FaceID-Grid

Filipe Rodrigues
filipe.rodrigues@ist.utl.pt

Instituto Superior Técnico
INESC-ID

Abstract. Facial 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 facial recognition field; however facial recognition is still a demanding task in the context of videos that, if executed in a single machine, has severe limitations to its potentialities. In this article we propose the development of facial recognition software to be executed in a grid environment, we made a comprehensive related work analyses in order to develop the architecture of such system, integrating it with existing grid middleware to create a distributed, scalable, efficient and mostly free system

Keywords: Grid, Scheduler, Resource Discovery, Face Recognition, Eigenvectors, Condor, Condor-G, MDS, ClassAd, HDFS.

1 Introduction

Distributed computing is a successful way of proving 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 the 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 software it makes a relatively affordable way of obtaining high computing capacity; moreover as grid infrastructures are not dedicated to a given application so organizations can share resources, All of these facts, justify the success grid computing has had and is having in the scientific community.

Facial 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 facial 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 facial recognition system is, multiple images to be process always by the same code.

The rest of the document is organized as follows: we first state the goal we want to achieve with this work in the Objectives section (Section 2), next we present a study of the relevant literature concerning the Grid Scheduling, Resource Discovery and Facial Recognition in the Related Work section (Section 3), followed by the presentation of our solution design an assessment criteria in sections Architecture and Evaluation (Sections 4 and 5); at the end, we finalize the article in the Conclusions section (Section 6).

2 Objectives

Our goal in this work is to develop the architecture of a facial recognition system for a grid computing environment, running in one or more clusters, leveraging existent grid middleware, to produce a distributed, scalable, efficient and mostly free infra-structure for facial recognition. For the Face recognition process we intend use existing stand-alone face recognition software, adapting it if necessary.

The developed architecture will specify the distributed architecture (schema or machines and grid components in clusters), a file storage schema, a grid scheduling and resource discovery architecture, as well as a novel scheduling approach design specifically for this purpose. In order to do so, we will study the necessary literature, namely grid scheduling, resource discovery and face recognition.

3 Related Work

3.1 Grid Scheduing

Grid computing [1] is a relatively successful type of multiprocessing infrastructure that has the objective of providing dependable, consistent and pervasive access to high-end computer resources [2], 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, and grids are no exception, these are **controlled by a scheduler**, which acts as a resource broker. Schedulers function is simple, they interact directly with the grid user that submits task to be executed, **select the resources** appropriate to the task execution using a predefined algorithm parameterized by information from the **Grid information Service**, another grid component responsible for the task of acquiring information about the grid resources.

In section 3.1.1 a description of some of the particular scheduling problems in grid is given, followed by characterization of scheduling algorithms in sections 3.1.2 and 3.1.3. In section 3.1.4 a description of scheduler architecture is presented, followed by a list of some relevant examples of schedulers in the literature in section 3.1.5. Lastly an analysis of the whole section is done together with selection of the appropriate schedulers for this work in section 3.1.6.

3.1.1 Scheduling Problems in Grids

Other branches of parallel computing like symmetric multi-processor machines or massively parallel processor computers operate under the assumption that schedulers have a single image of the whole system and controls all resources, the resource pool is mostly invariable and the effect of adding new tasks can be contained through some policy [1].

In Grids however, the individual machines may have **different characteristics, different access delays** (connection may be done throw non-uniform links) and may be located in **different domains**. That means that it is very difficult for the scheduler to have a complete image of the grid, since grids in different domains may have local schedulers, which also mean that direct control over all resources is not possible.

Resources in grids are **volatile**, frequently, nodes become unavailable and other became available, also the load of each node is cant be determined, for instance a job scheduled to a certain grid may be interrupted by an internal grid job or by another external job that the local scheduler rated with a higher priority. So the performance of a grid is highly variable when compared with a multi-processor machine for example.

3.1.2 Scheduling Algorithms

Grid algorithms can be classified as Static or Dynamic, related with the time at which the scheduling decisions are made [2], the following sections present a description of both types of schedulers.

Static Scheduling:

Scheduling decisions are made once, each task is assigned to a resource before the application is scheduled and that decision is **never changed** during its execution, information about all resources in the grid, as well as all tasks generated by applications, need to be known at the scheduling time. This type of scheduling have the advantages of working with a **global system view** (information on all tasks and all resources are known at scheduling time), the **tasks cost estimation is simplified** and it has **higher throughput** in general (less computation, less network delay). However it has no **fault tolerance** (if a node fails, application fail) and the **response time may vary** drastically if the selected resource is heavily loaded. These disadvantages may be solved using rescheduling mechanisms, which reduce the performance advantage of these schedulers when compared with dynamic ones. Pegasus [3] is an example of a static scheduler.

Dynamic Scheduling:

Scheduling decisions are made on the fly as application are executed, meaning the attribution of tasks to resources is done as the tasks are presented to be scheduled. It is especially useful when applications have **non-deterministic execution flows**, which is when the number and nature of tasks cannot be determined prior to application execution. These types of schedulers have the advantage **not needing to be aware of application execution flow**, providing **better load balancing**, being **fault tolerant** (task can be rescheduled) and produce **consistent response times** (If the selected resource is heavily loaded, task can be returned to the scheduler). However they have generally a **lower throughput** (more computation, more network delay) and the Scheduling tents to deliver **not the best execution times for each individual task**, it tends instead to maximize resource utilization. Condor [4] and Legion [5] are examples of dynamic schedulers

3.1.3 Scheduling Process

The scheduling algorithm itself is composed by resource selection and task selection; the following sections present a description of the most common algorithms for both components [6].

Resource Selection:

When selecting resources for a given task, schedulers have two lists, one with machines capable of executing the task to be scheduled and another, a subset of the first one with the machines that have enough free resources to execute the task immediately, the work in [6] four selection strategies are presented.

- **Biggest First:** Selects the machine with the most free resources, essentially tries to attribute the task to the machine that will be able to execute it with the higher performance, a disadvantage of this is that very demanding tasks may be delayed due to powerful machines being inundated with small tasks.
- **Random:** Selects a machine from one of the list at random, which on average provides a fair and balanced way to distribute resources to tasks.
- **Best Fit:** Selects the machine from one of the list that if assigned with the task will leave the least amount of free resources, essentially this strategy will try to fully utilize machines, attributing small tasks to machines with few free resources. When compared with Biggest First, that will assign tasks to the machine with more free resources, this strategy is less likely to impose delays on demanding tasks.
- **Equal Util:** Selects the machine with the lowest resource utilization, with the objective of promoting load balancing on all machines.

It is important to note that the list of techniques above are general purpose strategies that work in scenarios where the only aspect taken into account is whether or not a machine has the necessary resources to execute a task, other strategies may involve other factors, such as the cost of executing tasks in a given machine, schedulers like Nimrod/G [7] select machines to assign tasks depending on the cost of using them.

Task Selection:

When selecting tasks to be scheduled the most common strategies are based on list-scheduling, the work in [8] describes four of these strategies.

- **First-Come-First-Served:** Tasks are ordered by arrival time, a greedy list scheduling is used, meaning that the older task is scheduled when enough resources become available, the rest of tasks are not scheduled as long as the older tasks are not. This strategy has the advantage of producing a fair scheduling, not leaving behind or privileging any task, but has the disadvantage of causing a large percentage of machines to be idling for a long time if, for instance, the older task is high resource demanding and the rest of the tasks are not; they could be assigned to machines capable of running them but not the older one.
- **Random:** Tasks to be scheduled are selected at random, has the same issues as First-Come-First-Served, as high resource demanding tasks can cause large periods of idle time in machines, also it may cause starvation, selecting tasks randomly under heavy load there is no guarantee that each task will ever be selected.

- **Backfilling:** Similar to First-Come-First-Served with the difference that when a high resource demanding job is not scheduled, the scheduler tries to prevent idle time on machines by scheduling other jobs for which there are machines with enough resources. To prevent starvation of high demanding tasks, the scheduler requires the specification of a time estimate for the execution of each task, it will only assign a task to resources that will finish before that resource could be used to execute the high demanding task. For instance, if the older task is waiting for resources, a machine with enough resources is running another task that will finish in five minutes, the scheduler will not assign a low resource demanding task to that machine if its predicted to last for six minutes, otherwise it could cause starvation to high demanding tasks [9].
- **SMART:** Also requires execution times, but does not respect task submission order, it groups tasks according to their execution time, each group ceiling it defined by a geometry sequence based on a parameter, which can be used to tune load balance, all tasks in a group are assign to shelves, essentially tasks are grouped so that all can run in parallel, after this, the shelves are ordered using Smiths rule [10] (sum of all execution times in a shelf divided by the higher execution time of any task and the shelves with the highest values are scheduled first.

3.1.4 Scheduler Architecture

Schedulers can control grids and be positioned in relation to the resources in three ways, Centralized, Hierarchical and Decentralized [11], the following sections present descriptions of these scheduler Architectures.

Centralized:

In this Architecture, **only one scheduler** exists, it is responsible for assigning all tasks to all resources. Tasks submitted that cannot be scheduled immediately stay in a queue in the scheduler until enough resources are free. All information required by the scheduling algorithm must be kept by the scheduler (it is not required that the scheduler stores the information but at scheduling time the information has to be transmitted if stored elsewhere). This type of schedulers does not scale well as they can become a **bottleneck**, also it has **no fault tolerance**, if scheduler or its connection to the network fail system availability is compromised. But they are theoretically able to produce **better schedulings** since the have all information and control all resources. Condor [20] is a classical example of such schedulers.

Hierarchical:

More than one scheduler is present in a hierarchical architecture, tasks are submitted to a central scheduler, but that scheduler does not assign task to resources directly; instead, it **submits tasks to schedulers down the hierarchy**, until the task reaches a **local scheduler that will actually schedule the task to a machine**. In these schedulers, detailed information is kept by the lower hierarchy schedulers and the above ones have a summary of the resources available. This organization **mitigates the issue of scalability** since each scheduler has to make decisions about a smaller set of resources. textbfFault tolerance is partially provided, as long as the central scheduler does not fail, operability of the system is assured and the central one can have backup replicas. Legion [5] in an example of a hierarchical scheduler.

Decentralized:

In a decentralized architecture, **no central scheduler** is present, instead a number of schedulers assign task to resources relatively independently. This architecture does not have the bottleneck cause by a central scheduler, therefore is scalable, fault tolerant since only the failure of all schedulers will imply a system failure. The disadvantage of this type of schedulers is that each scheduler **is not able to produce an optimal assignment of task to resources**, since all information is not delivered to all of them; coordination in resource discovery and allocation between them may improve the optimality of the scheduling but it will impose overhead and reduce the performance. MetaComputing Online [12] is an example of this type of scheduler.

3.1.5 Relevant Examples of Schedulers in the Literature

In this Section, examples of grid resource management system (RMS) are presented together with a description of the respective schedulers and their classification according to the definitions made previously.

Globus:

Globus [13] is a grid resources toolkit, it does not have a particular interesting scheduler, in fact it is design to have external schedulers, for instance Nimrod and Condor; there are versions of both of these schedulers integrated with Globus, Nimrod/G and Condor-G respectively. It is however an interesting system when it comes to support distributed execution across multiple domains. Globus has a number of components that can be used to build a grid network, for instance security, resource location, resource reservation, data transfer and remote execution calls.

Condor:

Condor [4] is a RMS originally designed to leverage computational power of idle workstations and make it available to those who have higher computation needs than a single workstation can provide, but it can be used in a conventional grid, the only difference is the idle monitor is not present.

Condors scheduler is classified in the literature as **centralized**, but in reality it **works more like a decentralized or hierarchic one**, there is a **central coordinato**r and a **local scheduler** on each machine, local schedulers are the ones that do the actual scheduling, to avoid a large overhead of messages, local schedulers only communicate resource availability and need to the central coordinator that make the decisions to attribute resources available in a given machine to a another machine that requires them.

Condor ensures proper distribution of resources to all users giving higher priority to machines that require resources less often, this is done be associating a scheduling index to each machine, every time a machine is granted the resources from another, its index is incremented, every time a machine requests are not granted resources, the index is decreased. In each scheduling iteration, the coordinator attributes the available resources to the machines with the lowest index.

Condor-G:

Condor-G [14] is a RMS designed to allow users to **leverage computational resources in heterogeneous domains** as if they all belong to the same domain; it is basically constituted by the original **Condor scheduler** and the **grid resource toolkit Globus** [13]. Its objective is achieved via the separation of concerns between remote execution and computation resources management; remote execution is handled by the Globus toolkit and computation management is handled by Condor Scheduler. In detail, Condor-G uses GASS [15] (Global Access to Secure Storage) to implement the task I/O and GRAM [16] (Grid Resource Allocation Management) to submit and monitor task in remote machine, both GASS and GRAM are components of Globus.

Condor-G works similarly to condor as far as scheduler architecture is concerned, so it is officially a centralized scheduler but where the actual scheduling is done by distributed schedulers. To each user that submits tasks to be executed in the grid, a grid manager will be created, these can be seen as the local schedulers present in the original Condor and these are the ones that will actually schedule task to be executed in the machines controlled by the Globus tool-kit and also similarly to the original Condor a central coordinator (here called resource broker) assigns resources to each grid manager.

Pegasus:

Pegasus [3] is a RMS designed to execute application over various heterogeneous computer grid sites. It takes abstract workflows describing application and finds the appropriate data and grid resources to execute them.

Workflows are direct acyclic graphs whose elements are activities and the connectors are dependencies between activities; applications are described by workflows, where activities represent individual application components (software components or logical execution steps) and the dependencies represent control and data flow between those components. These workflows are described in an abstract form making no reference to the grid resources that will be used to execute them.

Pegasus has a **centralized architecture**, users define abstract workflows prior to execution and the system builds concrete workflows containing the association between the activities and grid resources, the latter workflows will ultimately be executed; it is therefore a **static scheduler**, since the scheduling decision are made for the whole application and prior to application deployment. Pegasus uses three criteria for scheduling: file placement, code placement and computational resources, it uses tree different Information System to acquire this information.

Pegasus does not control the actual execution of a workflow, instead it uses DAGMan [17] which is part of the Condor project, DAGMan parameterizes the Condor scheduler to execute the workflow.

Pegasus is also available with a **dynamic scheduler**, it works in a similar way, but instead of generating a complete workflow all at once, the workflow is partitioned and scheduled dynamically.

Legion:

Legion [5] is an object based RMS whose scheduler has a hierarchical architecture; every component is an object, these objects are organized in a hierarchy and are created and managed by the corresponding class and metaclass.

The organization of the scheduler is hierarchical since the scheduling decision is not made by a single scheduler or by a variable number of seemly independent ones, the scheduling is done at various levels throughout the hierarchy, and the various essential components are presented next:

- **Host and Vault:** Lower rank objects in the hierarchy, Host interact directly with machines, collect the machines' status information and are ultimately responsible for deciding the attribution of a task to machines. Vaults represent tasks, they contain all the information about a task, and they are used for task migration or recovery.
- **Tasks:** Representation of task a client requested to be executed in the grid.
- **Scheduler:** Responsible for creating the mapping between task objects and host objects, effectively assigning tasks to resources, it creates not one but a number of possible mappings.
- **Enactor:** Negotiator that tries to execute one of the Scheduler's mapings, it does this by requesting reservations to Host that as stated before have the ultimate responsibility of scheduling tasks, a given schedules is only executed if its entire mapping of task to Host is possible to execute.

MetaComputing Online:

MetaComputing Online [12] is a RMS designed to provide access to grid resources in heterogeneous sites, it differentiates itself form other RMSs, by allowing resources not only to be seen as hardware able to execute tasks, but also as services that perform tasks.

The scheduler has a **decentralized architecture**, a collection of machines is abstracted by a Central Node, normally only one per institution. These modules encompass all the information about the resources they abstract, and act as gate keepers in order to allow or deny access to the actual resources. Information must be exchanged among these modules. For this purpose, the system uses a transactional based protocol over **shared memory**, shared objects and backups are kept in every Central Node.

Being a fully decentralized scheduler, it is fault tolerant, the failure of one module does not affect the access to any others, this also means it is a very flexible system, since modules can leave and join at any time.

Summary Table:

Scheduler	**Scheduling Algorithm**	**Scheduler Architecture**
Condor	Dynamic	Hybrid Centralized / Decentralized
Condor-G	Dynamic	Hybrid Centralized / Decentralized
Pegasus	Static / Dynamic	Centralized
Legion	Dynamic	Hierarchical
MetaComputingOnline	Dynamic	Decentralized

Table 1: Summary of Schedulers Characterization

3.1.6 Analysis

There is no perfect scheduler for all types of applications; all the alternatives presented in the previous sections have advantages and disadvantages, it is therefore important to understand the problem this work will try to solve in order to make the correct selection of the scheduler.

From the nature of the application, to performing facial 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 of tasks can be made, not even about the nature of those tasks, for instance it is not possible to know the length of the videos that will be submitted, therefore **dynamic scheduling** will be adopted since static scheduling is not adequate to unpredictable scenarios.

The the selection of the scheduler architecture is not straightforward. 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, and would require sophisticated coordination between schedulers which increases complexity and decreases performance.

Also important is the location of the resources concerning the owner and the access policy to the resources, for simplicity we will assume that machines that can be used to execute tasks can only exist inside clusters. In this scenario we propose the use of the Condor-G scheduler, it is adequate for the purposes of this work because of its scheduling of task to external grid sites, using modules form Globus toolkit to support that execution. However Condor-G wont be applied without modifications, distinction between the main cluster and the remaining will be made, so that the initial task, video reception, partitioning and control of distribution will always be done in the main cluster, for all other cases the main cluster will be treated in the same manner as all the others, effectively a **hierarchical scheduler** will be implemented, each cluster will have a local scheduler (most common scenario) which will ultimately schedule tasks for the machine it controls, in addition a top level scheduler (Condor-G) that will submit task to the local schedulers will be present in the main cluster. Both schedulers will need to work accordingly to data location, when scheduling a task that will use a file already used before and cached in a given machine, the top level scheduler will give preference to machines in a cluster where the file is located (even if the task is allocated to a machine that doesnt have it cached, file transfers within the cluster are normally much faster that between clusters), and the local scheduler will also take into consideration the file location when assign a machine to execute the task.

3.2 Resource Discovery

Resource discovery is a vital component for distributed systems, in a grid a scheduler would not be able to work if not informed about the resources availability and their detailed characteristics, static information (e.g. CPU architecture and OS version) and dynamic information (e.g. CPU load, free memory). Resource discovery systems will enable one of the core objectives of a grid, create associations between tasks that need resources to be executed, and machines which have those resources and are able to execute tasks.

In section 3.2.1, resource discovery systems (RDS) are characterized; next, in section 3.2.2, a list of some relevant examples of resource discovery systems in the literature is presented, lastly an analysis of the whole section is done together with selection of the appropriate resource discovery technology for this work in section 3.2.3.

3.2.1 Resource Discovery Process

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. The work in [11] provides an appropriate classification of both processes:

Resource discovery:

Can be done using **queries** or **agents**; the major difference between these two is where resource discovery process takes place. While in resource discovery based in queries, a given scheduler that needs to obtain information, will ask an external directory system which contains all the resource information; in agent based resource discovery machines exchange information by sending active code fragments across the network which are interpreted on every reached machine, agent approach is used in systems where all nodes have global information and/or a directory service is not present. Query based resource discovery systems can be further classified as centralized, hierarchical and decentralized according to how the information database is accessed, the advantages and disadvantages of these different classifications are similar to the ones presented in the description of schedulers architecture. MDS [18] is an example resource discovery system based in queries and Bond [19] is an example of a RMS whose resource discovery system is based in agents.

Resource Dissemination:

Can be classified in two categories, **batch/periodic** and **on demand** according to when the resources disclose information about their status. In a periodic approach, information is sent in predefined time intervals, that way the resource information is maintained softly consistent (changes in availability of resources are not immediately propagated to the resources discovery system), but the system used with a relatively low amount of communication. In an on demand approach, updates are done immediately after resource status changes; the information is always consistent but it has a relatively high overhead in communication. MDS is an example of a RDS m with periodic resource dissemination and R-GMA [20] is an example of a RDS with on-demand resource dissemination.

3.2.2 Relevant Examples of Resource Discovery Systems in the Literature

In this section, examples of resource discovery systems are presented and classified according to the definition made above.

Bond:

Bond [19] is an agent based RMS. Its resource discovery system, also based in agents, works without the need for a directory service. Agents have a global view of the resources available, which is kept updated by periodically (**periodic resource dissemination**) sending active code fragments across the network; for this purpose Bond agents use the **knowledge querying and manipulation language** (KQML) [21]. Bond standard is to work in a flat organization, no hierarchical levels or multiple grid sites are supported.

R-GMA:

R-GMA [20] is an implementation of the Grid Monitoring Architecture proposed in [22], supported by a relational data base. This system defines three types of entities in a grid, **producers** (sensors that collect information about resources), **consumers** (user of the information collected by the producers, e.g. resource brokers) and a **directory service** (allows consumers to be aware of producers existence and the information they collect).

The system has a **centralized architecture**, with the directory service as the central node; producers register their identity and information on the relational database of the directory service and consumers query this database. At this point, it is important to refer that R-GMA manages different types of information in different ways. Static information or information that does not change frequently is stored in the centralized directory service and is used in consumers lookups, and dynamic information is obtained by the consumers directly from the producers, after their identification is provided by the directory service. This separation is not rigid as it could impair the capability of resource discovery, dynamic information vital to producer identification can be stored in the central data base.

MDS:

MDS described in [18] and [23], is a grid **resource discovery and monitoring system** based in queries, it is part of the Globus toolkit. Grid components and their detailed information is represented by **objects composed of type-value pairs**, these objects are organized and stored in a **hierarchical structure** called Directory Information Tree, this structure translates the relations between the various components, for instance a machine that has a network card that connects the machine to a local network is represented by three objects and the respective associations.

The system structure has a **hierarchical architecture**, composed of two types of modules, **producers** and **republishers**. Producers can be seen as the lowest hierarchical workers, they collect information directly from the computational resources via scripts or modules using well defined APIs, these modules compose the **Grid Resource Information System** (GRIS). Republishers are the higher hierarchical workers, they collect information from producers or from republishers lower in the hierarchy, these modules compose the **Grid Index Information Service** (GIIS), and their function is to preprocess resource information in a useful and customizable manner; for instance one republisher can index resources by their operative system or by their available memory while other can index the resources following a completely different schema. This means that specialized republishers can be used in resource brokering, easing the scheduling task, also republishers may hide details to the next one in the hierarchy making this a very versatile system.

Communication between modules is done using two protocols, **Grid Information Protocol** (GRIP) used by republishers to address queries to producers or other republishers and **Grid Registration Protocol** (GRRP) used by producer and republisher to register thenselves in republishers.

The system is based on **periodic resource dissemination**, this types of systems are characterized by an implicit delay between resource status change and the grid user becoming aware of that change, this delay is particularly significant when the system is composed by many levels of republishers; to diminish this limitation, users are not limited to top level republishes, information can be obtained from any module on the hierarchy, even directly from producers.

ClassAd:

ClassAd [24], is a **resource discovery system** based on **matchmaking**, it is part of the Condor system; it is used to make brokering decisions. One of the most distinctive features of ClassAd is that both provider and requesters of resources have a standard and straight forward way of **expressing restrictions** to the match between requesters and providers, unlike most systems where providers only advertise their characteristics and requesters ask the resource discovery system to find an appropriate provider.

The system has a **centralized architecture**, providers send classAds messages to a matchmaker describing their characteristics and restrictions; requesters also send classAds messages to the matchmaker describing a task waiting to be scheduled for execution, both have similar structure and are composed by a variable number of attribute names and their respective expressions, the matchmaker may not have any standard rules for the matching algorithm. Instead, the matching rules are defined in two special attributes contained in the classAds, the Constraint attribute and the Rank attribute, the first translates the compatibility between requesters and providers, and the second one is used to choose a matched pair when more than one match occurs, these attributes are in fact functions, that can be defined using the other attributes (e.g. other.ram>512 translates the condition that a task requires a machine with at least 512 MB of ram) but it can contain arbitrary functions that may be used to add expressiveness to the matching process (e.g. select resources based on file placement[25]).

ClassAd does not define a resource dissemination policy, it can work with both periodic and on-demand approaches, however when used in Condor a **periodic resource dissemination** is used. The system itself works periodically, classAds are received, stored and periodically the matchmaking algorithm will run, produce the matches and inform both the requester and provider, the actions resulting from that information are not of the concern of the resource discovery system.

D-dimensional DHT:

The work in [26] presents a **decentralized resource discovery system**, the system is based on distributed hash tables; it was developed to address the limitation of centralized and hierarchical systems, limited scalability, reduced fault-tolerance and poor network load distribution, it is especially useful for interconnection of large numbers of grid sites where systems with non-decentralized architectures would not be able to operate.

In computational environments composed by a large number of clusters, normally each cluster has a **gate-keeper** or controlling node, each one of these nodes collects information about the resources within its grid site and performs resource discovery in the other grid sites. That means that these nodes need to process two types of queries among each other, **Request Lookup Query (RLQ)** and **Request Update Query (RUQ)**, not having a centralized service means that such queries must be processed in a decentralized manner, information must be shared between all controlling nodes which traditionally would mean a high volume of update messages so that all could process the lookup queries correctly. Instead, a **distributed hash table** is used to maintain the resource information and to support the answering of queries, DHT naturally provide associations between keys and values, in this application keys are specific attributes (e.g. amount of ram) and the values are the machines that have those attributes.

Although DHTs address the limitations of centralized hierarchical systems, such as poor scalability, they are limited to **single pair associations**, for instance a DHT would only hold information about the amount of ram on each machine and it would only be possible to retrieve the information about which machines have a given amount of ram; that way, in a normal query where constraints may be imposed on various attributes, the system would have to retrieve information from various DHT, reducing the performance drastically. The solution for this problem is the use of **d-dimensional indexing**, where the index used in the DHT is a concatenation of all indexes. Another limitation remains however, queries are normally done specifying intervals such as machines with a given resource equal or superior to a certain amount, this would require that all nodes containing information about machines with resources within that interval to be inquired, which would also decrease performance. To minimize this limitation the indexing of the DHT is actually done by range of index values, so that fewer machines have to be contacted when the system processes a range query.

Summary Table:

RDS	**Resource Discovery**	**Resource Dissemination**
Bond	Agents	Periodic
R-GMA	Centralized Queries	Hybrid Periodic / On Demand
MDS	Hierarchical Queries	Periodic
ClassAd	Centralized Queries	Any (in Condor is Periodic)
D-Dimensional DHT	Decentralized Queries	Any

Table 2: Summary of RDS Characterization

3.2.3 Analysis

In the Grid Scheduling section the use of an hierarchical scheduler was proposed, composed by a central scheduler (Condor-G) and local schedulers (for simplicity we will assume the local schedulers are Condor). Resource information will need to flow from the resources themselves to a local grid level and them to the centralized scheduler level, given this scenario, **MDS** will be adopted since it is a **hierarchical information system**, which will naturally allow for different views of the information to the local schedulers and to the central scheduler. MDS also allows the representation of arbitrary entities and associations between them; therefore it will be used to represent files cached in machines.

Both Condor and Condor-G are considered centralized schedulers but in fact are composed by multiple schedulers, one for each machine in the case of Condor and one for each user in the case of Condor-G. What is truly centralized is the **resource brokering** and **schedulers coordination**, in both systems this is done by the **ClassAd Matchmaking**. Therefore ClassAd will be adopted, it will be present on each level of the hierarchy.

Resource information will be obtained from MDS GIIS at that level by translating the indexed information to ClassAd resources queries. The ClassAd attributes Constraints and Rank will allow the scheduling policy to be described.

3.3 Face Recognition

Face Recognition Technology has the objective of identify faces in a given image or verify if a certain face is present. It has received significant attention from the research community, significant advances have been achieved already and it has a large number of applications such as entertainment (video games and virtual reality), security (parental control and building access) and law enforcement and surveillance (suspect identification) [27].

Face Recognition Is divided into 3 main steps [27]:

1. Face Detection
2. Feature Extraction and Normalization.
3. Identification or Verification.

Note that step 3 is commonly know as Face Recognition, however in the interest of clarity we will refer to it as Identification or Verification.

Steps one and two may be done simultaneously, both detection and identification techniques can be divided into two categories, featured based and image based, the following sections 3.3.1 and 3.3.2 explain these in detail, specific techniques and algorithms for video based face recognition are described in section 3.3.3. Lastly, an analysis of the whole section is done together with selection of the appropriate techniques for this work in section 3.3.4.

3.3.1 Face Detection / Feature Extraction

It is the first step in the Face Recognition Processes, it takes the raw input image and outputs a smaller image containing a face present in the inputted image, concurrently, feature extraction and image normalization (size-wise is the more common normalization) can occur since the localization process either relies on feature detection, or in the case of some image-based techniques, feature extraction is not done. This is a vital step for the performance and feasibility of the following steps as the recognition algorithms will not performed well if faces are not accurately detected [27]. Face localization techniques are divided into two great areas [28]:

Feature Based:

This makes use of explicit face knowledge, such as skin color and face geometry, these are lower level methods. These may be composed by three increasingly sophisticated methods that complement each other.

The first one is a **low level analysis**, based on edges, gray levels, color and symmetry. These were the first techniques used in face localization, essentially they try to find lines and angles that may belong to a face or/and try to match clustered colored pixels with skin color. The work in [29] is an example of such technique, in this case based only on edges and gray levels. From the previous description it is evident that low level analysis is prone to a large number of incorrect face detection, objects in the background can have similar color and lines characteristics to a face.

A second method is used to filter the results from the first one, here two techniques can be applied. **Feature searching** which the work in [30] uses, tries to validate features by their size and relative position, for instance a pair of dark areas (eyes) at a certain distance may indicate the existence of a face, distance between the eyes and mouth may also be used to validate a face detected. However this technique is too rigid and based on heuristic information, therefore it may fail if presented with face pose variations and complex backgrounds. **Constellation analysis** used in the word in [31] tries to do the same as feature searching but resorting to a probabilistic model for the special arrangement of facial features, this technique can tolerate missing features and relatively high face rotations.

The most sophisticated of these algorithms that try to validate features is based on **active shapes** that take the form of image features, **snakes** [32] is an example of these algorithms; it uses an energy-minimizing function as guidance to adjust itself toward lines and edges assuming their form.

Image Based:

Makes use of pattern recognition theory, and addresses face recognition as a regular pattern recognition problem, the image is analyzed as a whole and the face knowledge is not explicit but is contained in the training schemas for this kind of techniques. These Techniques have a clear advantage over the featured based ones, they do not depend on explicit face knowledge, eliminating the potential errors or incompletions of knowledge; moreover it has better performance in multiple faces detection and clusters background situations. These can be divided into three types.

One of them is **linear subspace methods** [33], which makes use of eigenvectors in face representation. The work in [33] introduced a technique to detect faces using this representation, from a training set, a profile eigenface is built; this result is caledl the "face space". Face classification is then achieved using the distance-from-face-space, an error resulting from the calculation of the eigenvector that is used to determine a given image faceness. Another method is **neural networks**, which is largely used in pattern recognition problems) can also be used for face detection, the work in [34] is an example of such system. Finally there are statistical approaches based in information theory like Bayes decision rules [35].

3.3.2 Identification or verification

This is the second step in the Face Recognition Process, here face images outputted by the first step are analyzed and identified, similarly to face detection the identification also has feature and image based techniques, in addition it has Hybrid Methods.

Feature Based:

As previously stated, feature based algorithms make use of explicit face knowledge, most commonly eyes, nose and mouth position and form, are used to perform the recognition [27]. Examples of algorithms in this category are [36] and [37], in the last one features are represented as nodes in a graph and the face identity is represented by the Euclidian distance between the nodes, the recognition is then performed by graph matching.

Image Based:

Similarly to face detection, Image based algorithms treat face identification as general face recognition problems, process the image as a whole, not giving any particular relevance to local features such as eyes and mouth, also they have a high resilience to partially occulted faces and clouded background situations. The work in [38] and [33] are an example of such systems, as described before an image is transformed in eigenvectors, values that represent the identity of a face, face recognition is then done by compilation of the values of a face to identify with the values of know faces.

Hybrid Methods:

Humans use both feature and all image based techniques when recognizing faces [27], so a natural evolution of recognition systems would be to adopt both techniques, improving therefore their performance, in the work discribed in [39] for instance, eigenvectors are used for the global image as in a normal image based algorithm, but they are complemented by eigenfeatures which are the application of eigenvectors to local features like eyes and mouth. In addition, they also make use of various images for the same person, with different angles and expressions, that allows for better recognition rates and improved resilience to variations in different images of the same person.

3.3.3 Video Face Recognition

The previous section describes face recognition methods for still images, but video face recognition has a few particularities worth mentioning.

Challenges of video face Recognition:

The face images extracted from a video sequence are often of much **lower quality** when compared with the still images normally used for face recognition, video acquisition may happen outdoors or even indoors with bad conditions for video capture; illumination, visibility and the angles or poses of captured subjects may not be the ideal. Also, face images are **smal**l, video face regions may be as small as 15 x 15 pixels in contrast with 128 x 128 pixels of a still image [27], the reduced dimensions will difficult the Face Detection and therefore impair the Identification which depends on accurate face detection.

Face Detection in video sequences:

The same techniques used for still image face localization can be applied to video sequences; each frame can be treated as a still image and the techniques may be applied directly, however its possible to make use of motion to indicate the presence of faces. The work in [40] makes use of **frame differentiation** to detect faces and other body parts, it does this by calculating the differences between a number of frames and determining the areas where accumulated differences reach a certain value (threshold). The work in [41] uses the same technique to locate and extract face feature like mouth, nose and eyes.

Identification or Verification in video sequences:

Again, the same techniques used for still image face recognition can be applied to video sequences, but the fact that many similar images of a face are present (large number of frames contain similar images of a subject's face), allows for improvements over these methods, the problem of partial **face occlusion** may be mitigated by synthesizing a virtual frontal view from the large amount of different posed and angles present in the video frames. Another technique that improves face recognition is representing an identity not only by one image but by many, here **voting based recognition** is possible [27], where essentially an image to be identified is not only compared with one already identified image but with many images from the same person, each with a different pose, angle or illumination, that way improving the accuracy of the Identification.

3.3.4 Analyses

On the previous sections, from all algorithms and techniques described, there is one that is notoriously better for video based recognition, which is the **image based** techniques **eigenfaces**.

The reason to choose image base techniques over feature based is its superior performance in common conditions for video face recognition: low quality images, reduced size images and crowded background in face detection as well as in Identification or Verification. Its not the objective of this work to develop a novel face recognition algorithm, therefore more complex techniques such as Hybrid Methods will not be used, although they have been shown to be superior than simple eigenfaces, they will not be used in this work because of their extra complexity.

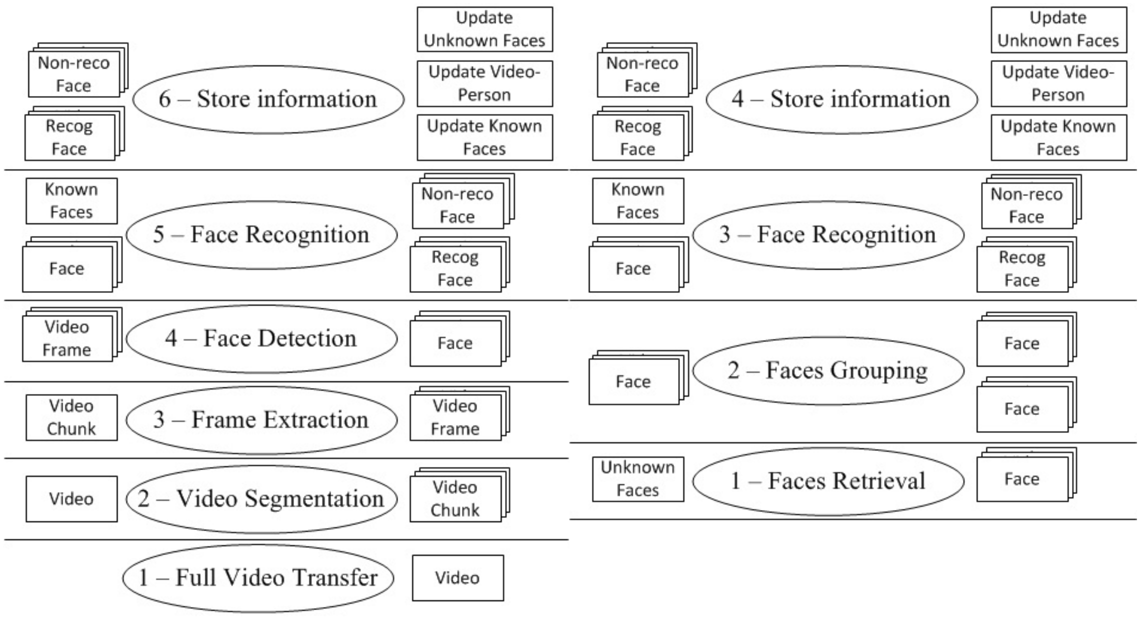
A simple **voting schema** will be adopted in order to take advantage of the large number of similar images video have available, again to simplify no complex voting schema will be used, instead a simple election will be done. When comparing a new face against the known identities the system will just verify is the majority of the images known for a person match the new one improving this way the accuracy of the recognition and at the same time not increasing the complexity to much.

4 Architecture

In this section, the architecture of the system we propose is detailed; first we specify the application division in individual tasks for each use case, secondly we will detail the distributed architecture and the middleware used to support the architecture as well, lastly we describe the scheduling algorithm that will be used in the system.

4.1 Use Cases

The figures in Figure 1 mention a few element of the system that are now explained: **Video** is a complete video file, **Video Chunk** is part of a video file,**Video Frame** is a still image from a Video, **Face** is a still image containing an isolated face, **Known Faces** is a repository of the mapping between Faces and persons' names used in the recognition process, **Recog Face** and **Non-reco Face** are respectively a Face identified and not identified in the recognition process, **Video-Person** is a repository of the mapping between videos and persons' names, and **Unknown Faces** is a repository of Faces not yet identified.



(a) New Video Submission. (b) Unknown Faces Reiteration.

Fig. 1: Use Cases.

Figure 1a) describes the main use case; the reception of a new video file, the execution of this use case is divided into six stages. **Stage 1** is the file transfer between a client application and the system, when completed the system has a complete video file to be processed. **Stage 2** is the segmentation of the video, since the application will be running in a distributed system, the bulk of processing a video files has to be shared by a number of machines; here two options could be considered, either machines would process entire files or smaller portions, the first option would difficultly produce good load balance, therefore the second will be adopted. **Stage 3** is the frame extraction, necessary since the face detection software works with frames. **Stage 4** is the first stage where the actual face recognition takes place, in this one, face images are extracted from the video frames. **Stage 5** is the second and final stage of face recognition; it produces a list of Recog Faces and Non-reco Faces. Stage 4 and 5 execution follows to process described in section 3.3.4. **Stage 6** is the final stage in this process; the various repositories are updated in this stage.

Figure 1b) describes the reiteration of the Unknown Faces from previous iterations; these Faces are kept in a repository. **Stage 1** is the retrieval of these faces from the repository. **Stage 2** is the grouping of these faces, for the same reason video files are segmented, the list of faces also have to be divided into groups. **Stages 3 and 4** are the same as stages 4 and 5 in the reception of a new video file.

The system have two more use case, not represented by images due to having only one stage, those are the system queries and the face identification. The first one is the answering of a question such as, what persons are present in a given video, for that purpose the video-person repository is used. The second one is the users providing the identification for a given face, this updates the various repositories.

All of these use cases functionalities rely on **mappings between the various elements**; this mapping is performed in the following manner: elements must have a unique identifier, while video files have a name given by the user, all other elements must have an auto-generated name; for simplicity, video chunks, frames and faces have names composed by the original video file name concatenated with a sequential number (e.g. the 2nd face, in the 23th frame, in the 6th video chunk of a file named ”video.wmv”, will have the name ”video.wmv-6-23-2”), this name will allow the correct mapping between an identified face and the video where it appears. The **Video-Person repository** maps videos to persons names, allowing the answering of queries; those are simple files containing a list of mappings. The **Know Faces repository** contains a mapping of names to a number of eigenvector, representation of faces, as specified in section 3.3.4, as for the **Unknown Faces repository** it is simply an unorganized list of faces.

4.2 Distributed Architecture

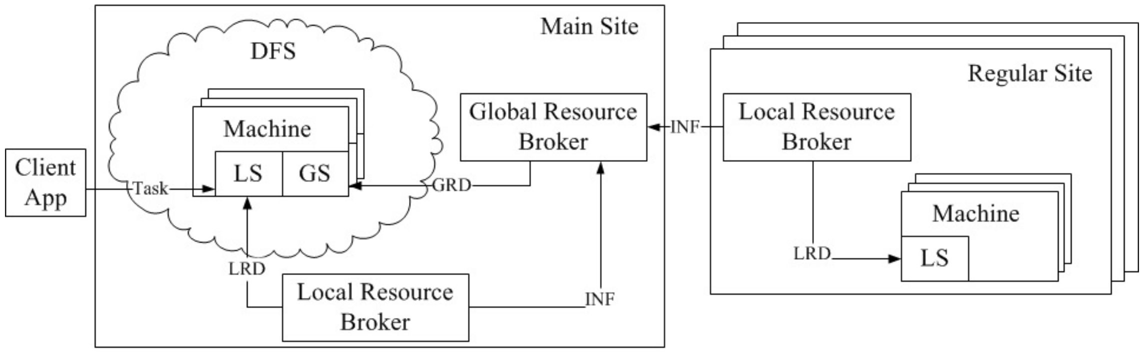


Fig. 2: Distributed Architecture.

Figure 2, shows the distributed architecture of the system, the arrow of the resource brokers mean: **GRD** - global resource decision, **GRD** - local resource decision and **INF** - information flux between resource brokers. In the architecture, we distinguish two types of cluster, the **main site** and the **regular sites**. The main site is where normally the whole application execution takes place and where all persistent data is kept. The decision of maintaining all data in the main site is justified by the reduction of software and control demand on other clusters (clusters to which we may have limited access). 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, so to scale the system we consider the hypothesis of sending tasks to other sites. The machines on the **main site** make part of a **Distributed File System**, where the repositories, videos files and intermediate files are stored. Each machine has a **Local Scheduler** and a **Global Scheduler**, the distinction between them is simple, the local scheduler makes decisions about in which machine a task is to be executed and the global scheduler makes decisions about in which cluster a task is to be executed. These two types of scheduler constitute a **hierarchical scheduler** as described in section 3.1.6. To perform these decisions and maintain consistency both types of schedulers rely on resource brokers, resource information systems that will make the actual decision. The machines in the **regular sites** will only have a **Local Scheduler** which work in the same manner as the Local Scheduler in the main site.

4.3 Software Architecture

Files are kept in a **distributed file system**, as the system execution will require intensive file access, traditional DFS that keep complete files in single machines could become a bottleneck. To eliminate this difficulty **HDFS** [42] will be adopted, this DFS standard behavior is to divide big files in smaller chuck and store those chunks in various machines, 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 video in chunks can be done accordingly to the HDFS file segmentation (corrected for frame boundaries within video files), and the chunk placement information can be used in resource brokering.In 3.3.4 we define the face recognition process, the actual face detection and recognition will be supported by the **OpenCV** library [43]. The performance of facial recognition software is not a concert of this work, the focus is on the distributed architecture, any facial recognition software could be used, our choice is justified by the large number of algorithms implemented in the OpenCV library.

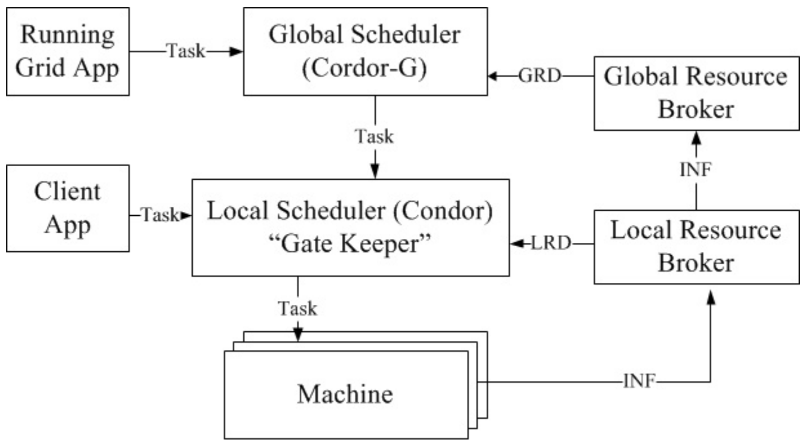


Fig. 3: Scheduler Architecture.

Figure 5 details the scheduler architecture and its software components. Here we differentiate two types of application, a **Client Application** and a **Running Grid Application**; the first one is an application running outside the grid, for instance an application that submits new video files to be processed and the second one is a system application running the face recognition process. Tasks from these applications are submitted to different schedulers for optimization purposes, the Stage 1 of any task that may be submitted by a Client Application will always be processed in the main site due to file placement restrictions, therefore the global scheduler is not useful for these tasks; on the other hand, tasks from other stages are submitted to the Global Scheduler as they can be executed in any site. As described in section 3.1.6 Condor-G will be used in the Global Schedulers and Condor for the Local Schedulers; in Figure 5 the local scheduler is classified as a **Gate Keeper**, these are normally the nodes responsible for controlling the access to a given site, normally only one or a reduced number of these node exist, but due to the characteristics of Condor (similar scheduler in all machines) all machines in a site can be used as a gate keepers, this will also provide better load balance.

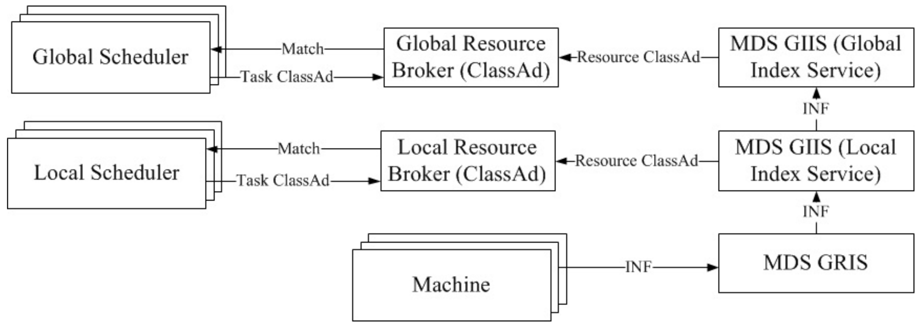

Global Scheduler
Match
Task ClassAd
Global Resource Broker (ClassAd)
Resource ClassAd
MDS GIIS (Global Index Service)
INF
Local Scheduler
Match
Task ClassAd
Local Resource Broker (ClassAd)
Resource ClassAd
MDS GIIS (Local Index Service)
INF
Machine
INF
MDS GRIS

Fig. 4: Resource Discovery Architecture.

Figure 4 details the resource discovery architecture and its software components. As described in section 3.2.3, the Global and Local Schedulers are coordinated by the **ClassAd Matchmacker**, both types of schedulers will submit classAds describing tasks due to be executed and receive match information from the Resource Broker. The actual resource information will be obtained by **MDS**, specifically a number of MDS GRIS modules that collect information directly from machines will be present in each site, in addition one MDS GIIS module will also be present in each site, this module will compile and index resource information that, after translated to classAds will feed the Local Resource Broker. The hierarchical architecture of MDS is useful in the construction of the MDS GIIS Module for the Global Resource Broker present only in the main site; this MDS GIIS compiles and indexes summarized information provided by the MDS GIIS modules at Local Index Service Level.

4.4 Scheduling

In this section we will describe the algorithm approach schedulers will use to choose in which machine each task will be executed. In general the scheduler will try to assign tasks to the machine in which the task will be executed in the least amount of time, in order to do that, we need to define a cost function that schedulers will try to minimize for each task. Before describing the approach it is important to decide what tasks are going to the scheduled. The system queries and the face identification done by the users, due to their time constraints and simplicity will not be scheduled; they can be submitted to any machine in the main site and will be immediately processed. All other tasks need to be scheduled, however stages 1 and 2 are done in the same machine, the one selected to execute the task in stage 1, it is this machine that will run the Running Grid Application that will control the execution of the rest of the process.

Cost Function:

The cost of executing a task for this particular system can be seen as the **predicted amount of time a task will take to be executed**, also, we must take into account the **data transfer time**. Therefore, the task execution time is the sum of the time to transfer the input and output files plus the execution time. The calculation of this cost function is a very demanding task, requiring high processing power if a site is constituted by a large number of machines, and requires very volatile resource information, the available inbound and outbound network speed as well as a prediction of the time a machine takes to process specific tasks, which would cause the resource information service to become the bottleneck of the system, as keeping this information constantly up-to-date is a demanding task. Also, It is not possible to make accurate prevision about the size of the output files (the results of processing a number of frames to extract faces can be one or one thousand faces). In addition ClassAd is not designed to make brokering decisions based on minimization of cost function, instead its design to find matches of values between resources and tasks.

Resources Attributes:

To solve the difficulties of the cost function calculations, we will describe the resources with groups of attributes, in this case {processor(CPU), ram memory (MEM), Hard Drive Capacity (HDC), available network speed (NTS)}, which one of these attributes may have one of these values Low, Med, High except for NTS which will have its actual value. In ClassAd the brokering decisions are made in two step, in the first one restriction and constraints are meet, task are matched with resources that can meet their requirement, if more than one matching is possible a second step is executed, in this second step one of the matched pair is chosen. In this work we will use the attributes CPU, MEM and HDC for the first step, where effectively we find machines that have the computational resources to execute the tasks; for the second step information about file placement and available network speed will be used to select one of the matched pairs by means of a cost function, machine with the needed files will have a cost of 0 while other will have a cost equal to the time the files would take to be transferred, note here that the output files are not accounted due to the impossibility of determining their size prior to execution.

The previous description was made for the Local Schedulers, the Global Schedulers, will work in a similar way but with different resource information, global scheduler will have information about how many machines are present in a cluster with one of the permutation possible for the groups of attributes (e.g. two machine with High, High, High, one machine with Low, High, High, etc), as for choosing between matched pairs, it is done in the same way as for the Local Schedulers.

In both schedulers, when the cost function is not sufficient to make a decision, a biggest first approach is used to select the resources, essentially the machine with the most resources available is selected, this approach has the disadvantage of filling big machines with small tasks, not allowing high resource demanding task to be executed, while smaller machines may be idling, however this is only used as a backup if all other criteria fails to produce just one matched pair, therefor it will not happen frequently enough to be a cause of concern.

4.5 Modification to the Existing Software

All components will have to be configured and connected; we will now list the most relevant modification we will do. APIs will be implemented in order to translate information between independent systems. File placement from HDFS will have to be translated and introduced in MDS, in which information collection is normally done by scripts or through APIs. Also, another API will be developed to translate the information from MDS to the format that ClassAd works with. As the Scheduling will be implemented via the Resource Brokers, the scheduling approach described in Section 4.4 will have to the written in terms of ClassAds objects components, namely Constraints and Rank. Also we must define the schema by which MDS will index the resource information and how it will summarize information sent from the Local to the Global index service.

5 Evaluation

The system to be developed will be tested and compared with similar ones, in this section we explain how we will test and evaluate the functionality correctness of the developed system as well as how we are going to compare our solution with others.

Firstly, we must assess whether the system is working correctly, as we will not develop the face recognition component, the test will be a simple verification of whether the **accuracy of the system** has been maintained. Accuracy will be measured by the percentage of **correct and incorrect recognized faces**, as well **detected and undetected faces**, the results should be identical, if not, the distributed architecture is not working correctly (system may be losing information).

Secondly and finally we must compare our solution with others, however, as it has be referred before, no face recognition system running in a grid environment exists, therefore the comparison will be done against other possible architectural solutions, that will also be implemented, those will be at least a centralized scheduler and centralized resource discovery system, note that an hierarchical architecture can easily be transformed into centralized. The solution will also be compared with the software running in a single machine. In order to perform these comparisons the following parameters will be measured: **total new video facial recognition time**, **face detection time**, **face recognition time**, **time to answer a query**, **average CPU load on each node** (balanced or imbalanced load distribution), the last parameter will not be measured in the comparison with a software running in a single machine.

6 Conclusions

We now wrap up this article with a brief summary of our work. Our objective in this work is to develop a facial recognition system that runs on a grid environment, developed to be scalable, efficient and mostly free.

After the related work research and analysis, we were able to choose the existing software that would be better adapted to our purposes, eigenvectors face for detection and recognition was chosen to perform the facial recognition related tasks, Condor and Condor-G were chosen for schedulers and ClassAd and MDS were chosen for Resource Discovery software.

Finally we described the systems distributed architecture and developed a novel scheduling algorithm specifically designed for this application, and in such a way that can be described by parameterizations of ClassAd queries.

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. E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.H. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In *Grid Computing*, pages 131–140. Springer, 2004.

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. S. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. The legion resource management system. In *Job Scheduling Strategies for Parallel Processing*, pages 162–178. Springer, 1999.

6. Volker Hamscher, Uwe Schwiegelshohn, Achim Streit, and Ramin Yahyapour. Evaluation of job-scheduling strategies for grid computing. In Rajkumar Buyya and Mark Baker, editors, *GRID*, volume 1971 of *Lecture Notes in Computer Science*, pages 191–202. Springer, 2000.

7. R. Buyya, D. Abramson, and J. Giddy. Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid. In *Proceedings. The Fourth International Conference/Exhibition onHigh Performance Computing in the Asia-Pacific Region, 2000.*, volume 1, pages 283–289. IEEE, 2000.

8. J. Krallmann, U. Schwiegelshohn, and R. Yahyapour. On the design and evaluation of job scheduling algorithms. In *Job Scheduling Strategies for Parallel Processing*, pages 17–42. Springer, 1999.

9. D.G. Feitelson and A.M. Weil. Utilization and predictability in scheduling the ibm sp2 with backfilling. In *Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International... and Symposium on Parallel and Distributed Processing 1998.*, pages 542–546. IEEE, 1998.

10. J. Turek, U. Schwiegelshohn, J.L. Wolf, and P.S. Yu. Scheduling parallel tasks to minimize average response time. In *Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 112–121. Society for Industrial and Applied Mathematics, 1994.

11. 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.

12. J. Gehring and A. Streit. Robust resource management for metacomputers. In *Proceedings. The Ninth International Symposium on High-Performance Distributed Computing, 2000.*, pages 105–111. IEEE, 2000.

13. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2):115–128, 1997.

14. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

15. J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke. Gass: A data movement and access service for wide area computing systems. In *Proceedings of the sixth workshop on I/O in parallel and distributed systems*, pages 78–88. ACM, 1999.

16. K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *Job Scheduling Strategies for Parallel Processing*, pages 62–82. Springer, 1998.

17. Condor Team. Directed acyclic graph manager, January 03, 2012. URL: http://research.cs.wisc.edu/condor/dagman/.

18. S. Fitzgerald, I. Foster, C. Kesselman, G. Von Laszewski, W. Smith, and S. Tuecke. A directory service for configuring high-performance distributed computations. In *Proceedings. The Sixth IEEE International Symposium on High Performance Distributed Computing, 1997.*, pages 365–375. IEEE, 1997.

19. L. Boloni and D.C. Marinescu. An object-oriented framework for building collaborative network agents. *INTERNATIONAL SERIES IN INTELLIGENT TECHNOLOGIES*, pages 31–64, 2000.

20. A. Cooke, A. Gray, L. Ma, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Byrom, L. Field, S. Hicks, et al. R-gma: An information integration system for grid monitoring. *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, pages 462–481, 2003.

21. T. Finin, R. Fritzson, D. McKay, and R. McEntire. Kqml as an agent communication language. In *Proceedings of the third international conference on Information and knowledge management.*, pages 456–463. ACM, 1994.

22. B. Tierney, R. Aydt, D. Gunter, W. Smith, M. Swany, V. Taylor, and R. Wolski. A grid monitoring architecture. *Global Grid Forum Performance Working Group*, 2002.

23. K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proceedings. 10th IEEE International Symposium on High Performance Distributed Computing, 2001.*, pages 181–194. IEEE, 2001.

24. 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.

25. R. Walker, M. Vetterli, A. Agarwal, D. Vanderster, RJ Sobie, and M. Grønager. A grid of grids using condor-g. In *Proceedings of Computing in High Energy Physics*, pages 13–16, 2006.

26. R. Ranjan, L. Chan, A. Harwood, S. Karunasekera, and R. Buyya. Decentralised resource discovery service for large scale federated grids. In *Proceedings IEEE International Conference on e-Science and Grid Computing.*, pages 379–387. IEEE, 2007.

27. E. Hjelmås and B.K. Low. Face detection: A survey. *Computer vision and image understanding*, 83(3):236–274, 2001.

28. W. Zhao, R. Chellappa, P.J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *Acm Computing Surveys (CSUR)*, 35(4):399–458, 2003.

29. O. Jesorsky, K. Kirchberg, and R. Frischholz. Robust face detection using the hausdorff distance. In *Audio-and Video-Based Biometric Person Authentication*, pages 90–95. Springer, 2001.

30. E. Hjelmas and J. Wroldsen. Recognizing faces from the eyes only. In *In Proceedings of the 11th Scandinavian Conference on Image Analysis.* Citeseer, 1999.

31. M.C. Burl, TK Leung, and P. Perona. Face localization via shape statistics. In *Int Workshop on Automatic Face and Gesture Recognition.* Citeseer, 1995.

32. M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

33. M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of cognitive neuroscience*, 3(1):71–86, 1991.

34. S.H. Lin, S.Y. Kung, and L.J. Lin. Face recognition/detection by probabilistic decision-based neural network. *Neural Networks, IEEE Transactions on*, 8(1):114–132, 1997.

35. L. Meng, T.Q. Nguyen, and D.A. Castanon. An image-based bayesian framework for face detection. In *Proceedings. IEEE Conference on Computer Vision and Pattern Recognition, 2000.*, volume 1, pages 302–307. IEEE, 2000.

36. I.J. Cox, J. Ghosn, and P.N. Yianilos. Feature-based face recognition using mixture-distance. In *Proceedings CVPR'96, 1996 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1996.*, pages 209–216. IEEE, 1996.

37. BS Manjunath, R. Chellappa, and C. von der Malsburg. A feature based approach to face recognition. In *Proceedings CVPR'92., 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1992.*, pages 373–378. IEEE, 1992.

38. B. Moghaddam and A. Pentland. Probabilistic matching for face recognition. In *Proceeding of IEEE Southwest Symposium on Image Analysis and Interpretation.*, pages 186–191. IEEE, 1998.

39. A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *Proceedings CVPR'94., 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1994.*, pages 84–91. IEEE, 1994.

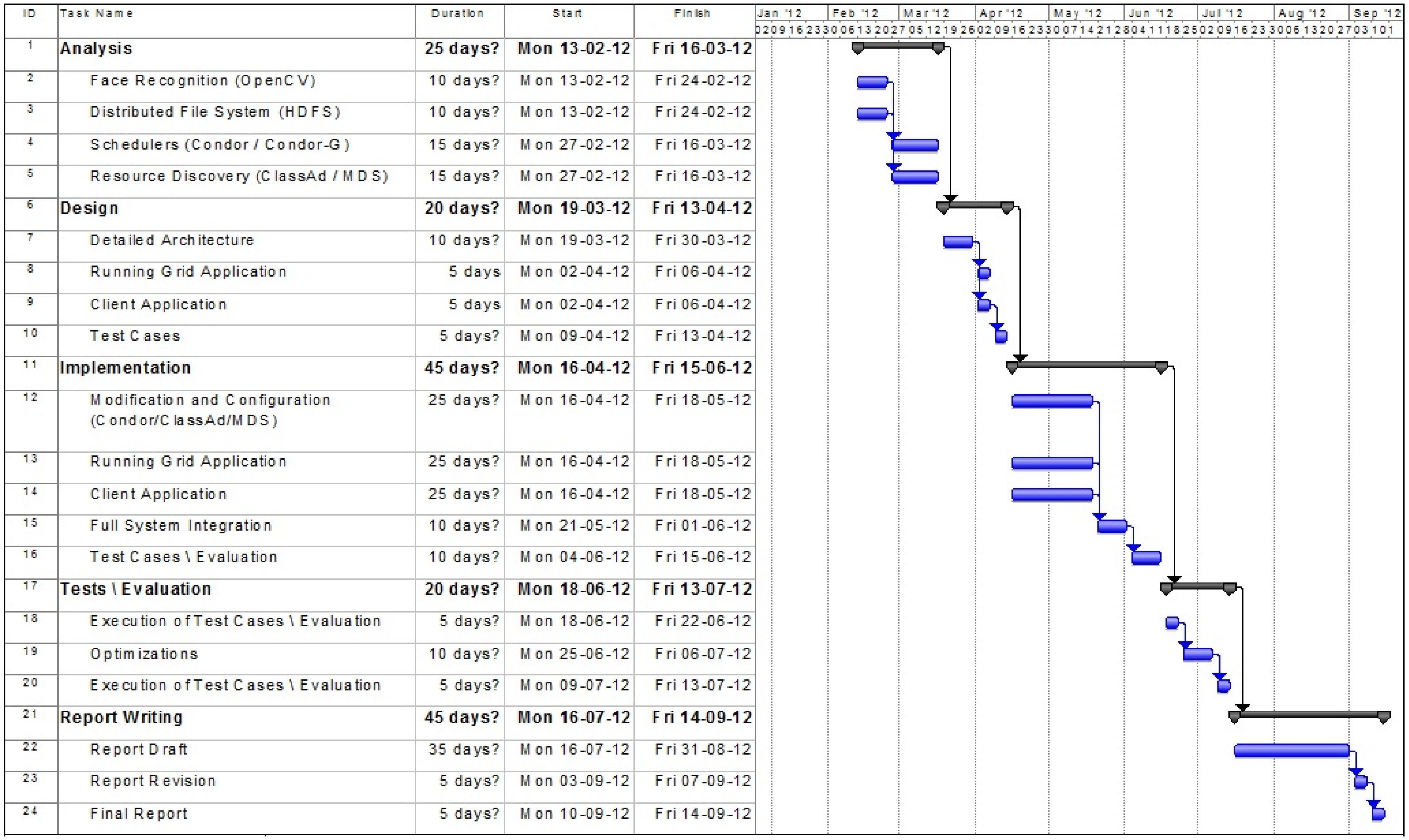
40. H.P. Graf, E. Cosatto, D. Gibbon, M. Kocheisen, and E. Petajan. Multi-modal system for locating heads and faces. In *Proceedings of the Second International Conference on Automatic Face and Gesture Recognition, 1996.*, pages 88–93. IEEE, 1996.

41. BK Low and MK Ibrahim. A fast and accurate algorithm for facial feature segmentation. In *Proceedings., International Conference on Image Processing, 1997.*, volume 2, pages 518–521. IEEE, 1997.

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

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

A Dissertation Work Planning



ID	Task Name	Duration	Start	Finish
1	**Analysis**	**25 days?**	**Mon 13-02-12**	**Fri 16-03-12**
2	Face Recognition (OpenCV)	10 days?	Mon 13-02-12	Fri 24-02-12
3	Distributed File System (HDFS)	10 days?	Mon 13-02-12	Fri 24-02-12
4	Schedulers (Condor / Condor-G)	15 days?	Mon 27-02-12	Fri 16-03-12
5	Resource Discovery (ClassAd / MDS)	15 days?	Mon 27-02-12	Fri 16-03-12
6	**Design**	**20 days?**	**Mon 19-03-12**	**Fri 13-04-12**
7	Detailed Architecture	10 days?	Mon 19-03-12	Fri 30-03-12
8	Running Grid Application	5 days	Mon 02-04-12	Fri 06-04-12
9	Client Application	5 days	Mon 02-04-12	Fri 06-04-12
10	Test Cases	5 days?	Mon 09-04-12	Fri 13-04-12
11	**Implementation**	**45 days?**	**Mon 16-04-12**	**Fri 15-06-12**
12	Modification and Configuration (Condor/ClassAd/MDS)	25 days?	Mon 16-04-12	Fri 18-05-12
13	Running Grid Application	25 days?	Mon 16-04-12	Fri 18-05-12
14	Client Application	25 days?	Mon 16-04-12	Fri 18-05-12
15	Full System Integration	10 days?	Mon 21-05-12	Fri 01-06-12
16	Test Cases \ Evaluation	10 days?	Mon 04-06-12	Fri 15-06-12
17	**Tests \ Evaluation**	**20 days?**	**Mon 18-06-12**	**Fri 13-07-12**
18	Execution of Test Cases \ Evaluation	5 days?	Mon 18-06-12	Fri 22-06-12
19	Optimizations	10 days?	Mon 25-06-12	Fri 06-07-12
20	Execution of Test Cases \ Evaluation	5 days?	Mon 09-07-12	Fri 13-07-12
21	**Report Writing**	**45 days?**	**Mon 16-07-12**	**Fri 14-09-12**
22	Report Draft	35 days?	Mon 16-07-12	Fri 31-08-12
23	Report Revision	5 days?	Mon 03-09-12	Fri 07-09-12
24	Final Report	5 days?	Mon 10-09-12	Fri 14-09-12

Fig. 5: Dissertation Work Planning.