FaceID-Cloud

Face Identification Leveraging Utility and Cloud Computing

Ricardo Caldeira

[1] Instituto Superior Técnico - UTL
[2] INESC-ID Lisboa
`ricardo.caldeira@ist.utl.pt`

Abstract. Detection and identification of human faces has been an intense area of study in the past decades, with many strategies being proposed with some good results. Yet, it is a computationally intensive task, especially in videos that can reach a considerable size. Hence, it is important to develop new solutions to improve the execution time performance of face identification methods. At the moment, one of the most promising technology paradigms 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 offline facial identification performance in large video databases.

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 IT area at a very fast rate, especially 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 fully aware of its power and usefulness for IT businesses and, indirectly, to the end-user. A great number of services typically used by today's average user are now cloud-based (e.g. Amazon online store) and the tendency is for this type of services to grow in number as companies start to acknowledge benefits of cloud computing such as on-demand allocation of resources and utility pay-as-you-go payment models. Given this facilitated access to computing and storage resources provided by emerging cloud platforms, it makes sense to design new cloud-based systems capable of presenting end-users with out-of-the-box solutions that can be used from almost any Internet-capable device.

In this context, offline facial identification and indexing of human faces in videos is a good example of a research area that could greatly benefit from exploiting the cloud potential, especially when dealing with huge databases of videos that can easily reach the terabytes mark. The nature of this problem implies a massive computational effort from the start, since video processing is a computationally heavy task and a face identification algorithm executing on top of it only aggravates this problem. Yet, video processing can be seen as belonging to the class of the generically dubbed ''embarrassingly parallel'' problems, meaning that it is easily divided into sub-problems that should not, considering a sequentially-framed raw video footage, have data dependencies between them. Knowing this, the main focus of research 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 I propose a solution to the problem in the form of the system FaceID-Cloud.

This text will focus on exposing a solution for the problem of implementing a facial identification system on top of a cloud infrastructure, describing the architecture and workflow of the proposed system, the main development steps and some test results to validate the solution.

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 [1] 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. execution 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 [2], and is also based upon some already existing technologies like distributed computing, virtualization or utility computing, which have been around for several years already[3]. 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 [4].
- **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 [2].

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

- *Amazon Web Services.* AWS is one of the most well-known cloud solutions available, providing mainly IaaS and also PaaS with a 99.95% uptime on a public deployment 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.
- *OpenNebula.* An open-source cloud infrastructure which provides a subset of AWS 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.* It is 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 (*i.e.* each node is independent and self-sufficient), multiple network models, distributed scheduler and asynchronous architecture, with several storage options and AWS 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 are representatives of 3 types of strategies to deal with the problem (holistic-based, feature-based and hybrid) and are not to be seen as the state-of-the-art on this area:

- **Eigenfaces.** Proposed by Turk and Pentland [1], 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) and 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). As an holistic approach, it is dependent of environmental variations such as lighting and face orientation.
- **Hidden Markov Model.** Another approach was proposed by Nefian and Hayes [5] 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.
- **View-based Modular Eigenfaces.** There are also systems that use both holistic- and feature-based approaches, like the one proposed by Pentland et al. [6]. In this system, the concept of "eigenfaces" is extended to individual facial features, resulting in "eigenfeatures". The use of both kinds of information (eigenfaces and eigenfeatures) reached a 98% recognition rate on the test database.

Despite several efforts to develop accurate face recognition algorithms, the problem of recognizing faces in large video databases is still not fully solved. It is a subject that comes with an increased complexity in terms of the necessary processing power and network utilization, and thus I think that cloud computing is the ideal solution to this problem, as the problem is highly parallelizable.

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

Before describing the architectural components of the system, it is important to define the architectural drivers which guided the design. To summarize, the system is based on three properties:

1. **Performance** - raw performance in terms of time vs. request size is critical in this system, so it should answer requests as fast as possible given the problem size and available resources.
2. **Scalability** - the system should scale gracefully without major performance loss in terms of number of resources utilized and number of videos to process.
3. **Flexibility** - the system should allow for different types of requests, with different quantities of resources in a seamless way to the end user.

Following these properties, we present a modular vision of the runtime components of the system in Fig. 1, divided into four different areas for easy interpretation and development: Management, Computation, Storage and Web Interaction. 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.

3.1 System Topology

Since there are several different components, with very different types of tasks and interactions, another layer of abstraction is added and the components are grouped according to their type of responsibilities: Management, Computation, Storage and Web Interaction (Fig. 1). This separation is important to clearly identify responsibilities and to achieve loose coupling between the components. A brief description is provided:

Management. The Management area is primarily responsible for accepting and dispatching new requests and managing the system's resources by allocating and deallocating new nodes when needed. Besides this primary aspect, components in this area can also execute system-level tasks, such as cleaning database entries, or training the system according to the face recognition algorithm requirements.

There are two main components in this area, FaceID-Monitor and FaceID-MSlave. The former, the *monitor*, is the brain behind the system and can be seen as the system scheduler, accepting new requests and dispatching them as jobs to the Computation area while maintaining the system state in memory and in the database. The latter component, *mslave*, is a special workforce to be used by the monitor as a worker node when it needs to perform tasks that require more computation, such as training the system.

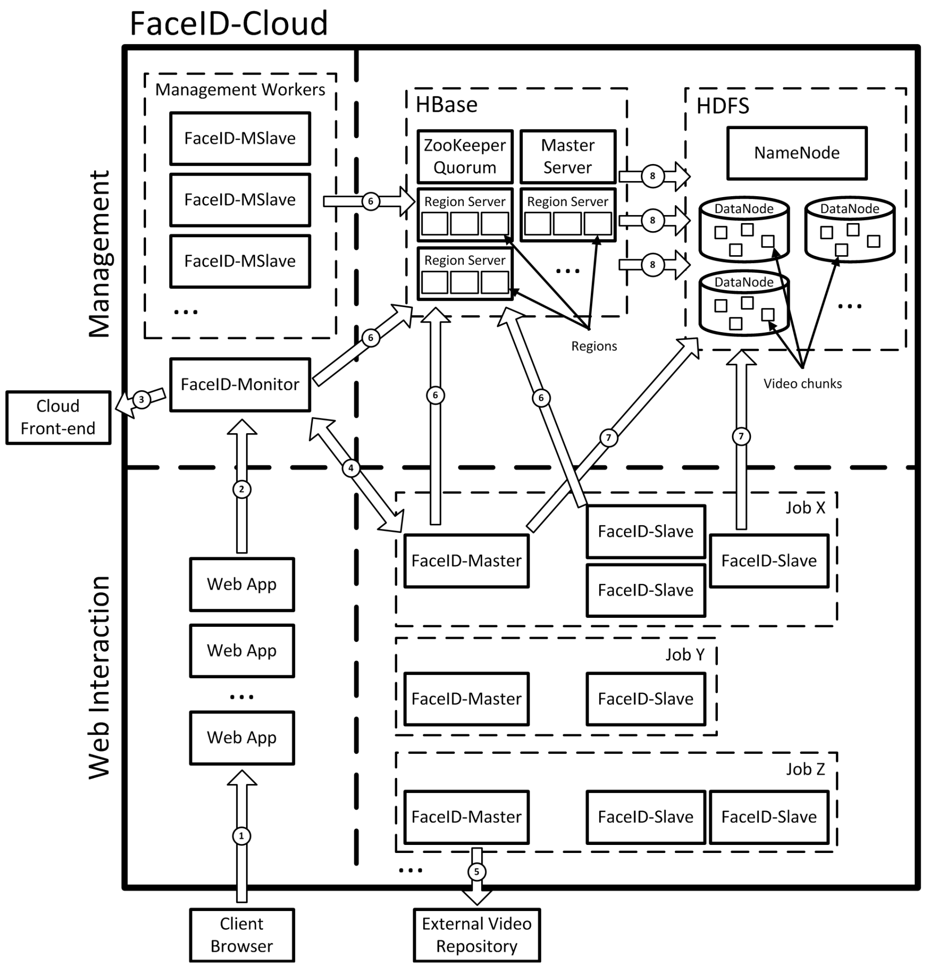


Fig. 1: System Architecture. Interactions: **1.** Client submits new video or query and receives results; **2.** Web server submits new job and gets an assigned master from monitor; **3.** Monitor can allocate and deallocate components through the Cloud Front-end; **4.** Master can ask for more slaves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

Storage. One of the most important areas in the system is data storage, affecting the performance of the whole system. As a data-oriented system, where it is the flow of data that most influences the computation and overall evolution, the way the system data is stored is crucial to achieve both a well-performing and scalable system. Due to the large average size of videos, even reaching the terabytes mark, there will eventually exist network congestions, with all the bandwidth being occupied with large video transfers, so the system should try to minimize large transfers as much as possible.

There are two important types of data flowing through the system: unstructured and structured data. The first type represents video chunks, which are stored as binary files. A large amount of space is needed for this type of data, while maintaining properties like consistency, availability and partition tolerance. To achieve this, the Hadoop Distributed File System (HDFS)[3] is used, supporting very large data and with a simple coherence model of write-once-read-many which fits the needs of our system to store videos. Besides being able to store a large amount of data, the distribution of data among several nodes will increase the performance of the system as several clients can be connected concurrently to different HDFS nodes.

The second type, structured data, is needed for a different purpose, that of storing information the system generates about videos and the current state. The use of a typical relational database is not advised in very large datasets due to inherent scalability problems, so key/value column-based database HBase was chosen to run on top of HDFS, leveraging its properties. Yet, this implies the loss of querying functionality, as HBase delegates that to the client application.

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 Grid Strategy

The face space is a very sparse multi-dimensional space resulting from the dimensionality reduction strategy applied in the eigenfaces algorithm using Principal Component Analysis. Nonetheless, a low number of eigenfaces can still generate a very large number of vector combinations in this space. Also, although the faces of one individual might be very similar, in this new coordinate system they can still be very far apart on some dimensions, hence the sparseness of the face space. It is very hard to compare a new face (its weight pattern in reality) with every single face available on the database, either in terms of time, memory or network bandwidth. Hence, a strategy must be applied in order to reduce the search scope for the classifier.

[3] http://hadoop.apache.org/docs/r1.0.4/hdfs_design.html (accessed May 2013)

Due to the key/value database chosen, which lacks querying functionalities, the solution found to address the large search scope problem in this particular database type is to use a *grid strategy*, which is integrated both on the way the system performs searches and the way the data is stored in the database. Summarily, this involves partitioning sets of weights into cells according to their vector coordinates, by aligning each coordinate to their next hundred, either towards $+\infty$ for positive numbers or $-\infty$ for negative numbers (zero is considered positive). In a sense, a new ''grid coordinate system'' is being built, where each weight can be described by its corresponding grid coordinates. Given a margin in relation to a test face to classify, the system only has to fetch from the database the cells, and the containing weights, which are closer to it in the grid space, effectively reducing the search scope and the cost of a full table scan.

3.4 Training - *Clustering*

As a supervised method, eigenfaces needs a base set of tagged faces to classify new unknown individuals. Yet, the system will possibly find several unknown faces in new videos, which are valuable for future runs. To leverage this information, the system groups all the faces resulting from a job into clusters which will at a later time be manually tagged by an end-user. To perform the clustering of faces an unsupervised learning algorithm must be used, with the limitation of not having a predefined number of clusters to start with, i.e. the number of the clusters should be defined according to the data itself and its density regions. An algorithm capable of this type of clustering is DBSCAN[7], and it is the one used in the system.

3.5 Workflow

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

1. The user contacts the web server component providing a URL for the video he/she wants to be analyzed by the system in search for human faces;
2. The web server contacts the system monitor and submits the information the user has provided, waiting for an response;
3. The monitor starts by checking its state for masters waiting to be de-allocated which have finished their previous jobs. If such a master exists, the new job will be assigned to it. If not, a new master is allocated through the cloud front-end;
4. The job information is sent to the chosen master, that will download the video from the URL provided by the user;
5. The master pre-processes the downloaded video by performing a conversion to a common format and splits it in several smaller chunks, which are uploaded to the datastore;
6. The master estimates how many slaves it needs to process the chunks and requests them to the monitor;
7. The monitor analyzes the system load and the resources available and sends the slaves it can dispense to the master, allocating more slaves if needed;
8. Upon receiving the slaves' information, the master distributes the tasks among them and waits for termination by polling the database;
9. Each slave will execute their assigned tasks by applying the face recognition algorithm in the chunks. When each task is finished, the results are written to the database;

10. When all tasks complete, the master officially finishes the job and sends a response to the monitor. All information is stored in the database for future requests on that video or the people contained in it;
11. The results are propagated to the web server that presents them to the user;
12. Finally, the monitor orders an m-slave to learn about the new faces found and integrate that knowledge into the database.

3.6 Data Model

Since the chosen database is column-oriented, the data model is be very simple and adapted to this type of database, containing seven tables (columns are not shown to simplify):

1. **Jobs** - Holds information about current and past jobs, namely the unique ID of each job, the resources allocated, job state and the results.
2. **Videos** - Holds information about the videos the system has processed, such as their names, URLs from where they were downloaded, the unique video ID attributed to each and video chunks location in the datastore.
3. **People** - Holds information about individuals that were added to the system, specifically their name, a unique person ID and the clusters to which the person is related.
4. **Faces** - Holds every distinct face found by the system, being one of the largest entities.
5. **Eigenfaces Data** - Holds information about the face recognition algorithm, namely the training information and results for each training session.
6. **Grid** - Holds the grid structure data, with information about each cell and their containing faces.
7. **Clusters** - Holds information about all clusters found, namely the mapping a cluster and the grid cells that belong to it.

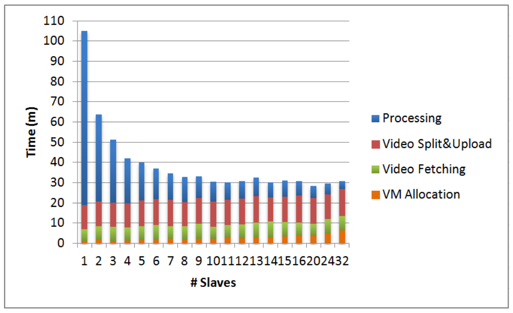
4 Implementation Details

We now give some detailed information on the technologies used to develop the system and some specific issues addressed during the implementation of a prototype system.

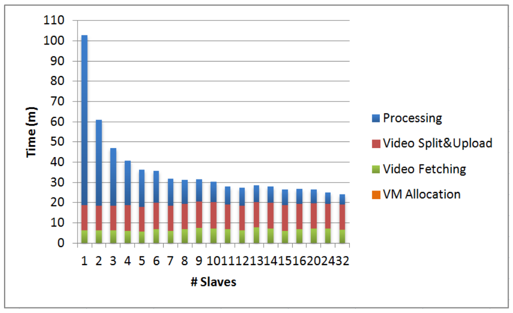
Deployment Environment. All system components were developed in Java, which facilitates the access to the storage components which already have APIs in this language. The virtual machines running the system components are deployed in OpenNebula, a mature and highly customizable open-source cloud platform. To facilitate the development of the face recognition module, the computer vision library OpenCV was used together with a Java wrapper.

ID Generation Throughout the system there is a need to uniquely identify jobs, faces and other entities, hence several counters are stored in the database which are incremented by each client component when needed. To reduce contention, a random back-off strategt is applied and more than one ID is generated at a time in ‘‘packs’’ to reduce database trips.

Denormalization In a relational database, normalization of the relational model is essential to guarantee consistency and mitigate redundancy across the domain. Yet, the lack of query capabilites in HBase needs the redundancy to accelerate queries, leading to a denormalization of the domain.



(a) Cold Cloud



(b) Hot Cloud

Fig. 2: Execution time evolution

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 with Hyper-Threading, 12GB of RAM at 1333MHz, hard disk with 7200 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 video used for testing has a 960x720px resolution and 1.39GB size, featuring a television show with at least four individuals.

This scenario illustrates several different runs of the test video with two variables - the cloud state (cold or warm) and the number of slaves (1 to 32). The difference between ‘‘cold’’ and ‘‘warm’’ states is similar to regular computer caches, where a cold cache has no stored information and will always return a miss, and a warm cache has already some data to return. Making the bridge to our system, a cold state will not have any virtual machines running, while a warm cloud will have all already allocated. As can be seen in Fig.2, a warm cloud will always return better execution times, since there is no overhead due to VM allocation times (orange in the image).

Regarding the number of slaves, Fig. 2 shows that after 4-5 slaves the biggest contribution for the total time is no longer the processing, but the fetching and splitting of videos, this being the reason for the execution time to stabilize at around 25 minutes. Nonetheless, if we only take into account the parallel fraction of the job (blue in the image), the system reaches a speedup close to 16 at 32 slaves, which is compatible with the fact that the processors use hyper-threading (which does not guarantee an equal performance to physical cores), while some communication overheads and database contention make up for the rest of the efficiency loss.

6 Conclusions

Our main goal was to design a system that can horizontally scale in terms of execution time performance and storage utilization 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.

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 Fundação para a Ciência e a Tecnologia, under projects PTDC/EIA-EIA/102250/2008 and PEst-OE/EEI/LA0021/2011.

References

1. Turk, M., Pentland, A.: Face recognition using eigenfaces. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Comput. Sco. Press (1991) 586–591
2. Zhang, Q., Cheng, L., Boutaba, R.: Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications **1**(1) (April 2010) 7–18
3. 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) (December 2008) 50–55
4. Tsai, W.T., Sun, X., Balasooriya, J.: Service-Oriented Cloud Computing Architecture. In: Seventh International Conference on Information Technology, IEEE (2010) 684–689
5. 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. Volume 5., IEEE (1998) 2721–2724
6. 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. Volume 02139., IEEE Comput. Soc. Press (1994) 84–91
7. Ester, M., Kriegel, H.p., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise, AAAI Press (1996) 226–231