Process, Storage and Video Facial Indexing in P2P Grids

Francisco Vasconcelos
francisco.vasconcelos@ist.utl.pt

Instituto Superior Técnico
INESC-ID

Abstract. Distributed peer-to-peer architectures have long since come to existence, and are motivated by the ultimate goal of pooling together large sets of resources, making their availability widespread to users and deployed applications. As such, resource discovery becomes paramount and subject to considerable focus by researching entities. Hence, two of the most common types of resource sharing systems are Grid and P2P, having evolved from two different communities. Also, study in face recognition has been done for over two decades now, and in the light of the hereby presented work, systems which perform these tasks can make great use of the aforementioned resource discovery and content sharing architectures. In this paper we intend to further their relationship, aiming for an architecture designed for face recognition in videos, which comprises of a peer-to-peer cycle-sharing system.

Key words: Peer-to-peer, Cycle Sharing, Face Recognition, CAN, Chord, Eigenvectors, Distributed, Decentralised, Cooperative

1 Introduction

Distributed computer peer-to-peer architectures are designed for computer resource sharing, while ultimately preserving the need for decentralisation, promoting direct exchange between participating peers, without the need for intermediation or support of a centralised authority. While pure content sharing accounts for most of peer-to-peer network overlay usage, resource discovery is also a key issue for many other applications, most of which - such as in Grids - are of scientific orientation.

One such application is facial recognition, which by itself and at a large scale, can make great use of cycle sharing. The process of analysing a video undertakes many stages, such as processing the various images within frames, tagging and analysing these images, identifying human faces across various poses, illumination and different attires worn, and ultimately recognising specific individuals. All of this is a potential heavy computational effort.

While there is a lot of work done in related subjects, there is none (as far as we know) accomplished for the specific goal of face recognition over widespread free peer-to-peer network overlays in Grid environments. As such, and while this comes as an added difficulty, it presents itself as an exciting and novel challenge to overcome. We hence intend to develop our solution, of a facial indexing and recognition video processing distributed system, within the Ginger Project: a peer-to-peer grid infrastructure that overcomes the existing barriers for widespread utilisation of Grids, allowing cooperative work among users.

The rest of document is organised as follows: we start by stating what we plan to achieve with our work in the Objectives section (at 2), to then make a study of the related work in the State of the Art section (at 3), moving on to the design and assessment of the architecture of our solution (at 4 and 5). We finalise the paper with concluding remarks on our work, at the Conclusions section (at 6).

2 Objectives

Our goal is to design an architecture for a peer-to-peer architecture which, while working in a Grid environment, allows for a set of determinate goals to be achieved. These are the storage, processing, indexing and efficient search of content and metadata relative to the detection and subsequent identification of human faces in video files.

As such, we intend to implement a peer-to-peer cycle-sharing system, that processes and indexes stored videos in the aforementioned architecture, thus making use of idle CPU cycles and free storage space on the nodes which belong to the resulting network. The architecture itself, is to be layered in a way which allows it to overcome the difficulties of processing the large amounts of resulting data, from the processing of videos to their iterative analysis.

In order to achieve the said goals, a thorough study of related work done is necessary, so as to better understand both the technical and theoretical requirements of our work. We thus focus on three main subjects: peer-to-peer, cycle sharing, and face recognition. Finally, after having revised these and developed the architecture for our solution, its evaluation shall consist of a set of qualitative, quantitative and comparative assessment criteria, which we believe to be adequate performance measures for our work.

3 State of the Art

In this section we present the three major research topics investigated, which we believe to be representative of the substrate of our work. As such, they are hereby presented, and they consist on: **Peer to Peer** (see 3.1), **Cycle Sharing** (see 3.2) and finally, **Facial Recognition** (see 3.3).

The two former sections (**Peer to Peer** and **Cycle Sharing**) consist on the initial groundwork on which the latter section (**Facial Recognition**) functions, and the understanding of this will become clear when the architecture of the solution is explained (see 4 for further details).

3.1 Peer to Peer

Almost two decades have passed since the birth of the Internet to the public eye, and about ten years ago a great spurt of growth was witnessed in the field of **peer-to-peer** (P2P) **networks**. First, with the introduction of Napster in 1999 which brought upon a great change of focus [1]: from HTML pages to file-sharing (as seen at the time in [2]). And this trend has hardly changed, as far as its general direction is concerned, with P2P taking up a large portion of the interest of users, of the bandwidth in most networks, and naturally, of the amount of information circulating in the Internet.

The interest in using peer-to-peer solutions and of basing applications on them, is founded on their *ability to function, scale and self-organise in the presence of a highly transient population of nodes, network, and computer failures, without the need of a central server and the overhead of its administration* (as seen in [3]). Naturally, the issues of **scalability**, **resource distribution** and of **resistance to censorship** are of great appeal, and are some of the advantages that peer-to-peer systems offer. Additionally, the users are the ones who make up these systems, as there is (generally) no centralised *notion of ownership* [3].

Summing up, peer-to-peer systems **accelerate communications** amidst their users, while doing so at **very low costs** (and at very shaky legal parameters in many cases), **enhancing overall collaboration**, having revolutionised both the usage of the Internet and its users. We first now present our definition of "*peer-to-peer*" (in the next section, see 3.1.1), to then study unstructured systems (at 3.1.2 and focusing on structured systems afterwards at 3.1.3). The section is concluded with a reference to hybrid systems 3.1.4 and with a correlation of this section with our work (at 3.1.5).

3.1.1 Defining Peer-to-Peer

Ever since its existence, the definition of **peer-to-peer** has been subject of great debate, and there is significant literature studying this, despite its roots reveal that at its core, these systems are about completely distributed systems, where all participants are equivalent as far as their responsibilities go.

Most past definitions are, therefore, either "*pure*" or "*impure*". The former refers to systems where there is absolutely no control (centralised or not) from a higher-up node or subset of nodes. The latter contradict the former proposition, and circumscribe systems with "supernodes" (like Kazaa), that are also widely accepted as **peer-to-peer**, or systems which *rely on some centralised server infrastructure for a subset of noncore tasks* [3].

And so, notwithstanding the diversity of definitions encompassing the subject, we consider - in accordance to [3] - that what is labelled as **peer-to-peer** should be so based on how these systems are perceived "externally": how the nodes perceive the system based on *the impression of providing direct interaction between computers.*

In the following three subtopics we shall hence cover the topic of understanding P2P systems, the caveats behind defining it and what they truly encompass in the context of our work.

3.1.1.1 What is Peer-to-Peer

When attempting to define **peer-to-peer** systems, certain specific characteristics should be emphasised, so as to have a clear understanding of where the bedrock of our definition settles itself in. Hereinafter, we present three main traits of these systems.

Resource Sharing

Every P2P system should make use of **resource sharing** by direct exchange between the participant nodes, rather than making use of any sort of centralised control. Nevertheless, whilst these may perform certain noncore tasks (like the addition of new nodes to the network), we consider that the nodes should not *rely on a central server coordinating the exchange of content and the operation of the entire network* [3], but will in fact perform **resource sharing** while depending solely on themselves.

Node Equality

As can be drawn from what was previously said, the nodes will in fact be unilaterally, independently, personally responsible participants that perform every task involved in whatever their activity be in the network - even if they are not capable of acting accordingly, due to such issues as availability. To boot, they thus perform the following independently: search and cache content, for other nodes, connect and disconnect from neighbouring nodes, encrypt and decrypt, introduce and remove content, among other activities. Thusly achieving the notion of **node equality**.

Self-Stabilisation

Finally, nodes in **peer-to-peer** systems should have the ability to *treat instability and variable connectivity as the norm, automatically adapting to failures in both network connections and computers* [3], transiently shifting freely through different populations of nodes, as they conform automatically to different conditions.

3.1.1.2 Classifying Peer-to-Peer

With the fundamental characteristics of P2P systems understood, we now present our definition of these systems: *Peer-to-peer systems encompass a class of systems that make use of resources in a distributed fashion, where their nodes are self-organisable in fault-tolerant and self-organising network topologies which promote their participants equality, accommodating the connectivity and performance of a transient population of nodes.*

Furthermore, as seen in [3], we feel like a proper classification of the current peer-to-peer infrastructures will aid the understanding of the correlation of this section of the document with our work (as seen in 3.1.5 below).

In an introductory fashion (since the following mentioned infrastructures shall be covered ahead), a brief description of the **routing and location infrastructures** presented is now given. **Chord** is a scalable peer-to-peer lookup service which given a key, it maps it to a specific node; **CAN** works as a scalable content addressable network, providing hash-table functionality for the

Table 1. Classification of Peer-to-Peer Infrastructures

Infrastructures for routing and location	**References**
Chord	Stoica,I. et al. [4]
CAN	Ratnasamy, S. et al. [5]
Pastry	Rowstron, A. et al.[6]
Tapestry	Zhao, B. et al. [7]
Infrastructure for anonymity	**Reference**
Tarzan	Freedman, M.J. [8]
Infrastructure for reputation management	**Reference**
PeerTrust	Xiong, L. et al. [9]

mapping of file names to their locations; **Pastry** and **Tapestry** are fault-tolerant wide-area location and routing infrastructures.

As for the **anonymity infrastructures**, **Tarzan** is a peer-to-peer decentralised anonymous network layer; and as for the **reputation management infrastructures**, **PeerTrust** is a decentralised, feedback-based reputation management system that uses satisfaction metrics and the number of interaction metrics. We shall concern ourselves merely with the four firstly mentioned architectures: Chord [4], CAN [5], Pastry [6] and Tapestry [7].

Now, at this point one might think of including **Grid** technologies in this discussion, but we feel like a small contrast must be settled first, since these are disparate enough to be settled in their own niche of definitions. So, we tackle the issue of P2P versus Grids later on in the **Cycle-Sharing** section (see 3.2). As such, we now move forward into introducing **structured**, **unstructured** and **hybrid** systems of **peer-to-peer** architectures, as these are paramount to the architecture of our solution (in 4).

3.1.2 Unstructured Systems

There are millions of users using **peer-to-peer** systems, making use of massive data-sharing and content distribution, and it has been stated, the applications that make it possible are built upon *network overlays that provide mechanisms to discover data stored by overlay nodes* [10]. As such, there are two major types of overlays: structured and unstructured.

In the **unstructured** overlays, the placement of content is completely unrelated to the overlay topology, so it needs to be searched for with searching mechanisms which have many different approaches. These range from brute force methods such as network floods, to more sophisticated strategies that include the use of random walks and routing indices, and have a determinant impact on these networks, namely on what concerns availability, scalability and persistence. Examples of such systems include Napster, Gnutella, Kazaa, FreeHaven, among others.

We now explain three main categories of **unstructured** architectures, specifying to what extent the network overlays are decentralised, despite the fact that **peer-to-peer** networks should be completely decentralised (which is far from being true, since there are *various degrees of centralisation* which can be encountered [3]).

Purely Decentralised Architectures

Purely decentralised architectures such as the **Gnutella** network are built upon virtual overlay networks with their own routing mechanisms, that allow users to share files with other peers. Since there is no central coordination of the activities in the network, users connect with each other directly through software applications, functioning as servers and clients: *servents* [3]. We focus the next two subtopics with the Gnutella example, as it is representative of how these systems function.

Resource Organisation

Naturally, resource organisation in these networks is generally dependent of the participant nodes, which hold the files independently. Therefore, there is no particular form of organising where content is distributed.

Data Discovery

Locating files is done by nondeterministic searches; in the (original) Gnutella architecture a flooding/broadcast mechanism distributes messages to nodes, that forward the received messages to their neighbours, and the answers along the opposite path through the original path from where the request came from. Just so these messages do not overflow the network, they have a fixed TTL (time-to-live) value that makes them drop if ever with that field at zero.

Partially Centralised Architectures

Finally, some architectures are only partially centralised. In these, similar functionality to the one in hybrid systems can be found, but they use *supernodes*. These nodes are assigned dynamically with the task of servicing subsets of the peer network, indexing and caching files contained therein. An example of such a network is **Kazaa**, a well-known proprietary file-sharing system.

Resource Organisation

The partially centralised systems organise their content in accordance to the dynamic assignment of supernodes, that have several connections with other superpeers, forming an unstructured network of superpeers. Ergo, the contents are organised amongst these, which receive requests for files from regular client nodes.

Data Discovery

The supernodes are the ones who index the files shared by peers connected to them in these architectures, and proxy search requests on behalf of these nodes, thus all queries being initially directed to the supernodes. Since they index subsets of the overall content, discovery time is quite reduced in these systems, with there being no single point of failure, since the assignment of supernodes is dynamic.

It should be pointed out that there are other methods for overcoming the basic unscalability of unstructured peer-to-peer architectures, we give such examples hereafter.

1. **Random Walks**
   In these, each peer chooses a random neighbour, and propagates its requests to it only. In Gnutella 0.4 [10], each node in the overlay *maintains a neighbour table with the network addresses of its neighbours in the graph*, with these neighbour tables being symmetric amongst the peers.

2. **Sophisticated Broadcast Policies**
   These (as seen in the work by Yang et al. in [11]) select which neighbours to forward search queries to, based on past recorded history and on the use of *local indices*, which maintain indexes of the data stored at the nodes within a specific radius from itself.

3. **Intelligent Search Mechanisms**
   There are some approaches like these, and in one such method (by Kalogeraki et al. in [12]) each peer forwards queries to a subset of its neighbouring peers, with the selection being based on peer profiling (specifically, on their performance), thus creating a ranking system that allows for the choosing of the most appropriate peers.

4. **Routing Indices**
   The routing indices are *tables of information about other nodes, stored within each node* [3], that provide a list of neighbours which are more likely to be route-friendly for message-carrying given their content. With the information about the files being held by these neighbouring nodes, this solution (suggested in the work of Crespo et al. in [13]) results in a great upheaval of performance as far as scalability and searching goes.

Hybrid Decentralised Architectures

In these, each client computer stores content shared with the rest of the network, and all clients connect to a central directory server. A typical example of such a network is **Napster**: which relies on static system-wide lists of servers. A quick and obvious caveat to these systems is their lack of scalability.

Resource Organisation

In these systems, the central directory server maintains a table listing the files each user holds and shares in the network, with the metadata descriptions of these contents. In these systems, file discovery is rather quick and efficient, although they are very vulnerable to such issues as censorship, malicious attack, surveillance and technical failure. This is because a single institution controls the content sharing and discovery.

Data Discovery

The central directory server, besides also holding the aforementioned file table, it also stores a table of registered user connection information. As such, when a user joins the system, it contacts the central server that reports the files it maintains: then the user discovers files by requesting them to the server, that in turn searches for matches in its index, returning a list of users that hold the matching file.

There are also *loosely structured* network overlays, since their category is hard to pin down in any of the other two definitions. In these, the location of content is not completely specified, but it is affected by routing. One such example is Freenet [14], though we shift the focus of this paper now, as we move on to analysing **structured** overlay networks, which are of greater relevance for the time being.

3.1.3 Structured Systems

In this section we briefly cover the (widely accepted as) four main structured peer-to-peer implementations, as well as two other implementations: **Chord**, **CAN**, **Tapestry** and **Pastry** are the four main structured systems, with **Viceroy** and **Koorde** being the other two implementations. Generally, these systems, in response to scaling problems that unstructured designs usually suffer from, support a *distributed hash table* (**DHT**) functionality [15]. Files are then usually associated with a key, and each node in the system is responsible for storing a certain range of keys. Habitually common to these systems is the **lookup(key) operation**, that returns the identity of the node storing the object with the given key. The relevance of enumerating these is to clarify any implementation choice settled further in our work.

Chord

In Chord [4], an infrastructure for peer-to-peer routing and location is accomplished, with there being a mapping of file to node identifiers. In fact, in its core, this protocol supports just one operation: *given a key, its maps the key onto the node* [4]. Also, **consistent hashing** is used to assign keys to Chord nodes, thusly achieving a balanced load, with nodes receiving an approximately same number of keys. Chord's primary keyword achievements are: **load balance**, **decentralisation**, **scalability**, **availability** and **flexible naming**.

Resource Organisation

Chord achieves a very important concept of **load balancing**, acting as a distributed hash function, spreading the keys evenly over the nodes [4]. As so, the previously mentioned **consistent hashing** function assigns each node and key an m-bit *identifier* using a base hash function. The node identifier is chosen by hashing the node's IP address, with a key identifier being produced by hashing the resource name. Specifically, identifiers are ordered *in an "identifier circle" modulo* 2^m [3]. So key k is assigned to the first node whose identifier is equal to or follows k in the identifier

space: this node shall be called the *successor node* of key k. In order to maintain the **consistent hashing** working properly when a node joins the network, a subset of the keys previously assigned to its successor now become his own; and when a node leaves the network, all of its assigned keys become his successor's.

Data Discovery

As it can be drawn from what was previously stated, data discovery in Chord is made by queries for a given key being passed around the circle via the successor pointers until node that contains the key is encountered: typically, since each node maintains information about only O(log N) other nodes, resolving lookups via O(log N) messages to other nodes [4], discovering content is pretty straightforward. Albeit, traversing all N nodes might be necessary in the worst case, and this is why Chord maintains a "**finger table**", where each entry i points to the successor of node $n + 2i$. Hence, when a node performs a lookup for a given key, it will consult the finger table, to first check for a closer node to the required key.

CAN

The Content Addressable Network is *essentially a distributed,* ***Internet-scale hash table*** *that maps file names to their location in the network, by supporting the insertion, lookup and deletion of (key,value) pairs in the table* [3]. The CAN design comprises scalable, fault-tolerant, self-organising networks [5] that provide robustness and low-latency properties. Each node in these networks **holds a part** of the overall information regarding resource organisation - a "*zone*" - distributing the responsibilities among these.

Resource Organisation

Through the use of a **virtual *d*-dimensional Cartesian coordinate space**, that stores (*key K*,*value V*) pairs, each *zone* a node is responsible for corresponds to a segment of this coordinate space. As such, any key K is thusly deterministically mapped onto a point P in the coordinate space. The (K, V) pair is then stored at the node responsible for the zone within which point P lies.

Data Discovery

Each individual node in a CAN network **stores a "*zone*" of the hash table**, together with the information about a **small number of adjacent zones** in the table. So, requests to insert, lookup, or delete a specific key are **routed via intermediate zones** to the node that maintains the zone containing the key. A retrieval of an entry corresponding to a key K by a node, is made by applying the determinist function that was used to map the corresponding (K, V) pair to retrieve the corresponding value V from the node covering the point P. The routing of the messages, in essence, is made by greedy forwarding messages to neighbours with (**X-axis**, **Y-axis**) coordinates closest to the destination coordinates, along a defined coordinate space direction. As such, routing in CAN works by following a **straight line path** through the Cartesian space, from a source node to any given destination coordinate.

Tapestry

Tapestry [7] is based upon a self-organising network topology, where nodes come and go freely, with the network latency varying as well. It supports the location of objects and the routing of messages to them (or the closest copy of them, if more than one copy exists) in a distributed, self-administering, and fault-tolerant manner, offering system-wide stability by bypassing failed routes and nodes, quickly adjusting communication topologies to circumstances. Furthermore, the routing/location information is distributed among the nodes.

Resource Organisation

Tapestry is based on the **Plaxton Mesh** [16], a distributed data structure that allows nodes to locate objects and route messages to them across an arbitrarily-sized overlay network, using routing maps of small and constant size. While not providing explicit resource organisation, a *root node* is used for each object stored, that provides a guaranteed node from which the object can be located. And nodes, through the process of *data discovery* come to know of these. Also, *storage servers* (also nodes), republish location information for objects they store at regular intervals, with the possibility of finding copies of objects also imbued in the algorithm.

Data Discovery

Every node in **Tapestry** maintains a multiple-level *neighbour map* [7] - with each level l containing pointers to nodes whose ID must be matched with l rightmost digits - and messages are incrementally routed to the destination node digit-by-digit. Details aside, nodes can publish objects, and when **locating** an object, they send messages to them, which is routed through the network topology, finding either the object itself or the nearest replicas of the aforesaid object.

Pastry

Pastry [6] is a scalable distributed object location and routing substrate for wide-area peer-to-peer applications [6]. Performing application-level routing and object location, it can be used to support a variety of peer-to-peer applications, such as global data storage, data sharing, group communication and naming. In this system, every node has a unique identifier, the *nodeId*, which is used to pass messages among the nodes along with a key. It comprises a decentralised, scalable, self-organising architecture.

Resource Organisation

Alike **Tapestry**, each node keeps track of its immediate neighbours in the *nodeId* space, notifying applications of new node arrivals, node failures and recoveries. Since these IDs are randomly assigned, the set of nodes with adjacent *nodeId* is diverse in geography, ownership, etc. Thus, applications such as PAST [17] (which is a self-organising, large-scale global storage application) can leverage this, since **Pastry** can route to one of k nodes numerically closest to the key. Again, in PAST, as an example, which is built upon **Pastry**, file storage is made based on these concepts, making file storage a more agile process.

Data Discovery

Since each node has a unique identifier, when presented with a message and a numeric key, a node routes the message to the node with a *nodeId* that is numerically closest to the key, among all currently working Pastry nodes. It is expected that the number of routing hops be O(log N), with N being the number of Pastry nodes in the overlay network. All of this is done while taking into account network locality, seeking to minimise the distance messages travel according to a scalar proximity metric like the number of IP routing hops, or the round-trip time.

Having seen the four primary structured approaches, we now scoop out two equally important systems, that are also worthy of mention. These are **Viceroy** and **Koorde**, and are covered hereafter.

Viceroy

Viceroy [18], is designed proposing constant-degree routing networks of logarithmic diameter, requiring **no global coordination** for the addition or removal of nodes. It ensures that the congestion of the network is generally within a logarithmic factor of the optimum, **managing the distribution of data** among a changing set of servers, and allowing clients to contact any server in the network to find any stored resource by name, employing **consistent hashing** like the **Chord** protocol.

In **Viceroy**, **Resource Organisation** is made by a varying set of servers (called a *configuration*). The servers are organised in *levels*, that indicate a server's placement in the network [18]. As a functioning example, we mention a *JOIN* operation, where a server joins the network. In these, a server gets into the "server ring" (like **Chord**), and *transfers all the key-value pairs with keys between the one his predecessor has and himself* [18]. Moreover, **Data Discovery** is accomplished by the *LOOKUP(x,y)* subroutine, that starting at server y, find the server closest to the value x. The way this functions is elaborate, so it suffices to say that it makes use of the distributed hash functioning along with the *level* structure used.

Koorde

The **Koorde** system [19] is also based on **Chord**, looking up keys by contacting O(log n) nodes with $O(1)$ state per node. Thus, keys are mapped to nodes, with a node and keys having identifiers that are uniformly distributed in a 2^b identifier space. Keys are stored in node successors, which are the neighbouring nodes that follow others in the identifier circle.

While **Viceroy** [18] does not approach issues such as fault tolerance, and it is also relatively complex, **Koorde** tries a different approach. Using consistent hashing to map keys to nodes, these are distributed in a 2^b identifier space, with keys being distributed at their *successors*. Now, the difference between Koorde and Chord is that the former embeds a de Bruijn graph on the identifier circle for forwarding lookup requests. This basically makes routing a message from node m to node k by taking the number m and shifting the bits of k - since a de Bruijn graph has a node for each binary number of b bits - one at a time until the number has been replaced by k, with each shift corresponding to a routing hop to the next intermediate address.

It should be noted that we addressed only the four main implementations of structured networks (and two other approaches briefly) whilst there are many more - either novel designs or (mostly) based on any of the structured systems covered - that could be approached. Such examples are **PAST** [17] (based on Pastry [6]), **Freenet** [20], or **Free Haven** [21]. We now move forward into understanding how these four main **structured peer-to-peer architectures** are related to the nature of our work, after a brief mention to **hybrid systems**, which are mixtures of both **structured** and **unstructured** systems.

3.1.4 Hybrid Systems

Hybrid systems come to existence in order to compensate for the disadvantages found in both structured and unstructured designs. These (also called *loosely structured* in other literature [3]) are characterised as being in between structured and unstructured networks, where the location of content is not completely specified, but it is affected by routing hints. Many hybrid approaches have been proposed to overcome these drawbacks [20,22,23]. One such approach is *Kademlia* [23]: a peer-to-peer DHT where each peer is ampped to a 160 bit key through the SHA-1 hash function.

The fundamentals behind its functioning go like this: each peer subdivides the space of possible distances between any keys, defined as their XOR. Also, every node is aware of at least one other participant, whose distance from its key is between $2i$ and $2i + 1$, for $0 <= i < \log(N)$. The aforementioned ranges are called *buckets*, and each peer tries to maintain knowledge of k peers in each bucket - also, each file key is also stored on the k peers closest to its key. Finally, routing is performed by calculating the XOR of the requesting peer's key with the lookup key and forwarding the lookup request to a peer of the appropriate bucket.

3.1.5 Interrelationship with our Work

In this section, we assign which systems are of our interest, as we answer two different questions: "**which peer-to-peer system serves us best: unstructured or structured?**" and "**out of the chosen systems, why were these chosen over the rest?**".

1. **Unstructured or Structured?**
   We have already seen that in **unstructured** peer-to-peer network structures the placement of content (files) is completely unrelated to the overlay topology [3]; moreover, content typically needs to be located often through *flooding*, and these searching mechanisms (also like *brute force*) tend to have a detrimental impact on **scalability**. Additionally, these systems are more appropriate for accommodating highly-transient node populations. Clearly, none of this is advantageous for our work, and as such, **structured** systems are clearly our choice - even if widely known to not do well under high churn rates, which is not a concern for our work, since we will not deal with a highly transient population of nodes - since the aforementioned issues are not a problem in these. As an added value, table 2 (as seen in [24]) shows that while the **robustness** of structured systems is in fact lower - due to these not being adequate for **intermittent** users that join, depart and rejoin the system unpredictably - the possibility of **periodic update** messages and **range queries**, in addition to the inherent **order and data locality** [24] characteristics, make **structured** systems a more appealing choice, despite their caveats.

Table 2. Unstructured and Structured P2P systems

Characteristics	Unstructured	Structured
Scalability (time)	O(log n)	O(log n)
Scalability (traffic)	N*avg degree	O(log n)
Robustness	High	Lower
Periodic update	No	Yes
Range Queries	No	Yes

2. **Why Chord and CAN?**
   To begin to understand the choice of **Chord** and **CAN** as preferable structured systems, we begin to highlight that **performance is not the issue**, as we present a comparison of expected performance measures, as seen in [18], omitting Pastry because of its similarity with Tapestry. On a sidenote, the $\boldsymbol{n}$ in any of the metric evaluation refer to the number of nodes, and in CAN's case, $\boldsymbol{d}$ is a constant that refers to a d-dimensional torus.

Table 3. Performance Comparison of Lookup Schemes

Lookup scheme	Dilation	Congestion	Linkage	Neighbours per node
Chord[4]	log n	(log n/n)	log n	O(log n)
CAN[5]	log n	(log n/n)	log n	O(d)
Tapestry[7]	$dn^{(1/d)}$	$dn^{(1/d\text{-}1)}$	$O(1)$	O(long n)

One very important argument supporting **Chord** is an example of a possible application for which it would provide a good foundation, as stated in [4]: multi-layered software structures for a cooperative mirror work, with higher layers closer to the users (using file naming, authentication), mapping lower levels with subsequently simpler (basic) formats of data. Since we envision a multi-layered environment as well, **Chord** comes off as particularly appealing, notwithstanding its "blind", simple **load balancing** techniques, scattering keys regardless of the resources needed for each document.

So, while **Chord** may span to a multi-dimensional architecture, why use any other system? All of these algorithms start by assuming all nodes as having the same capacity to process messages, and then, only later, add on techniques for coping with **heterogeneity**. But what about other aspects? Well, for starters, the most obvious measure of the overhead associated with keeping routing tables is the **number of neighbours** [15], and as can be seen, most of the algorithms require O(log n) neighbours, while **CAN** requires only O(d) neighbours. Another aspect, as can be taken from [5,7], is that **CAN** is prompt by design to structure information in a manner that is useful to our work, around various parameters; also, its simple and efficient routing is also an appealing feature.

Having understood the different **Peer-to-Peer** systems and architectures - and how they correlate themselves with our work - how they function and how they differ, the section on **Cycle Sharing** is hence approached, as a lot of our work bases itself upon the sharing of resources as well.

3.2 Cycle Sharing

Cycle sharing (or **resource sharing**, it depends on the author, though one could say "cycles" are a subset of resources that are apportioned between peers), is a process where large sets of computational resources are pooled together and made available to users and deployable applications alike. Of course the resources themselves need to be **discovered** - thusly making themselves available - if they are to be of any use. This is why **resource discovery** is of sovereign importance: systems who make use of it discover resources across multiple administrative domains, based on a list of predefined attributes, returning lists of locations where the required resources reside.

Some of the most common types of resource sharing systems make use of the **Grid** and **Peer-to-peer** technologies, having evolved from different communities and originally serving different needs. **Grid** systems are in fact (in their origin) meant for the sharing of resources (such as CPU time, storage, data, etc) among interconnected computer clusters (generally for research purposes, through attribute specification queries like multi-attribute queries). **Peer-to-peer** systems usually serve a more diverse multitude of purposes, such as the all-time popular *file-sharing* (KaZaA), *real time data transfer* (Skype), *cycle stealing* (SETI@HOME), or *collaboration* (Groove) [24]. In table 4 we can see some basic differences between **Grids** and **Peer-to-peer** [24].

Table 4. Differences between Grid and Peer-to-Peer systems

In terms of	Grids	Peer-to-Peer
Users	Scientific community	Any desktop user
Computing	Workstations, multiprocessors, computer clusters	Any desktop user
Network	Mostly high-speed dedicated networks	Internet TCP connections
Administration	Centralised or hierarchical	Distributed
Applications	Complex scientific applications such as large scale simulations	File sharing, real-time data streaming, cycle stealing
Scale	Connect a relatively small number of specialised sites	Connection is possible from any desktop
Security	Secure services for job submission and execution	Protocols for file sharing
Participation	Static or slowly changing participation of nodes	Nodes can enter or leave tottaly unpredictably
Trust	Trusted users	Untrusted, anonymous users

So by now we know the different nature of **Grids** and of **Peer-to-peer**, how do these two intertwine constructively? Well, as the scale of Grid systems increases: *centralised management* will prove inefficient and other methods will have to be considered; *QoS constraints* that govern most Grid applications will loosen up; *resource participation rules* will have to be lax to accommodate a more diverse range of user intentions. Also, *Peer-to-peer* systems will have to accept a more complex range of *QoS levels*, *applications* and *more elaborate queries*. Now, we already understand how resource discovery is made on **unstructured** and **structured** systems (as seen in the previous

section at 3.1, under 3.1.2 and 3.1.3 respectively). So, to further understand how these correlate, we move forward to introducing Grid systems that incorporate P2P resource location methods (starting with 3.2.1), with an additional mention to resource discovery solely made in Grid systems (see 3.2.4).

3.2.1 Grid Resource Discovery and Unstructured P2P systems

As we have already stated in 3.1.5 that our choice would befall **structured peer-to-peer systems**, our mention on **unstructured** grid-based resource discovery shall be a short one, with a brief review of all the possible architectures. As so, these can be **Flat Peer-to-Peer networks** [25], **Tree-based Overlays** [26] and **Super-Peer Networks** [27]. In the **first** [25], a fully decentralised architecture for resource discovery in Grid environments is proposed. In this architecture, every node in a VO (Virtual Organisation) publishes information on one or more local servers (*nodes* or *peers*), that store and provide access to local resource information. It is to these servers that *nodes* forward their requests. In the **Tree-based Overlays** such as in [26], a routing indexes Grid resource-based searching mechanism is implemented. Here, nodes are organised in a **tree-structured overlay network**, *where each node maintains information about the set of resources it manages directly, and a condensed description of the resources present in the sub-trees rooted in each of its neighbouring nodes* [24]. As such, these indexes are explored by the data location algorithm in order to route queries towards areas where matches can be found.

A special mention goes to the third and final methodology, described in [27]. Originally, in **Super-Peer Networks** such as these, the original thought was to achieve a balance between the inherent efficiency of centralised search, and the autonomy, load balancing and fault-tolerant features offered by distributed search [24]. As already mentioned in the **Peer-to-Peer** (3.1) section of the **State of the Art** (3) part of our document, *super-peer nodes* act as centralised servers for a number of regular peers, whilst these connect to each other to form an overlay network that exploits **peer-to-peer** mechanisms at a higher level. According to studies such as [27], this model is advantageously exploited in the Grid context: viewing a large-scale **Grid** as a network interconnecting small-scale subsets of it, where each one is composed by a set of nodes within one administrative domain. And within each of these subsets, one or more nodes act as **super-peers**, with the other nodes using them to access the Grid. This ensures limited network load and reduced response time with respect to pure-decentralised peer-to-peer systems.

3.2.2 Grid Resource Discovery and Structured Peer-to-Peer systems

Most **structured** approaches either (generally) adopt one DHT for all attributes [28] or they adopt one DHT per attribute [29]. While they have both proved effective, multi-DHT approaches are easier to implement and provide multi-attribute search capabilities in a simpler, more intelligible manner, while requiring more memory, thus being more expensive in the case of resources with a high number of attributes [24]. In the next two subsections (3.2.2.1 and 3.2.2.2) we cover these two methods with an example each ([28,29]).

3.2.2.1 All Attributes in One DHT

In this system, a multi-attribute set of queries uses a single one-dimensional DHT [28]. As stated in [24], a space filling curve is used, mapping all possible d-dimensional attribute values to a single dimension. So, each resource with a set of attribute values is mapped to the node whose ID is generated by interleaving the binary representations of its attribute values. As an example: a resource with three attributes (1 (**01**), 2 (**10**), 3 (**11**)) shall be stored in a node with an ID 011101. The only issue with this solution (memory-wise) is that if resources with many attributes need to be stored, and each attribute has a wide range of possible values, it will require a DHT with IDs of many bits.

3.2.2.2 One DHT per Attribute

The work done in [29], states that they extended the CAN system to allow range queries for a Grid information service. All Grid resources are thus described by a set of attributes, and for

each attribute either a standard DHT or the proposed CAN extension is used, depending on the type. Succinctly, attributes with a limited number of values are handled by the standard DHT, while "*continuous*" types of attributes use the extended CAN system. Locating resources specified by several attributes implicates that the information infrastructure queries - for each attribute in the query - the appropriate DHT and then concatenates the results in a database-like operation. This solution implicates the usage of a subset of servers that act as nodes, who store the (*attribute,resource id*) pairs, and each of them are responsible for a certain subset of the attribute values: these are called the *Interval Keepers* (*IK*).

3.2.3 Grid Resource Discovery and Semantic Information

Resource discovery in Grids aims for *best approximate* matches usable for the requester. These would be better achieved by the usage of well-defined semantic information which would be added to the resource descriptions [30]. Now, even though adding **semantic information** to resource descriptions improves the precision of a resource discovery service, it raises a problem of *semantic interoperability* [24]: it requires using common ontologies in service descriptions, and the usage of **semantic information** for precise resource discovery in large-scale, dynamic and heterogeneous environments is a new topic under research.

3.2.4 Resource Discovery in Grids

Having seen how resource discovery in Grids interrelates itself with peer-to-peer overlay networks, other classical approaches to Grid resource discovery (such as the renown work done at SETI@home [31])- which are typically centralised or hierarchical - are henceforth looked through. Specifically, we mention to two such approaches: **Hawkeye**[1] and **MDS** [32].

Hawkeye is part of the Condor Project [33]. Condor is a fully operational scheduling system, meant to maximise the utilisation of workstations, identifying idle workstations and scheduling background jobs on them. It does this by using a **centralised**, **static coordinator** which assigns background jobs, gathering system information in order to implement a long-term scheduling policy, thus knowing which jobs are waiting and which are already being executed, and the location of any idle stations. In order to prevent complete system failure, a **distributed approach** is also used, where each workstation keeps state information of its own jobs and has the responsibility of scheduling them. **Hawkeye** is designed to automate *problem detection* (such as high network traffic, or high CPU loads, or failure of resources) and *software maintenance in a distributed system*. It works by configuring Condor to periodically execute specified programs, which produce output in the form of attribute/value pairs (using Condor's *ClassAd* to identify resources). Having this, it can execute jobs based on attribute values of resources, identifying problems: as an example, if it detected a CPU load greater than fifty on machine *X*, it would kill an ongoing job such as Netscape, in order to lower that value.

The Monitoring and Discovery Service (**MDS** [32]) is an information services architecture that addresses several requirements - such as performance, security, scalability - used in the Globus Toolkit [34]. Globus is essentially a system which intends to achieve a vertically integrated treatment of application, middleware and network, building on *metacomputers* (which as the authors describe, are *networked virtual supercomputers*). So, **MDS** uses a mechanism for publishing and discovering resource status and configuration information, and a **decentralised hierarchical structure** allows it to scale, and for it to handle large numbers of resources, queues, and other types of data. Basically, three main components compose it: *GIIS* (Grid Index Information Service), a module that provides an aggregate directory of lower level data;*GRIS* (Grid Resource Information Service), which runs on resources and acts as content gateway for a resource; *IPs* (Information Providers) that interface from data collection services and then communicate to a GRIS.

3.2.5 Interrelationship with our Work

As already stated earlier, the correlation to our work is clearly defined on the **structured** approaches, where Grid resource-based discovery is held. With this in mind, we refer to the work of

[1] Hawkeye - http://www.cs.wisc.edu/condor/hawkeye

Schmidt, C. et al [28], where multi-attribute queries are used using a single one-dimensional DHT. Given the **simplicity**, **ease of use** and the **fact** that our work will not rely on a multitude of values large enough for the weakness of this system - too many attributes with too many values - to prevail, it clearly is the work of most relevance, as far as Grid resource-based searching goes. Also, the adopted solution for **load balancing** is to our liking as well, since it includes the exchange of load between neighbours, which we consider optimal.

3.3 Facial Recognition

Face recognition has been object to great study and focus in the past twenty years, and computer-based methodologies have had a particularly significant progress over the last decade. Expressly, such were these advances, that automatic face recognition evolved from research systems to a wide range of commercial products [35].

The driving factors for this focus can be portrayed into three different and complementary stimuli [36]: the aforementioned wide range of commercial and law enforcement applications, the wide availability of feasible technologies after 30 years of research, and the fact that this area attracts researchers from disparate disciplines (like image neural networks, pattern recognition, image processing, psychology, *et al*).

Although present systems offer a great deal of accuracy (i.e., verification accuracies consonant to those of fingerprint recognisers), the conditions in which these systems operate involve (generally) frontal face images taken under controlled lighting conditions.

In order to further understand the topic of face perception we shall first define this concept (see 3.3.1), then understand the human component of this topic (in 3.3.2), to later analyse the computational methods by which this is achieved (at 3.3.3). We summarise the issue correlating it to our work, highlighting what is most relevant, wrapping the subject. (see 3.3.4).

3.3.1 Defining and Understanding Facial Recognition

The following definition of what Facial Recognition encompasses is broadly accepted (as mentioned in [37], and as such we define this subject as one which comprehends the following fields: the **detection** of human faces, their **tracking**, **recognition**, subsequent ramifications of each of these steps - such as **lipreading** and **facial expressions**, and the **training** of these systems, among others.

Like other channels of human-computer interaction (but perhaps to a greater extent) which are herein enveloped with computer face perception - such as speech and gesture language recognition - as facial recognition thrives, so must the **data bases**, underlying **processing infrastructures** and other underlying **networks** which support them thrive as well. Ergo, our work in resource gathering and computation distribution is of ample avail, and quite justifiable, in this sense.

It should be further noted that the ultimate goal of automated face recognition systems is to have them at a stage where they can equal, and eventually surpass human performance [38]. This implies that the study of the human component must be broadened, as it gives great insight on the development of these computer systems.

3.3.2 Human Components in Facial Recognition

Although, as has been mentioned before, there was significant progress in improving the performance of face recognition systems in the past, much of the work put into this area seldom centres its attention to the human side to the subject. Notwithstanding the obviousness of how important the silicon component is in face recognition, the algorithms and methods used are in some way based on the understanding of psychological and neural studies - even though it is manifestly discernible that computers can not mimic all of a person's senses, at least presently.

So, the process of human face recognition is subsumed in a great diversity of studies, three of

which will be given attention hereafter: the general **psychological** approach, how a person **recognises** faces and the current **relation** of these factors with **computers**, and an analysis to the relativity in **human and computer** performance.

3.3.2.1 Psychological and Neural Approach

There are many issues related to face recognition systems that are spun off from studies in psychology and neuroscience (as seen in [39,40,36], and referenced in work such as [41,42]). We focus only on certain key aspects (summarised in [36]), as seen below:

1. **Holistic or Feature Analysis?** As seen in [43], holistic (i.e., the whole image) and feature (certain revealing features which act as determinant recognition characteristics) information are of equal relevance for face recognition.
2. **Significance of Facial Features** Studies show that facial features are of definite importance, and particularly: the eyes, mouth, face outline, hair (as stated in [43]), and nose. Another aspect known is that the upper part of the face is of better use in facial recognition than the lower part (i.e. face symmetry issues).
3. **Lighting Effects** Illumination variation continues to challenge face recognition algorithms [41], and effects such as bottom lighting (which makes identification harder) or top lighting (which eases the identification task) take their toll in the process of facial recognition.
4. **Facial Expressions and Viewpoint-Variant or Invariant Recognition** Facial expression analysis has been proven to be done in parallel to face recognition, in accordance to studies done in the neurophysiological field (seen in [37]). Also, one particularly important subject is the viewpoint [36]: staring at a static 2D photo in contrast to watching a 3D image play out in a series of images is much different.

After summarising all of these topics, one would ask:*what about **facial motion**, does it help or hinder the process of facial recognition*? Having understood the previously mentioned areas, we now delve into how a human being performs face recognition - starting off with the topic of **facial motion** - so as to understand the comparison between a person's performance with one of a computer's.

3.3.2.2 How Humans perform Face Recognition

We shall now deal with how facial motion affects human face recognition (as seen in [38]), as we clearly define various aspects (summarising many, as there are far too many to deepen) of this process and characterise them, with an effort to relate them to how the computer-based processes work, when possible. We start off with **spatial resolution**, and how it relates to humans.

Recognition as a function of Spatial Resolution

Studies (such as [40]) show that there are two main hypotheses about the role of facial motion in face recognition: the **supplemental information hypothesis** and the **representation enhancement hypothesis**.

The **former** posits that characteristic facial motions or gestures of individual faces are in effect represented together with static information, although people are presumptively assumed to rely mainly on static information for recognition. This is because the invariant structure of the face is a more reliable cue for recognition. The **latter** posits that motion adds to the quality of the structural information accessible from a human face, and as such, contributes to recognition since it eases the perception of the three-dimensional structure of a face.

And so, the most important conclusions withdrawn are that:

1. **Face Familiarity**

   The subject of **face familiarity** (studied in [44]) mediates the role of dynamic information in recognition, since these are intermittent and scarce, hence learnt in a slower fashion - ergo, experience (i.e. **familiarity**) is of a chief role since dynamic information will be better gathered and related with greater familiarity.

2. **Poor Viewing Conditions**

   The poorer the viewing conditions the more dynamic information contributes to face recognition. This is because when viewing conditions are worsened, it becomes harder to extract the facial structure, so an easier dynamic identity information is more adequate for extraction. Thus, dynamic information excels in poor viewing conditions.

Holistic versus Piecemeal Processing

As we have covered before, faces can often be identified from very little information (specific facial cues), and even one feature may suffice for identification. However, the *holistic context seems to affect how individual features are processed* (as seen in [38]): if the top-half features of a face are combined with the lower-half features of another, the holistic figure becomes unrecognisable, since both features disrupt each other.

This comes to prove that while individual cues of a certain face may suffice for its identification, in the context of a holistic figure (a face, in this case), the latter needs geometrical symmetry and logical wholeness to be able to be recognised properly. Additionally, it has been concluded that feature processing is actually interdependent with configural processing (i.e., holistic), as they both ultimately complement themselves.

As a note, the **eyebrows** (as seen by Sadr et al [45]) together with the **eyes** are actually among the most important facial cues for face recognition. Tests have shown that face recognition performance with images lacking eyebrows is significantly worse to that with the originals, and even with images which lack eyes. A possible explanation for this phenomena may be that eyebrows are very important in the conveying of emotions and other nonverbal signals.

The Nature of Cues Used

While holistic and/or piecemeal processing is very determinant, two very important issues also take their toll in face perception. These are **illumination changes** and how ***view-generalisation*** *appears to be mediated by temporal association* (as seen in [38]).

As far as **illumination changes** go, while many computer-based algorithms only perform under determinate lighting conditions (or under the combination of various illumination conditions) to be able to have robust results, there is evidence that with humans the case is not quite the same. Humans can infer different representations of a face given a determinate illumination condition. So, while we are indeed sensible to illumination conditions (how strong it is, its direction, etc), we are also capable of generalising novel illumination conditions based on previously known conditions.

Humans have an extraordinary capability of being able to recognise faces across many variations in the viewing angle: and this is a very challenging computational task for computer-based systems. And so, we are able to link the various moving correct images due to the **temporal association** which serves as "*perceptual glue*"[38], biding different images together.

Neural Foundations

To sum up the issue of how humans perform face recognition, we bring the attention of how the ***human visual system*** *devotes specialised neural resources for* ***face perception***: it is strongly believed that specialised face processing mechanisms in the human visual system (the visual cortex) do exist. The theory supporting this encompasses that the fusiform gyrus (*located in the ventromedial surface of the temporal and occipital lobes, is thought to be critical for face recognition* [36,46]) of the extra-striate visual cortex acts as the primary center for face processing, since this region shows a pattern of selectivity for human faces.

3.3.2.3 Human and Computer Performance

Although there is little work regarding the accuracy of face recognition algorithms in humans - notwithstanding, some work has shown that computer-based face recognition has surpassed human

recognition (in [47,48]) - we consider worthwhile mentioning [41], as this work clearly shows the state of modern algorithmic solutions versus human beings.

There was some previous work done on this area, more remarkably in [49], where the FRGC (*Face Recognition Grand Challenge*) is described. The work done here envisioned the encouragement of the development of facial recognition algorithms which already excel among others (verification rates equal or above 98%). The work in the previous paper hence served as an inspirational motto for the evaluation procedures used in [41], which benchmarks human performance against face recognition algorithms.

The study carried out a direct comparison between humans and seven face recognition algorithms, under **different illumination conditions** - which as stated before takes a huge toll on recognition, since it *changes the overall magnitude of light intensity reflected back from an object, as well as the shading and shadows* [41,38,50]. The effect of lighting changes is well-documented, and in fact, as far as computer-based algorithms go, controlled lighting conditions offer recognition rates much higher than those which are not: median verification rates of 0.91 versus 0.42.

The general results show that while there is an implicit assumption that human beings perform face recognition much better than machines, the experiments show otherwise. Out of the seven algorithms, for "difficult" face pairs, three algorithms were more accurate than humans (notably [51,52,53]); and for "easy" face pairs, six of the seven algorithms performed more accurately than humans. The overall conclusion is that these algorithms **compete favourably** with humans, which is already a great step, despite considerations that human beings may suffer from fatigue and/or lack of attention in these experiments (which can be lengthy and irksome).

Alas, there are many kinds of computer systems used for face recognition, but they can roughly sectioned into static-image or video processing systems. Within these, there are a lot of significant differences [36], which will be covered in the next section.

3.3.3 Classification of Computer-Based Systems

Along the document we have mentioned the different categories in which certain computer-based algorithms techniques fit in: **holistic matching methods**, **structural (feature-based) matching methods**, and the **hybrid methods**. Within these categories, different classifications can be considered, but we just mention the most relevant studies for our work in the following table (see 5 below). Afterwards, we make a brief mention to each of the categories, so as to understand their different benefits. To cinch the subject, we refer to the issue of **face detection** in particular, since part of our work passes through using an efficient method of detecting faces and subsequently, **recognising** them individually).

Table 5. Categorisation of Still Face Recognition Techniques

Approaches and Representative Work		
Holistic Methods	Eigenfaces	Turk, M.A. et al. [54]
	Fisher Weight Maps	Shinohara, Y. et al. [55]
	Probabilistic Eigenfaces	Moghaddam, B. et al. [56]
Feature-based methods	Pure Geometry Methods	Cox, I.J. et al. [57]
	Dynamic Link Architecture	Wiskott et al. [58]
Hybrid Methods	Modular Eigenfaces	Pentland et al. [59]

As mentioned in [36], **holistic approaches** have the advantage of having a reduced sensitivity to noise, and they register *a good performance under blurring, partial occlusion and changes in background* - mainly in eigenvector based approaches. As we shall see in the next section (see 3.3.4), the work done (using eigenvectors) [54] is of great importance to our work.

The aforementioned approach done by Turk, M.A. et al. [54] treats face recognition as a two-dimensional recognition problem, and is motivated by information theory, basing its functionality

on small sets of image features. These are idealised for approximating the set of known face images. The general idea is to perform several mathematical calculus (**eigenvectors**) that find the principal components of the distribution of faces, to then have a basis on which future face recognition is performed - by successively capturing, calculating, and comparing new faces to previous sets of known faces.

The **feature-based approaches** are in their genesis based on the geometry of local features, and some are very robust to illumination changes, translation, distortion, rotation and scaling, even if not widely accepted as a proper overall solution, since **holistic** or **hybrid** approaches are safer.

Such an example of these approaches was implemented by Cox, I.J. et al [57] where a **mixture-distance** approach is used. They use a novel distance function, constructed upon *local second order statistics as estimated by modelling the training data as a mixture of normal densities*. Basically, their work based itself upon statistical pattern recognition, having higher efficiency rates with a higher set of pictures of each individual.

Finally, **hybrid approaches** make use of both **holistic** and **local** approaches: the work mentioned above in [59] makes use of global eigenfaces and local eigenfeatures, making them complementary to each other. As stated in the work of Pentland et al, their work yielded *higher recognition rates as well as a more robust framework for face recognition*. While these approaches may be more complex, they may also defer great rewards.

The work performed above should also be moderately clarified: it uses a *modular eigenspace* description technique, that **incorporates features** like the eyes, nose, and mouth in an eigenfeature layer, along with a typical eigenspace technique for use in face recognition (also, under variable poses). Fundamentally, the work done scaled the classical eigenvector technique made by Turk, M.A. et al, enhancing it to larger recognition problems. Since it is view-based, it allows for recognition under varying head orientations, incorporating determinate facial features like the eyes.

Face Detection

Face detection is widely used as a preprocessor in face recognition, nonetheless being an indispensable part of it, for the most part of systems that are composed of biometric identification, video conferencing, indexing of image and video databases, and intelligent human-computer interfaces, as seen on the survey by Erik Hjelmas et al [60], on which this section bases itself upon. Mostly meant for offline processing, there are both **feature-based** and **image-based** (i.e., **holistic**) approaches.

Feature-based approaches have three sequential areas of activity. A **first low-level analysis** that deals with segmentation of visual features, using pixel properties such as grey-scale and colour. Since these can be ambiguous, a **second feature analysis** is performed at a higher level, where the knowledge of face geometry is applied to characterise and subsequently verify various features from their ambiguous state. Finally, the **third active shape models approach** is reached: these models depict the actual physical and hence higher-level appearance of features.

Holistic approaches can be roughly divided into **three** different sections. The **first** is **linear subspace methods**, like the one done by Turk, M.A. et al [54], where an error called "distance-from-face-space" (DFFS) results from calculus of eigenvectors, giving a good indication of face existence through the observation of global minima in the distance map. The **second** approach is **neural networks**, like modular architectures, committee-ensemble classification, complex learning algorithms, autoassociative and compression networks. The **third** and final approach is composed of **statistical approaches**, based on methods such as information theory, support vector machines, and Baye's decision rule.

3.3.4 Interrelationship with our Work

In our work, we intend on making use of a **holistic approach** to the solution, using the concept of **eigenfaces** (the first successful implementation of machine recognition of faces). All work previous

to [54] implementation ignored what aspects of the face stimulus were important for identification, and so, an *information theory approach of coding and decoding face images* was thought to maybe *give insight into the information content of face images.*

This approach intends to emphasise certain features, not necessarily the nose or the eyes, but the relevant information in a face image, to then encode and compare it to a database of models which were encoded similarly. As such, the principal components of the distribution of faces, or the **eigenvectors** of the covariance matrix of the set of face images are sought after. They should be thought as *a set of features which characterise the variation between face images.* An **eigenface** will represent the display of the **eigenvector** in a sort of ghostly fashion.

Naturally, there will be a training set where each **face image** can be represented as a linear combination of the **eigenfaces**. The "best" **eigenfaces** (with the largest **eigenvalues**) are those who *account for the most variance within the set of face images*, and exist due to the need for computational efficiency - where the best *M' eigenfaces span an M-dimensional subspace - "face space - of all possible images.*

In order to summarise the proceedings followed in this methodology, we now cite and enumerate the steps in [54]:

1. **Acquirement** of an initial set of face images (training set).
2. **Calculation** of the eigenfaces from the training set, *keeping only the M images that correspond to the highest eigenvalues. These M images define the face space. The eigenfaces can be updated or recalculated as new faces are collected.*
3. ***Calculation*** *of the corresponding distribution in an M-dimensional weight space for each known individual*, thus projecting the face images onto the "face space".

The previous three operations can also be performed from time to time, when there is free excess computational capacity. And with the initialisation of the system done, the following steps are used to recognise new face images:

1. First off, is the **initialisation** where the training set of face images is acquired and the **eigenfaces** are calculated, defining the **face space**.
2. When **encountering a new face image**, calculate a set of weights based on the input image and the *M* **eigenfaces** by projecting the input image onto each of the **eigenfaces**.
3. Then **determine if the image is a face at all** to check if it's sufficiently close to the **face space**.
4. **If it is a face**, then the weight pattern is classified as a known or unknown person.
5. (Optional) **If the same face is seen several times**, then its characteristic weight pattern is calculated and incorporated into the known faces, so as to learn to recognise it.

The eigenface approach is thusly free from intuitive human notions of facial parts or features, while basing itself on the notion of a small set of features that approximate the set of known face images. It is a *fast, and relatively simple* solution, that is as practical as it is capable of working adequately in constrained environments. A small caveat should be highlighted: our work bases itself not upon the study of these many and diverse algorithms, hence justifying the choice of working with a holistic, simpler method.

4 Solution's Architecture

The architecture of our solution (which we named "*P2P-Visio-Grid*") is depicted in 1. The aforementioned variables of the figure are the following:

1. **M**: Size of the video
2. **n**: Number of nodes
3. **i**: Number of human faces
4. **j**: Number of identified human faces
5. **k**: Number of face-id/identifier node pairs

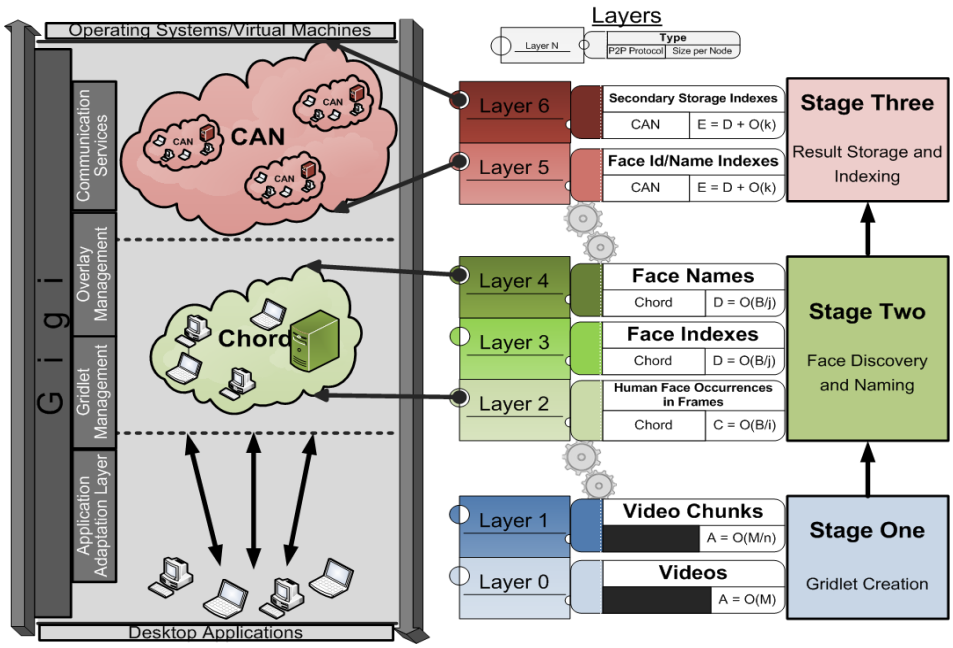

Fig. 1. Architecture design solution

Now, our work is to develop an architecture for face recognition over a series of peer-to-peer overlays, with Gigi [61] - an application (and programming model) designed for the trivial deployment of widespread applications on peer-to-peer grid infrastructures - as the substrate of it all. Following the overlay links, two main peer-to-peer overlay network protocols (see 3.1, under 3.1.3) - **Chord** [4] and **CAN** [5] - are used to exploit the distribution of data among the various layers of operation. Also, these layers are conceptually divided in stages, which we believe depict the overall functioning of the solution.

The first two layers, that belong to **Stage One**, are about the creation of **gridlets**, which are Gigi's semantics-aware chunks of data, which run on virtual machines local to each node belonging to Gigi, and that associate operations to be performed on the data as well. So, in **Layer 0**, a node would have a video which needs partitioning, and through Gigi's *Gridlet Management* - on **Layer 1** - that video is split into several fragments, each of them in a separate gridlet. Subsequently, these gridlets are exchanged with other nodes, through Gigi's *Overlay Management*.

At this point, **Stage One** is done, and the following stage is initiated, which uses Chord as a network overlay. With a subset of nodes having gridlets which represent the original video sequence, we intend on using an application code that already resides at these servicing peers. This application code will run a **Face Recognition** (3.3) algorithm based on **Eigen Vectors** [54]. So while **Layer 2** runs preliminary face detection - being **exclusively local to each node** - **Layer 3** identifies faces so that characterisation ran by eigen-vectors can be performed. In **Layer 4** (the final one of **Stage Two**), virtual clusters of nodes operate to achieve the goal of face recognition and naming. In the two previous layers, we plan on the exchange of information between clusters, and on user intervention to aid in the recognition process.

With **Stage Two** out of the way, the final **Stage Three** enters in motion. In it, this time through CAN as a network overlay and firstly in **Layer 5**, a global directory of (**face-id**, **name**) pairs is maintained - alternatively or additionally (**face-id**, **node**) pairs, which contain information about where a given face is located on a specific node. A close monitoring on the correlation between faces and names is also done, as different virtual clusters may have different results. Finally, **Layer 6** handles video-recycling: if all (or most) faces are already successfully recognised, the video can be kept in secondary storage.

With a general idea of our architecture's functioning already presented, we now look inwards, moving to an in-depth analysis of how the various layers function within and with each other. Since there is a large amount of information circulating through the various layers, a need for **metadata** is obvious, so as to have a way to better look for faces in videos. Thus we reduce the search space, using this **metainformation** gathered iteratively (by hashing video-related-data), with our main focuses being: **flexibility**, **decentralisation** and **load distribution**. With this in mind, we enumerate the steps taken in the architecture once more, relating them to the solution's layers.

Step 0: As we have many files in **Layer 0**, with different file names, a **hash[*filename*]** will return a **fileID**.

Step 1: At **Layer 1** videos are split into various internal sequential fragments, and these are scattered over the nodes. As such, the **hash[*filename*,*fragment*]**, returning **chunkID**. A **chunkID** will thus reference a set of frames: **(chunkID, frames)**. Also, for a given **fileID**, information about its corresponding chunks can be attained and passed later on to the layers upwards: **(fileID, chunkID)**.

Step 2: As application code runs at **Layer 2**, searching for human face occurrences in frames (by comparison with a predetermined training set of vectors), a set of human-face-related frames is attained. Thus, the **hash[*frame*]** will return a **frameID**. And so, a node with a **chunkID** can attain a set of human-face frames: **(chunkID, human-face frameIDs)**.

Step 3: At **Layer 3**, the previous frames are indexed, and a **faceID** (an identifier for the detected face) is collected. A set of them can be gathered, according to respective frame: **(frameID,faceIDs)**. By adjoining this information, using Chord, within each virtual cluster, a set of frames can have even more faceIDs, as the nodes cooperate with each other. Also, the same information is transmitted upwards to the fifth and sixth layers.

Step 4: Afterwards, at **Layer 4**, faces start to be recognised and assigned identifiers (either automatic processed ones, or specific names attributed by human interaction). We call such an identifier as a **personID**, thus a set of these identifiers is collected, and can be related to each **fileID**: **fileID,(personIDs)**. Again, using the network overlay (Chord), virtual clusters can adjoin the values, collecting **personIDs**, and can relate frame information with these, transmitting it to the following layer.

Step 5 and 6: Using CAN's unique characteristics, each dimension (N dimensions) within the **overall CAN overlay** (in contrast to the smaller CAN overlays that correspond to each virtual cluster) will correspond to certain characteristics stored as metadata, such as: face ids, person names, etc. Each smaller CAN overlay, as it receives information from the layers below, transmits this data to the overall CAN overlay. At this point, **data mapping** can go both ways, and from a given **fileID**, **personIDs** can be attained, and the other way around. Of course, since this metadata was collected iteratively, correlation can stretch to **faceIDs**, **frameIDs** and **chunkIDs**. As the face recognition process goes underway, metadata is propagated and correlated. So, each virtual cluster creates or re-uses information relative to *eigenfaces* in a small CAN overlay, so as to make detecting and recognising similar faces more efficient, and it is at this point that the metadata is adjoined finalising the face recognition process. Due to CAN's **locality** characteristics in node joining and localisation, similar faces are expected to be located in closely situated nodes.

5 Solution's Assessment

The assessment of our solution shall consist of three types of assessment parameters, that attempt to measure how efficient, fast, and adequate our solution is. In that sense, given the fact that the architecture spans over various layers, **qualitative**, **quantitative** and **comparative** parameters are used. We now present some parameters that fit into these kind assessment parameters.

Qualitative

1. Correct facial recognition as interpreted by user feedback.
2. False positive results as interpreted by user feedback.
3. False negative results as interpreted by user feedback.

Quantitative

1. Video processing time in terms of file dimension, in the various layers.
2. Face detection time in terms of total data dimension.
3. Face recognition time in terms of total data dimension.
4. Metadata search time.
5. Identified or recognised face search time.
6. Average load in nodes, or load imbalance among nodes.
7. Data distribution averages (metadata and/or bytes stored per node).
8. CPU usage per node.
9. Size of chunks/metadata, as distributed per node.

Comparative

1. Comparisons of times for search and processing with centralised and peer-to-peer based resource discovery architectures.
2. Comparisons of times for search and processing with cluster based resource discovery architectures.

6 Conclusions

We now wrap up the paper with some concluding remarks, which we intend on being a light overview of what was studied and referenced, and also of what we hope to achieve with our work, namely the architecture and its goals.

While resource sharing systems are used profusely in many different communities, it is peer-to-peer which has the most widespread usage among common everyday users, particularly file sharing (e.g., Gnutella, KaZaA). Albeit, participation in these peer-to-peer networks is usually dynamic, as users enter, depart and rejoin at there own free will, unpredictably. Grid systems, on the other hand, are mainly meant for interconnecting large scientific computer centre infrastructures (computer clusters, storage systems, etc). Also, they are generally used for scientific applications which are complex, CPU and time intensive, and which abide to strict QoS (Quality of Service) rules.

Adjoining the two (P2P and Grid systems) can thus be quite rewarding. By making use of the distributed, fault-tolerant and decentralised characteristics of peer-to-peer, and implementing them on Grid systems (hence making them more fault-tolerant, and lax towards their strict functioning), many studies in this field have shown that there are beneficial rewards to be reaped. In our work, we intend on doing so as well, orienting ourselves towards the process of face recognition in video, which by itself is already a complex undertaking. Naturally, processing a video, analysing its various fragments and underlying frames, to then identify human faces and recognising them is a complex process, which we hope to lighten with our work.

Finally, we intend on achieving a novel implementation of an architecture designed for face recognition in videos, which comprises of a peer-to-peer cycle-sharing system. As such, we achieve this goal through the iterative partitioning of data, and the design of a decentralised, flexible, load distribution system.

References

1. Yilei Shao and Randolph Wang. Buddynet: History-based p2p search. *Advances in Information Retrieval*, 3408/2005:23–37, 2005.
2. Subhabrata Sen and Jia Wang. Analyzing peer-to-peer traffic across large networks. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 137–150, New York, NY, USA, 2002. ACM.
3. Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.
4. I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, page 160. ACM, 2001.
5. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.

6. A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 11:329–350, 2001.
7. B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. *Computer*, 74:11–20, 2001.
8. M.J. Freedman. Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 193–206. ACM New York, NY, USA, 2002.
9. L. Xiong and L. Liu. Building trust in decentralized peer-to-peer electronic communities. In *Fifth International Conference on Electronic Commerce Research (ICECR-5)*. Citeseer, 2002.
10. M. Castro, M. Costa, and A. Rowstron. Peer-to-peer overlays: structured, unstructured, or both? Tech. Report MSR-TR-2004-73, Microsoft Research, Cambridge, July 2004.
11. B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *INTERNATIONAL CONFERENCE ON DISTRIBUTED COMPUTING SYSTEMS*, volume 22, pages 5–14. IEEE Computer Society; 1999, 2002.
12. V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A local search mechanism for peer-to-peer networks. In *Proceedings of the eleventh international conference on Information and knowledge management*, pages 300–307. ACM New York, NY, USA, 2002.
13. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *International Conference on Distributed Computing Systems*, volume 22, pages 23–34. IEEE Computer Society; 1999, 2002.
14. I. Clarke, S.G. Miller, T.W. Hong, O. Sandberg, and B. Wiley. Protecting free expression online with Freenet. *IEEE Internet Computing*, 6(1):40–49, 2002.
15. Sylvia Ratnasamy, Ion Stoica, and Scott Shenker. Routing algorithms for dhts: Some open questions. *Peer-to-Peer Systems*, 2429/2002:45–52, January 2002.
16. C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory Comput. Syst.*, 32(3):241–280, 1999.
17. Peter Druschel and Antony I. T. Rowstron. Past: A large-scale, persistent peer-to-peer storage utility. In *HotOS*, pages 75–80, 2001.
18. Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: a scalable and dynamic emulation of the butterfly. In *PODC '02: Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 183–192, New York, NY, USA, 2002. ACM.
19. M. Frans Kaashoek and David R. Karger. Koorde: A simple degree-optimal distributed hash table. In *IPTPS*, pages 98–107, 2003.
20. Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, 2000.
21. Roger Dingledine, Michael J. Freedman, and David Molnar. The free haven project: Distributed anonymous storage service. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 67–95, 2000.
22. R.Y. De Camargo and F. Kon. Design and implementation of a middleware for data storage in opportunistic grids. In *CCGrid 07: Proceedings of the 7th IEEE/ACM International Symposium on Cluster Computing and the Grid*. Citeseer, 2007.
23. Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. *Proceedings of IPTPS02, Cambridge, USA*, 2429/2002:53–65, 2002.
24. Paolo Trunfio, Domenico Talia, Harris Papadakis, Paraskevi Fragopoulou, Matteo Mordacchini, M. Pennanen, Konstantin Popov, Vladimir Vlassov, and Seif Haridi. Peer-to-peer resource discovery in grids: Models and systems. *Future Generation Comp. Syst.*, 23(7):864–878, 2007.
25. Adriana Iamnitchi, Ian T. Foster, and Daniel Nurmi. A peer-to-peer approach to resource location in grid environments. In *HPDC*, page 419, 2002.
26. Moreno Marzolla, Matteo Mordacchini, and Salvatore Orlando. Resource discovery in a dynamic grid environment. In *DEXA Workshops*, pages 356–360, 2005.
27. Carlo Mastroianni, Domenico Talia, and Oreste Verta. A super-peer model for resource discovery services in large-scale grids. *Future Generation Computer Systems*, 21(8):1235 – 1248, 2005.
28. C. Schmidt and M. Parashar. Flexible information discovery in decentralized distributed systems. In *Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing*, page 226, 2003.
29. Artur Andrzejak and Zhichen Xu. Scalable, efficient range queries for grid information services. In *Peer-to-Peer Computing*, pages 33–40, 2002.
30. David De Roure. The semantic grid: Past, present and future. In *ESWC*, pages 726–726, 2005.
31. David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. Seti@home. *Commun. ACM*, 45(11):56–61, 2002.
32. Karl Czajkowski, Carl Kesselman, Steven Fitzgerald, and Ian T. Foster. Grid information services for distributed resource sharing. In *HPDC*, pages 181–194, 2001.
33. Michael J. Litzkow, Miron Livny, and Matt W. Mutka. Condor - a hunter of idle workstations. In *ICDCS*, pages 104–111, 1988.
34. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of High Performance Computing Applications*, 11(2):115, 1997.
35. Ralph Gross, Simon Baker, Iain Matthews, and Takeo Kanade. Face recognition across pose and illumination. In Stan Z. Li and Anil K. Jain, editors, *Handbook of Face Recognition*. Springer-Verlag, June 2004.
36. W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *ACM Comput. Surv.*, 35(4):399–458, 2003.
37. Yao Hongxun, Gao Wen, Liu Mingbao, and Zhao Lizhuang. Eigen features technique and its application. In *Signal Processing Proceedings, 2000. WCCC-ICSP 2000. 5th International Conference on*, volume 2, pages 1153–1158 vol.2, 2000.
38. P. Sinha, B. Balas, Y. Ostrovsky, and R. Russell. Face recognition by humans: Nineteen results all computer vision researchers should know about. *Proceedings of the IEEE*, 94(11):1948–1962, Nov. 2006.
39. A. Mike Burton, Stephen Wilson, Michelle Cowan, and Vicki Bruce. Face recognition in poor-quality video; evidence from security surveillance. *Psychological Science*, 10(3):243–248, 1999.
40. Alice J. O'Toole, Dana A. Roark, and Hervé Abdi. Recognizing moving faces: a psychological and neural synthesis. *Trends in Cognitive Sciences*, 6(6):261 – 266, 2002.
41. A.J. O'Toole, P.J. Phillips, Fang Jiang, J. Ayyad, N. Penard, and H. Abdi. Face recognition algorithms surpass humans matching faces over changes in illumination. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(9):1642–1646, Sept. 2007.
42. John Wright, Allen Y. Yang, Arvind Ganesh, S. Shankar Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):210–227, 2009.
43. Vicki Bruce, Mike A. Burton, and Neal Dench. What's distinctive about a distinctive face? *The Quarterly Journal of Experimental Psychology Section A: Human Experimental Psychology*, 47(1):119–141, 1994.
44. D.A. Roark, A.J. O'Toole, and H. Abdi. Human recognition of familiar and unfamiliar people in naturalistic video. In *Analysis and Modeling of Faces and Gestures, 2003. AMFG 2003. IEEE International Workshop on*, pages 36–41, Oct. 2003.
45. J. Sadr, I. Jarudi, and P. Sinha. The role of eyebrows in face recognition. *Perception*, 32:285–293, 2003.
46. I. Gauthier and N.K. Logothetis. Is face recognition not so unique after all? *Cognitive Neuropsychology*, 17(1-3):125–142, 2000.

47. Xiaoou Tang and Xiaogang Wang. Face sketch recognition. *Circuits and Systems for Video Technology, IEEE Transactions on*, 14(1):50–57, Jan. 2004.
48. Xiaoou Tang and Xiaogang Wang. Face sketch synthesis and recognition. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 687–694 vol.1, Oct. 2003.
49. P.J. Phillips, P.J. Flynn, T. Scruggs, K.W. Bowyer, and W. Worek. Preliminary face recognition grand challenge results. In *Automatic Face and Gesture Recognition, 2006. FGR 2006. 7th International Conference on*, pages 15–24, April 2006.
50. A Johnston, H Hill, and N Carman. Recognising faces: effects of lighting direction, inversion, and brightness reversal. *Perception*, 21(3):365–375, 1992.
51. Chengjun Liu. Capitalize on dimensionality increasing techniques for improving face recognition grand challenge performance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(5):725–737, May 2006.
52. Chunyan Xie, Marios Savvides, and B.V.K. VijayaKumar. Kernel correlation filter based redundant class-dependence feature analysis (kcfa) on frgc2.0 data. *Analysis and Modelling of Faces and Gestures*, 3723:32–43, 2005.
53. M. Husken, M. Brauckmann, S. Gehlen, and C. Von der Malsburg. Strategies and benefits of fusion of 2d and 3d face recognition. In *Computer Vision and Pattern Recognition - Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, pages 174–174, June 2005.
54. M.A. Turk and A.P. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, pages 586–591, Jun 1991.
55. Y. Shinohara and N. Otsuf. Facial expression recognition using fisher weight maps. In *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*, pages 499–504, May 2004.
56. B. Moghaddam and A. Pentland. Probabilistic visual learning for object representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):696–710, Jul 1997.
57. I.J. Cox, J. Ghosn, and P.N. Yianilos. Feature-based face recognition using mixture-distance. In *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Conference on*, pages 209–216, Jun 1996.
58. L. Wiskott, J.-M. Fellous, N. Kuiger, and C. von der Malsburg. Face recognition by elastic bunch graph matching. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(7):775–779, Jul 1997.
59. A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 84–91, Jun 1994.
60. Erik Hjelmas and Boon Kee Low. Face detection: A survey. *Computer Vision and Image Understanding*, 83(3):236–274, 2001.
61. L. Veiga, R. Rodrigues, and P. Ferreira. Gigi: An ocean of gridlets on a "grid-for-the-masses". In *Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*, pages 783–788. IEEE Computer Society, 2007.