be stored: raw video and processed information, such as knowledge about people and videos, and eigenfaces related data. Hence, two different storage schemes are used according to the type of data.
To store raw video, a large distributed data store is sufficient, following a write-once-read-many access model for the stored files. Most regular distributed file systems could be used, but since scalability is part of the main goals of this work, the Hadoop Distributed File System (HDFS)[9] was chosen, as it is capable of scaling up to thousands of datanodes. The choice of HDFS as the raw datastore is also connected with the need for a database system to store the information produced by the processing of videos. Both relational (SQL-based) and key/value (NoSQL) databases were considered, but the former are known to manifest some scalability

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

problems, while the later scales better and is much less complex in terms of data access (at the cost of query and data integrity functionalities). Therefore, we decided to use HBase[10] from the Hadoop ecossystem, which runs on top of HDFS.
The system has dedicated storage nodes which do not take part in computation jobs, being always available and having sufficient space to accommodate all the necessary data. It is important to clearly separate data nodes from computation nodes, given that they have different allocation strategies. Data nodes will typically not be allocated and deallocated on demand like computation nodes, being much less dynamic. The expansion of storage space is performed by system administrators by adding new VM nodes and informing HDFS of its location.

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. This group comprises the FaceID-Master and FaceID-Slave components, which are allocated on demand by the monitor and are responsible for the processing of video material and application of the face identification algorithms.
A master is instantiated whenever there is no other free master that can handle a new job that arrived. It fetches the video and stores chunks of it in the datastore, asking the monitor for the necessary resources (slaves) to continue the job, and waits to merge the processed information that the slaves will write on the database.
The slaves' function is simply to apply the face identification algorithm on the video chunks assigned to them by the master and storing the results in the database.

Web Interaction. There is only one component in this group, the Web Server, which provides a user interface to interact with the system. This component acts as a middle-man between users and the system, simply contacting the system monitor when submitting new jobs and displaying the results to the user.

3.2 Work Units

In this system, there are two basic units of work, **jobs** and **tasks**. The former correspond to the processing of one video, and have a single master component assigned to it which coordinates the whole operation. The latter correspond to the partition of the videos in chunks and are assigned to the available slaves, which can be assigned more than one task.

3.3 Workflow

To better describe the execution of a job in the system, the steps taken for its completion are explained in detail:

1. A client contacts the Web Server and provides a URL/URI for the video.

[10] http://hbase.apacher.org

2. The web server contacts the system monitor in order to submit the new job.
3. The monitor checks the system state for masters who do not have an assigned 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 in the raw datastore and fills the new information on the database to start the processing. It then requests slaves from the monitor.
6. Upon receiving the complete 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 then delievered 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.

Data Model. Since the chosen database is column-oriented, the data model is 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.

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 supposed to deal with an increasing amount of people, whose different types of faces will possibly not be adequately covered by the training data, and so it should be able to perform online re-training when "needed". In order to tackle the problem of the very large number of faces (e.g. at 30fps, a 1 minute video with one person will result in $\sim$1800 faces), we intend to subsample the face database and use this smaller subset that include some new faces to train the system, hopefully increasing the accuracy. As of now, this subsampling is performed semi-randomly (at fixed steps according to the database size), but a better strategy may be applied in future work.

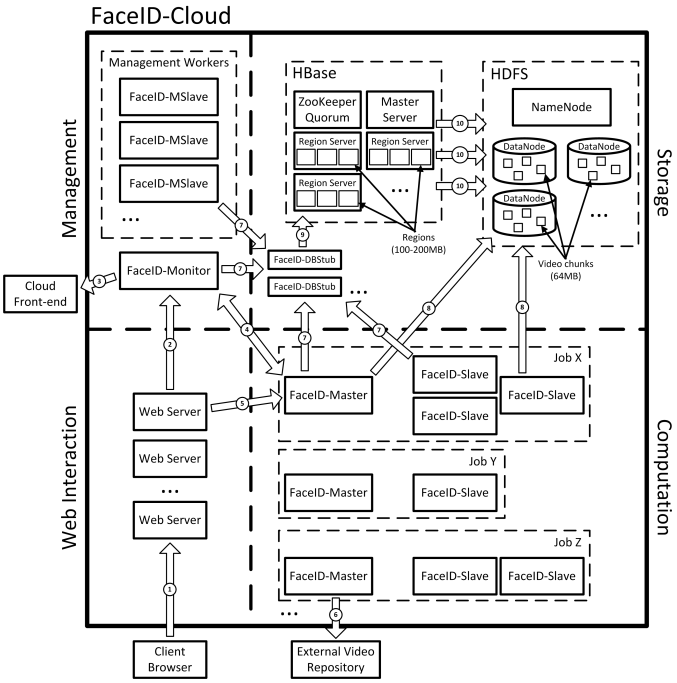


Fig. 2: System Architecture. Interactions: **1.** Client submits new video or query and receives results; **2.** Web server submits new job and gets an assigned master from monitor; **3.** Monitor can allocate and deallocate components through the Cloud Front-end; **4.** Master can ask for more slaves 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.** 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

It is hard to assert when a re-train of the system is required, but we intend to use some simple heuristics: average identification confidence level; precision of the system based on user-feedback; number of new faces added to the database; time since last training. Also, training is to be performed 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 database. 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 with an epoque-based approach.

4 Implementation Details

We now give some detailed information on the technologies used to develop the system, starting with a diagram of the software-stack on each component on Fig. 3.

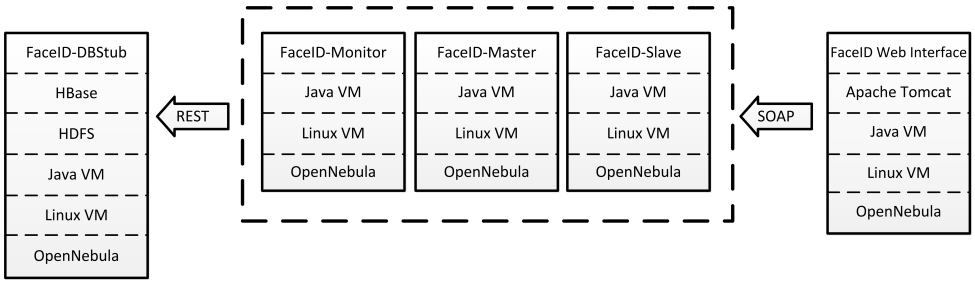


Fig. 3: Component Software Stack Layered View

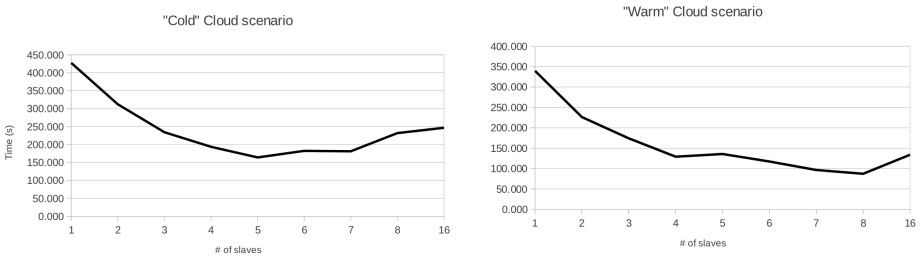
Every component uses a similar stack of software modules, simplyfing the development effort:

- **Deployment Environment.** All components are deployed on VMs running a minimal installation of Ubuntu Server 10.04 LTS (Lucid Lynx). The cloud infrastructure is managed by OpenNebula[11], an open-source solution that enables an Infrastructure-as-a-Service model (IaaS) on top of a private datacenter, and supports a subset of Amazon's EC2 query API. The virtualization infrastructure used together with OpenNebula is KVM. All components were developed in Java for simplicity, as both HDFS and HBase are written in Java and provide APIs in this language. We think Java has sufficient performance to be used in the face recognition module as well, but a C/C++ version can be easily "plugged" if it proves to be a better solution in the long term.
- **Resource Management.** All resources are allocated and deallocated through the monitor component, which interacts with the cloud front-end for this purpose. Hooks were made and abstraction layers were developed in order for a low decoupling between our system and the chosen cloud provider, so that different providers can be easily added (namely through generic interfaces and factory patterns). Specifically about OpenNebula, we use its XML-RPC API, which has a Java wrapper available as well, although it is a little verbose.
- **Component Communication.** There are three types of component-component interaction: REST-based (JSON), SOAP and direct socket connections. All accesses to the database are performed through stubs, which possess a REST enabled web server accepting JSON requests. It is a lightweight protocol and allows for the sending of multiple requests on the same message, including small data transfers. To allow other applications to access the database, a client package is also provided. SOAP is used by the front-end web server component to communicate with the system monitor, providing stateful operations and asynchronous processing. Finally, the 3 core components (monitor, master and slave) communicate through direct socket connections.

5 Evaluation

In this section we present some test results about the performance of the system and briefly describe and try to justify them. All tests were performed under the same conditions (unless specified) on a cluster composed of 6 machines featuring an Intel Core i7-2600K CPU at 3.40GHz, 12GB of RAM at 1333MHz, hard disk with 7200

[11] http://http://opennebula.org/



(a) "Cold" Cloud (b) "Warm" Cloud

Fig. 4: System perfomance under different cloud conditions

RPM and Ubuntu Server 12.04 LTS 64bit. Storage components were already available before every test started, with 3 VMs running HDFS, HBase and an instance of the FaceID-Stub component each. Finally, the component FaceID-Monitor was also already running before every test.

The tests focus only on the detection (i.e. not recognition) of every face on a 3m24s video with 16.9MB, 450x360px resolution and 25fps, and the storage of the face images found on the database. The time was measured from the moment the new job is delivered to the monitor, to the moment the assigned master checks that all slaves have finished their tasks.

"Cold" Cloud. In this scenario, the cloud platform was "cold" (i.e. as in a "cold" cache scenario), meaning that no worker VMs (masters, slaves) were already running and waiting for a job. This implicates the allocation of new resources by the monitor, and thus the performance is affected when more VMs are used, as can be observed in Fig.4a when we use more than 5 slaves. Nonetheless, when using 5 or less we still achieve some speed-up (max around 2.6x).

"Warm" Cloud. Unlike the previous test, the cloud platform was "warm", so all necessary resources were already allocated. As can be observed in Fig.4b, this yields much better results, and the benefit of using extra slaves is maintained for a larger number of nodes (between 8-16), yielding a speedup of 3.9x between 1 and 8 slaves. The reason for the performance decay after 8 is probably due network overheads and database accesses from all components.

5.1 Discussion

From the obtained results, we can observe that two conditions greatly affect the performance of the system: the state of the cloud (i.e. "cold" vs "warm"); and overheads coming from utilizing more nodes. The way we found to deal with the former is to keep the cloud resources from being rapidly de-allocated after they finish a task. Upon finishing, they are transferred to a resource pool in the monitor, where they will wait either for a new job or for de-allocation. We believe the "recycling" of VMs will help the system achieve better results. However, it may be costly to have several VMs on this pool (cloud businesses is usually pay-as-you-go), so they must eventually be de-allocated when the overall system load is reduced. For the other condition, we will use such results to determine the number of slaves to use on each job (between 5 and 8), but we think there are no perfect solutions that can cope with

the diversity of video sizes, resolutions, formats, etc. For example, on larger videos accesses to the database are probably rarer, as the slaves need more time to process each frame, and that may lead to better results with a larger amount of workers. We will approach this in future work.

6 Conclusions

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 with the introduction of a system monitor and master components which aim 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.

Judging by the existing systems performing face recognition in the cloud, we think that our solution approaches a problem that is not yet fully solved, the facial identification on videos. We think our solution is robust and should fit in real-world scenarios on large video databases.

Acknowledgements: This work was partially supported by national funds through FCT Fundao para a Cincia e a Tecnologia, under projects PTDC/EIA-EIA/102250/2008 and PEst-OE/EEI/LA0021/2011.

References

1. 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
2. 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
3. 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
4. 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
5. 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
6. 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