# FaceID-Grid: A Grid Platform for Face Detection and Identification in Video Storage

Filipe Rodrigues
filipe.rodrigues@ist.utl.pt

Instituto Superior Técnico

**Abstract.** Face Recognition systems have received significant attention from the research community as well as from the market; its applications are now vast, such as video games, social networking and security. Significant advances have been made in the face recognition field; however face recognition is still a demanding task in the context of videos that, if executed in a single machine, has severe limitations to its potentialities. The goal of this thesis was to develop a face recognition system that is executed in a grid environment to create a distributed, scalable and efficient system. We used existing stand-alone face recognition software from the OpenCV library to support the face recognition functions and the system is executed on top of a Condor grid. We develop the distributed architecture of the system, a file storage schema for the face recognition data, a distributed coordination system for the distributed execution, a resource discovery system tailored for the specific needs of this application and a scheduling algorithm approach for the system

**Keywords:** Grid, Video, Face Recognition, Eigenvectors, Condor, HDFS.

## 1 Introduction

Distributed computing is a successful way of providing high computational power to applications; one of the more common types of distributed computing is grid computing, where essentially the computational power of a number of machines is combined to execute a given application.

High resource demanding applications can benefit from being executed in a grid environment. Traditionally, these applications would have to be executed in super computers or large scale dedicated computer infrastructures which would provide the resources those applications need. However, super computers or dedicated computer infrastructures are not at the financial reach of all organizations; Grid Computing on the other hand, is able to harness the computational power of commodity hardware, and due to the available grid middleware, it makes a relatively affordable way of obtaining high computational capacity; moreover as grid infrastructures are not dedicated to a given application, organizations can share resources. All of these facts, justify the success grid computing has had and is having in the scientific community.

Face Recognition, especially in videos, is an example of a high demanding application that can benefit from grid computing. While, in the literature, we can find a large amount of research done in grid computing, to this date we are not aware of any developed or planned face recognition system that leverages grid computing, despite being relatively evident that such a system can be created. Grid Computing works well with a particular type of parallel computation, the multi-data-single-instruction; which is precisely what a face recognition system is, multiple images to be process always by the same code.

The rest of the article is organizes as follows: we present our objectives in section 2, we first present a brief summary of the related work in Face Recognition and Grid Computing fields in the Related Work section 3, next we present the System Architecture in section 4, followed by some implementation details and the evaluation in sections 5 and 6, at the end we finalize the article with the Conclusions in section 7.

## 2 Objectives

Our goals for this work were to develop a video face recognition system executed in a grid computing environment with the objective of improving its performance. The system is to be able to run in one or more clusters, leveraging existent grid middleware.

Specifically we had the objective of developing a distributed system with better performance than a standalone application, a distributed systems that can scale to a large size, the distributed architecture of the system, incorporating existing stand-alone face recognition software and a scheduling algorithms approach that balances the load across the machines in the cluster.

## 3 Related Work

### 3.1 Grid Computing

Grid Computing [1] is a relatively successful type of distributed computing infrastructure that has the objective of providing dependable, consistent and pervasive access to high-end computer resources, it does this by combining the computational power of a high number of computers, usually for scientific computing and e-science tasks.

Any distributed system has to be controlled, Grids are controlled by a scheduler, which acts as a resource broker. The schedulers function is to interact directly with the Grid user that submits tasks to be executed, select the resources appropriate to the task in question using a predefined algorithm parameterized by information from the Grid resources to which it can assign tasks, and finally assign the task to the selected resource.

In the literature we find that schedulers, can be classified as Static or Dynamic, related with the time at which the scheduling decisions are made [2]. Also the schedulers can be positioned in relation to the resources in three ways, Centralized, Hierarchical and Decentralized [3].

From the nature of the application, performing face recognition in videos arriving at any given time and answering queries to retrieve information about the faces identified on the videos submitted earlier, it is clear that static scheduling would not be appropriate, since no prediction about the amount or nature of the tasks can be made, therefore we chose to use a Dynamic scheduler. A centralized scheduler would be able to provide optimized schedules, but would easily become the bottleneck of the whole system and provide poor fault tolerance; a hierarchical scheduler would mitigate these problems but would not eliminated them; finally a distributed scheduler would provide good fault tolerance and scalability but would more difficultly produce optimal schedules, also it would require sophisticated coordination between schedulers which increases complexity and decreases performance.

For the development of our system we choose Condor [4] for the scheduler, having dynamic scheduling it is appropriate for our purposes. And the fact that it is a centralized scheduler means that it is able to produce optimal schedules; it also means that scalability is limited, however as the execution time of a task is certainly measured in minutes and a scheduling decision time is measured in seconds in a worst case scenario, we are able to classify the bottleneck effect as minimal. Condor only controls one grid site but the system works in more than one, first we try to use a hierarchical scheduler, Condor-G [5] as a central scheduler that would decide in which site a task would be executed and all sites would be used in the same way. However due to the characteristics of the system, specially the fact that its IO demands are relatively high, it is useful to prevent the overhead of transferring data to outside the site. In this scenario, the additional overhead of having a scheduler only to decide which site to send a task is not justified, instead we use a trigger that redirects the tasks to other sites if the main site is overloaded.

### 3.2 Resource Discovery

A scheduler is informed about the resources availability and their detailed characteristics by the Resource Discovery System. The Resource Discovery can be seen as two processes, resource discovery and resource dissemination, which are, an application trying to find resources and a resource advertising its availability respectively [3].

Resource Discovery can be done using queries or agents. These are the Schedulers fetching information from an external directory system which contains all the resource information and sending active code fragments across the network which are interpreted on every reached machine respectively. Query based resource discovery systems can be further classified as centralized, hierarchical and decentralized according to how the information database is accessed.

Resource Dissemination can be classified in batch/periodic and on demand, according to when resources disclose information about their status. In a periodic approach, information is updated in predefined time intervals, while in an on demand approach, updates are done immediately after resource status changes.

Our system can run in multiple grid sites and in each site the chosen scheduler (Condor) controls the machines, Condor is considered a centralized scheduler but is in fact what is truly centralized is the resource brokering and schedulers coordination, in Condor that is done by the ClassAd Matchmaker [6, 7]. Therefore ClassAd was adopted.

ClassAd requires an external system to provide the dynamic information it needs, also the trigger system mentioned in the Grid Scheduling section equally needs resource information. These two modules need information with different levels of abstraction classad needs detailed machine information at different levels of the hierarchy. MDS could provide in such manner, however the resource dissemination in MDS is periodic, which imposes delays, these delays in file placement information would mean that the information would simply be unavailable at scheduling time. We chose to develop an application similar in architecture to MDS, but to which the file placement update can be sent on demand.

### 3.3 Face Recognition

Face Recognition has the objective of identifying or verifying the presence of faces in a given image. It is divided into two main steps, face detection and face recognition, the first one is the isolation of a face in an image, and the second is the recognition it-self. The algorithms in both these steps can be based in features or in pattern recognition.

Feature based algorithms work with explicit face knowledge, for instance face edges, gray levels and eyes, mouth and ears positions; these are more sensible to bad quality images, crowded backgrounds and face position changes. Feature based face detection algorithms detect faces by comparing the positions and characteristics of a number of features in a given object with an average face determined previously form a training set. Face recognition is similar, only they try to determine which faces has the closes characteristics to the one being identified. Haar-like features [8] is an example of a feature based algorithm, is this case for the face detection. These algorithms address the face recognition as a regular pattern recognition problem and do not depend on explicit face knowledge; these are less sensible to bad quality images, crowded backgrounds and face position changes. The detection and recognition of faces are done in a similar manner to the feature based, only in these the images are transformed into a mathematical representation which is latter used to determine de similarities between the known faces and the one being identified. Eigenfaces [9] is an example of a pattern recognition based algorithm.

It was not the objective of this work to develop a novel face recognition algorithm, nor was its actual performance of grate importance, we simply had the objective of improving the performance of an existing face recognition software executing it in a grid environment. Nevertheless, we chose

the software that seemed to perform better in video base face recognition, remembering that face recognition in videos normally means that the system has to work with low quality images, reduced size images and crowded background. We chose Haar-like Features for the face detection process, although it is a feature based algorithm, which theoretically performs worse than a image based algorithm, the fact that it uses a list of rigorously studied and defined rules to perform the detection will assure the same or better results, with the advantage of having better performance and with the additional benefit of not being necessary to train the system before the execution. For the face recognition process we chose Eigenfaces which produce better results in low quality images, common in videos.

## 4 Architecture

In this sectionr, the architecture of the system is detailed, we detail the systems distributed architecture in section 4.1, followed by the software architecture in section 4.2 and the resource broker and Resorce discovery system in section 4.3. Lastly describe a repository search optimization done in this work in sections 4.4.

### 4.1 Distributed Architecture

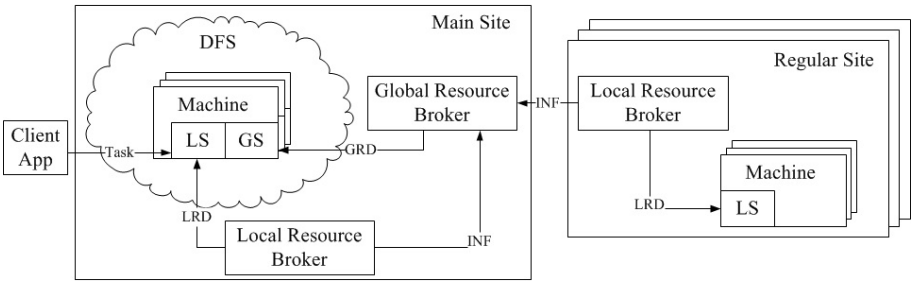


Fig. 1: Distributed Architecture.

Figure 1, shows the systems distributed architecture, the arrows of the resource brokers mean: GBD - global brokering decision, LBD - local brokering decision and Info - information flux between resource brokers.

Two types of cluster exist, the main site and the regular sites. Normally the whole application runs and all persistent data is kept in the main site. Maintaining all data in the main site is justified by the reduction of software and control demands on other clusters (clusters to which we may have limited access). Also as the application is a IO intensive, data transfer are to avoid as much as possible. Since all persistent data is kept in the main site, it makes sense to execute the application always in the main site; however computational power is not infinite, therefore to scale the system, there is the option of sending tasks to other clusters.

The machines on the main site make part of a Distributed File System, where persistent data is stored. Each machine has a Scheduler, a Grid Application and the Recognition Software (not represented in the figure). The Scheduler makes decisions about in which machine of the site to execute tasks, the Grid Application which serves as a gatekeeper for the Client Application and submits tasks to the scheduler, submitting to the local machine scheduler or a scheduler in a machine of another cluster effectively makes the decision of in which cluster to execute tasks. Enabling these decisions are the resource brokers, systems that make the actual decisions.

The machines in the regular sites only have a Scheduler and the Recognition Software as those are just worker machines to which tasks can be sent from the main site. Apart from that they work in similar manner. Note that the architecture described here also include components to control the regular sites, which seems to go against the previous description of not controlling the regular sites, they are in fact placeholders for any other grid control system that can be replaced by adapters to other systems.

### 4.2 Software Architecture

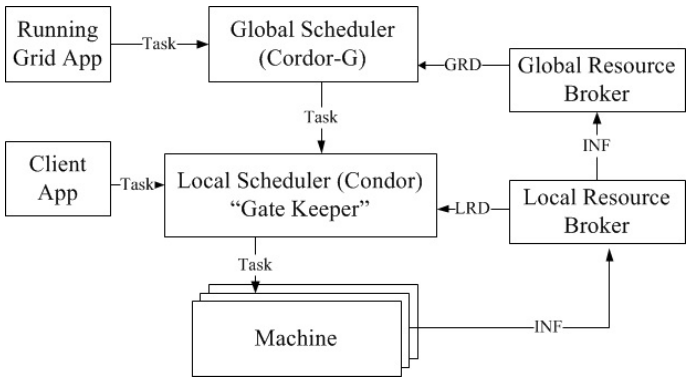


Fig. 2: Software Architecture.

Figure 2 details the system's software architecture, for simplicity the figure only shows components of the main site. The main system is composed by 3 applications. The Client Application interacts with the Grid Application in one of the site's machines; it submits videos to be processed or queries. Each Grid Application submits task to an instance of the Condor Schedule, The scheduler in turn selects a machine and sends the tasks to be executed by the Recognition Software. The Recognition Software, is the component that actually performs the objective of the system also submits tasks to scheduler in the same way the Grid Application does, the reason for this is that the recognition process is divided in a number of steps.

The repositories (video files and intermediate files) are kept in a distributed file system, all communications between the executables are done through the distributed file system except the communications between the Client Application and the Grid Application, these use standard TCP. A traditional DFS keeps complete files in a single machine and transfer those files on demand, this kind of behavior would easily become the bottleneck of the system. Because of this we chose to use HDFS [10], whose standard behavior is to divide big files in smaller chunks and store those chunks in various machines and the chunks themselves are replicated; these characteristics not only allow for good fault tolerance but also for good load balance. In the particular case of this work the segmentation of the videos in chunks was done according to the HDFS file segmentation and the chunk placement information is used for resource brokering.

The Condor Scheduler relies on the ClassAd Matchmaker to make decisions. ClassAd requires an external system to provide dynamic resource information, in this system the application RDS (Resource Discovery System) Producer present in each machine of the cluster gathers information from the local machine, translates it to a ClassAd advertisement and sends it to the matchmaker. These updates are done periodically, which impose delays in changes propagation, while these delays are acceptable in machine status, they are not in file placement, since the delays will mean that information is simply unavailable in contrast with the slightly outdated information of the machine status. The RDS Indexer solves the issue of file placement propagation, when either the Grid Application or the Recognition Software stores new files they trigger the RDS indexer which

will fetch the information from HDFS on demand, this information is returned together with the scheduler selection, the local scheduler or another site's scheduler.

### 4.3 Resource Brokers Architecture

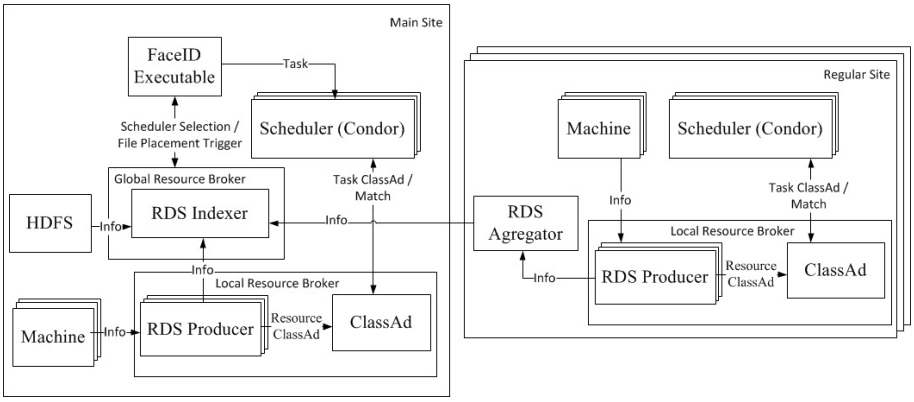


Fig. 3: Resource Brokers Architecture.

In figure 3 we detail the Resource Brokers and Resource Discover System architectures.

The Local Resource Broker decides in which machine of a site to execute a task, it is composed by the ClassAd Matchmaker and the RDS Producer. ClassAd coordinates the schedulers by matching the tasks and resource classAds. The task classAds are defined by the FaceID Executable that submits the tasks and the machine classAd are defined by the RDS Producers. The Global Resource Broker decides in which site to execute a task and is used to introduce file placement information into the schedulers on demand. The site to execute the task is selected using the summarized information from the main and regular sites sent by the RDS Producer. To avoid a flooding of updates from the regular sites an additional component of the RDS is present in the regular sites, the RDS Aggregator, its job is to summarize the sites resource information and send it to the RDS Indexer.

### 4.4 Know Faces Repository Search

Traditional face recognition systems compare a new face against the entire data-base, such a solution require very large file transfers. In order to avoid this problem we developed a way of limiting the search on the repository using the mathematical properties of the vectors (norm and angle), these two values do not vary wildly for images of the same person, in fact we found experimentally that images of the same person on average would vary at most 1000 units in norm and 10 degrees in angle. With this information we stored the vectors according to their norm and angle, in ranges of 1000 units of norm an 10 degrees of angle and we only try to identify a face against the nine groups of vectors adjacent to it, the ones within 1000 units of norm a and 10 degrees limit, figure 4 shows the groups (represented by a T) that are transferred for a given projection.

This limit also works as a threshold for the face recognition, a face that is not within these limit is not considered a match even if it is the closest one.

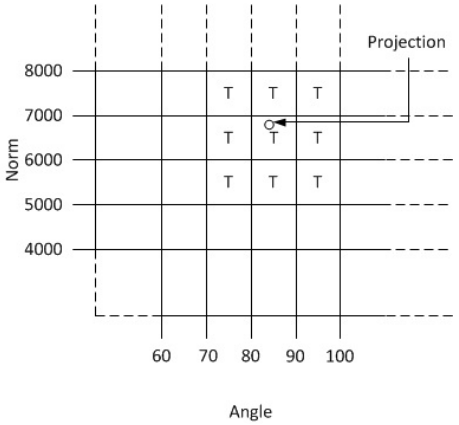


Fig. 4: Know Faces Repository Search.

## 5 Implementation

In this section we give a brief explanation on how the system was implemented followed by a description of the scheduler parameterization in 5.1.

All executables in C++ because the main system dependency, the OpenCV Library [11] was written in C++. The system dependencies extend further from just the OpenCV library, but all other dependencies are independent applications, the interaction with those application was done using the C function int system(const char *command), that executes shell commands, the result output of those command is then parsed by the system executables, although this may not be the ideal interface between applications, it is an effective, systematic and generic method of interacting with any application. The Application them selves are litle more that a list of commands that implement the desired functionality.

### 5.1 Scheduling

The scheduling in the system is defined with classAds that the ClassAd Matchmaker uses to perform the scheduling decisions, these classAd define condition to match machine with task, an use for that machine attributes (Memory, CPU and HDD space) and the file placement information. The memory and hard drive capacity are fixed requirements, meaning that only machines who meet those requirements are considered to run a given task, the CPU available and the file placement information are used to balance the resource usage and to prevent starvation. The system is not required to respond in real time, instead it must be able to process large amount of videos using the available resource as efficiently as possible.

For each task that can be scheduled we wrote a deferent ClassAd advertisement (figures 5 and 6), we present only the Requirements and Rank attributes as these are the one that actually take place in he scheduling

```
Executable   = Face-Detection
Requirements = FACEID_CPU == 2 || FACEID_CPU == 3
Rank         = 2 * (FACEID_CPU == 3) + Machine == <MachineName>
```

Fig. 5: Face Detection ClassAd advertisement

Requirements is a boolean expression and Rank is a numeric expressions. ClassAd will choose the machine in which the Requirements evaluates to true and the Rank evaluates to the highest value.

```
Executable   = Face-Recognition
Requirements = FACEID_MEM == 3 && FACEID_HDC == 3
Rank         = 2 * Machine == <MachineName> + (FACEID_CPU == 1 || FACEID_CPU = 2)
```

Fig. 6: Face Recognition ClassAd advertisement

The values of the machine attributes 1, 2 and 3 describe the availability of resources, the higher the number the higher the availability. The preference for a machine that have the required files locally is expressed by the Machine == <MachineName>.

The Face Detection tasks require mostly CPU therefore the Requirements are "FACEID_CPU == 2 || FACEID_CPU == 3". Within those machines ClassAd gives preference to machines with High CPU and machines that have the required files locally, therefore the Rank is "2 * (FACEIDCPU == 3) + Machine == <MachineName>". The CPU attribute is more important for the scheduling, hence the multiplication by two. The reason for this is that the file transfer overhead is of reduced significance in these tasks. The Face Recognition tasks on the other hand are relatively short tasks that require most of all Memory and Hard Drive Capacity. The overhead from transferring the required files is not neglectable. The Requirements for these task are therefore "FACEID_MEM == 3 && FACEID_HDC == 3" which means that only machine with High Memory and Hard Drive Capacity are considered. When choosing between the considered machines ClassAd gives preference firstly to those that have the required files locally and secondly to machine the CPU med or low CPU. The Rask for these task is "2 * Machine == <MachineName> + (FACEID_CPU == 1 || FACEID_CPU = 2)".

## 6 Evaluation

In this section we present and evaluate the results from tests done to the system. The evaluation is divided in two main sections, the first deals with the speedup achievable in the system and analysis of its scalability.

### 6.1 Speed Up

In the test we use a grid with 5 computers equipped with Intel I7 processor (4 cores, 3.4Ghz), 12Gb of Ram memory and a 1Gb network adapter. A number of videos were used in the tests, both HD (1280x720) and SD (640x360). For the test the video chunk size was 15 MB and each video was divided into 15 chucks. The graph in figure 7 illustrates the average full execution time for both these categories of videos.

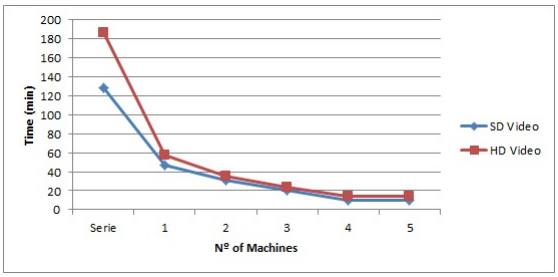


Fig. 7: Complete Execution Time Vs Number of Machines.

The first dot in the graph in figure 7 is the series execution. Using only one machine (8 slots) we can see immediately a significant reduction in execution time, the videos files which were divided

into 15 chucks, had 7 of those chunks being processed at the same time (8th slot is reserved for the Face Detection to prevent starvation). Adding machines 2 and 3 there is additional speed up but it is only when a 4th machine is added that the number of tasks being executed in each machine are bellow the number of real processors and as a result we can observe a very significant increase in speed up.

### 6.2 Scalability

With the test setup we used it was not possible to determine experimentally the scalability limits of the systems, the number of machines was not enough to reach any relevant bottleneck or congestion point that would prevent the system from processing more videos even adding more machines. Still it was possible to get a good idea of the overheads present in the systems, and with that we were able to theorize about its scalability. For this purpose, we increased the number of chunks a video is divided into, by means of reducing their size.

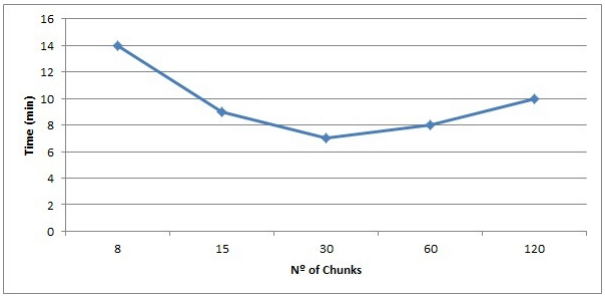


Fig. 8: Complete Video Execution Time.

Although the individual task execution time always diminishes with the chunk size reduction as expected, the complete video execution time does not; thus, this allows us to identify the break-even point that reveals when the overhead of having more short running tasks becomes greater than the gains obtained by the extra parallelism. The overheads are the files transfers and the delays between the task submitting and starting to execute. While the file transfer overheads are reduced with the reduction in size of the chunks the delays in submission and task start only increase with the number of tasks being scheduled.

We can conclude from this information that the system does not scale indefinitely. The graph in figure 8 demonstrates this well, doubling the chunk number produces a reduction of less than 50% while the resource consumption doubles. Therefore it is safe to conclude that the system can in-fact scalable to very large dimensions proving that the chunks are large.

## 7 Conclusions

We now wrap up this article with a brief summary of our work. Our objective in this work was to develop a face recognition system that run on a grid environment, that system would have to be scalable and have significantly better performance than its stand alone counterpart.

We develop a distributed architecture, a distributed resource discovery system and a parameterization for ClassAd to implement our scheduling approach. In addition to that we developed a optimization to improve the search in the know faces repository when identifying a new face.

From the results we got in the evaluation we can say that the objectives were achieved.

## References


1. I. Foster and C. Kesselman. *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann, 1st Editon, ISBN-13: 978-1558604759, 1999.

2. F. Dong and S.G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. *Technical Report 2006-504, School of Computing, Queens University, Kingston, Ontario*, January 2006.

3. K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, 2002.

4. M.J. Litzkow, M. Livny, and M.W. Mutka. Condor-a hunter of idle workstations. In *Proceedeings of the 8th International Conference on Distributed Computing Systems, 1988.*, pages 104–111. IEEE, 1988.

5. D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

6. R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings. The Seventh International Symposium on High Performance Distributed Computing, 1998.*, pages 140–146. IEEE, 1998.

7. N. Coleman, R. Raman, M. Livny, and M. Solomon. Distributed policy management and comprehension with classified advertisements. *University of Wisconsin*, 2003.

8. P. Viola and M.J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

9. M. Agarwal, N. Jain, M. Kumar, and H. Agrawal. Face recognition using eigen faces and artificial neural network. *International Journal of Computer Theory and Engineering*, 2(4):1793–8201, 2010.

10. Dhruba Borthakur. Hdfs architecture, January 03, 2012. URL: http://hadoop.apache.org/common/docs/current/hdfs_design.html.

11. OpenCV. Opencv, January 03, 2012. URL: http://opencv.willowgarage.com/wiki/.