Distributed Middleware Architectures for Scalable Media Services

V. Kalogeraki, D. Zeinalipour-Yazti [*], D. Gunopulos

Department of Computer Science and Engineering
University of California - Riverside
Riverside, CA 92521, USA

A. Delis

Department of Informatics and Telecommunications
University of Athens
Athens, Greece

Abstract

The fusion of Multimedia and Internet technology has introduced an ever increasing demand for large-scale reliable media services. This exposes the scalability limitations of current middleware architectures, as they traditionally operate on either very large server configurations or on tightly coupled distributed systems. On the other hand, the wide availability of high-speed networks and the widespread deployment of powerful personal computing units by end users, has emphasized the advantages of the Peer-to-Peer (*P2P*) computing model. In this paper, we evaluate a number of different middleware architectures that facilitate the timely and reliable delivery of media services in *P2P* networks. Our evaluated architectures exploit features including availability of high-performance links, replication and caching of popular items and finally state-of-the-art search techniques proposed in the context of structured and unstructured *P2P* overlay networks. Through detailed simulation we investigate the behavior of the suggested *P2P* architectures for video provision and examine the involved trade-offs. We show that under realistic assumptions, the evaluated architectures are resilient to multiple peer-failures, are scalable with respect to dropped requests when the number of messages in the network increases and provide good response times to the user requests.

Key words: Video Services, Peer-to-Peer, Storage and Retrieval, Multimedia Systems.

[*] *Contact author: csyiazti@cs.ucr.edu tel: 951-827-2838, fax: 951-827-4643*
Email addresses: vana@cs.ucr.edu (V. Kalogeraki), csyiazti@cs.ucr.edu (D. Zeinalipour-Yazti), dg@cs.ucr.edu (D. Gunopulos), ad@di.uoa.gr (A. Delis).

1 Introduction

The ever improving network infrastructure in combination with the emerging peer-to-peer (*P2P*) framework offers new opportunities for distributed organization of computing systems. In this paper, we investigate the deployment of the *P2P* framework in order to offer efficient and reliable video services over a heterogeneous network of computing nodes. Thus far, most of the work in the area has concentrated in the exchange/sharing of "small" objects including MP3 music files, images, and audio. Efforts such as Seti-at-home [34] and Napster [33] along with follow-up projects including Gnutella [32], Freenet [14] and Oceanstore [25], have promoted distributed computing as an effort to share not only data objects but also CPU cycles and memory. On the other hand, prior work in furnishing video over computer networks has exclusively focused on Video-On-Demand (VOD) systems [2,7,30,22,44,12,43]. Although there have been a number of proposals, research prototypes, and some VOD products, it is evident that initial investment required for commercial use is steep. Such systems are also restricted by the number of concurrent accesses that they allow as well as load balancing issues that ensue when the demand for video streams is skewed [22,30,9].

In this paper, we build upon the approach of ad-hoc *P2P* networks of resources and evaluate new architectures that can efficiently support video-related services. The range of such services is wide and includes storage and management of movies, video-clips, and documentaries. For high quality MPEG-compressed video streams, one would approximately need 30 frames/second and therefore around 1.5Mbit/sec network connection to the source. In terms of storage requirement, for a two hour MPEG-I movie 1.35 Gigabytes are required. The corresponding figures for network transmission for MPEG-II are 8Mb/sec network bandwidth and 7.2 Gigabytes space per movie. In the context of a *P2P* infrastructure realization, two issues need to be considered:

(i) One should provide fast connections to the end user. However, this is well within reach as more individuals choose T1-level (or higher) connections both to their businesses and homes. In addition, cable and other specialized modems (ADSL, HDSL, etc.) do provide for asymmetric connections with impressive downstream rates (around 4-10Mb/sec) while maintaining significant upstream capabilities (close to 0.5Mb/sec).

(ii) The size of a reasonable population of movies can definitely increase the disk space requirements into the Petabyte area. The accommodation of such volumes calls for collaborative computing can be carried out only in a distributed setting.

Problem Statement: Movies and/or video-clips are maintained by a network of computing sites. The latter are termed "servers" and they are the computing nodes responsible for storage as well as retrieval of the multimedia elements in discussion. Via the existing networking infrastructure, servers stream requested clips and movies to user-sites for viewing and/or processing. We assume that all sites are connected via a multi-hop network. However, the key provision is that (some of the) peers may be connected via a low-latency and high-bandwidth networking option capable of effective shipment and handling of high data volumes; for instance the network could function even at the OC3 level [35]. Customers interact with the infrastructure via (thin) clients that allow for the search and display of the clips obtained from the network. The video-segments are of considerable volume –at least 0.5 Gigabytes– and are organized by storage managers. The latter operate on top of multiple disks resident within the servers' chassis. Segments and movies are all entitled, feature a number of keywords pertinent to their content, date of creation, names of producers, owner, distributor, and cast, as well as a summary of their contents and terms and conditions for the video's usage.

In the above operating environment and while observing quality of service (QoS) requirements for the delivery of multi-media data, a number of issues have to be addressed:

(i) "7x24" Operation: Services should be offered in a reliable manner even in the presence of (multiple) node failures. Once services are set up, it is expected that they will not be disabled in any significant manner unless segments of the network collapse. Users should be able to locate and view the outcome of their requests independent of the state of the participating data servers.

(ii) Transparent Evolution of the System: The evaluated solutions must deal with the dynamic aspects of the system such as arrival/departure of a server node, load-balancing in light of skewed accesses, publishing/withdrawal of video-segments by users/servers and on-line recreation of indexes in a seamless fashion.

(iii) Efficiency: The organization of video servers and distributed indexing mechanisms must allow for efficient retrieval of multimedia objects. The routing of queries in the network should avoid "flooding" of messages and comply with QoS requirements for the delivery of data.

Finally, we have to consider the implications of using different *P2P* networks. We consider both *structured* and *unstructured P2P* networks. In situations where the rate of arrivals and departures is low and the overlay topology changes slowly (structured *P2P* networks) we can either impose a global view or use some special structure which can guarantee object lookups in logarithmic time [41], thus improving the efficiency of the system. On the other hand in less stable environments (unstructured *P2P* networks) such techniques are difficult to use; instead we have to apply a fully distributed search technique [3,24,27,42,49,50].

Our Contribution: Our major contributions are:

(1) We evaluate a number of distributed architectures for delivering scalable media services. The discussed architectures, range from centralized architectures to completely decentralized architectures. Our aim is to showcase the advantages and problems of different *P2P* architectures, specifically structured and unstructured *P2P* network models, in order to facilitate scalable, timely and reliable delivery of media services.
(2) We show how the deployment of object replication and load distribution mechanisms, improve the performance of *P2P* architectures. The discussed mechanisms are instrumental for ensuring reliable operation and improving the availability and performance of the services.
(3) Finally, we perform an extensive experimental study that shows the reliability, flexibility, scalability and performance of the *P2P* architectures that we discuss. We believe that our study will enable researchers and practitioners to answer questions regarding which replication strategy or search algorithm is suitable for a given architecture. This will help improve the design and the implementation of their architectures or tune the deployed search algorithms for better performance.

2 Related Work

In [25], peer-to-peer wide-area storage facilities for files are presented. The proposed configurations target applications that include calendar organization, distributed documents, chat-rooms, and hot-listing. In [41], a look-up service based on consistent hashing for *P2P* applications is discussed. A variant of this technique appears in [1]. [46] discusses the JXTA method for generalized search and outlines ways to improve accessing documents in *P2P* networks. In [24,50], a method for intelligent retrieval of documents and nearest-neighbor search are examined. The scheme identifies the most likely sites that a query has to be. The JXTA and Hailstorm initiative intend to offer *P2P* architectures that follow a fully distributed and server-based approach respectively [28,16].

Khazana [8] uses shared "regions" of memory space among a number of LAN-based sites in a seamless manner in order to assist the delivery of multimedia. In [51], video staging is used as a tool to retrieve only parts of a video-stream from a server located across a wide area backbone network. The design of a tightly connected distributed system for the provision of Video-on-Demand and its evaluation is discussed in [19]. Techniques that improve the throughput of long-term storage subsystems for multimedia with the use of staggered striping, disk scheduling, and multi-user editing servers are presented in [7,5]. In [31], the design of a fault-tolerant VOD system that is able to overcome the failure of disk failure is described. The use of segmented multimedia objects at proxy servers is advocated in [17] to guarantee quality of service requirement for video distribution. [48] proposes a media data assignment protocol to create incentives for peers to share their bandwidth and improve peer waiting times. The use of forward error correcting codes in streaming data is used as the basis for Digital Fountain's solution to media delivery to multiple clients [21]. The functionality of BeeHive, a novel resource manager that handles transactions with quality of service requirements over the Internet, is discussed in [45].

Content Distribution Networks (CDNs) such as Akamai [4] have been successfully deployed on the Internet. These provide a better service for a given content delivery cost at the expense of deploying and managing an extensive infrastructure. On a different approach, the Narada [13] and Nice [6] application-level multicast protocols have been proposed. These can scale up to a large number of participants, but have the overhead of building and maintaining the multicast tree without taking into consideration the distinct capabilities of the peers that comprise the tree. Additionally, there is some unfairness, as nodes located higher in the multicast tree inherit all the burden of content delivery while leaf nodes don't contribute any resources. SplitStream [10] enables efficient distribution of content among nodes with heterogeneous capabilities. The distribution of load is kept fair, by splitting the multicast content into k stripes. In such a scheme a node is an interior node in the distribution of a single stripe and a leaf in the rest k-1 stripes.

In [26], the design of system modules for multi-resolution media delivery to clients with a variety of network connections is outlined. Possibly the work more closely related to ours is [49]; however, the latter mostly deals with file sharing options in the design of *P2P* applications. In contrast, our work is the first effort to the best of our knowledge that designates architectural choices for the development of *P2P* video-services with guarantees for reliability and outlines the dynamic behavior and reconfiguration of the peers as needed.

3 Architectures for Reliable Video-Services on *P2P* Infrastructure

In this section we describe a number of architectures that can render efficient video-services in *P2P* networks. [1] These architectures, which range from completely centralized to fully decentralized, reflect a wide range of options currently available for providing Media Services in a networked environment.

[1] An earlier version of these architectures was presented in [23].

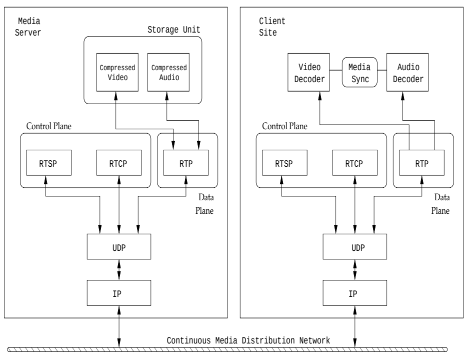


Fig. 1. Configuration for Continuous Media Distribution

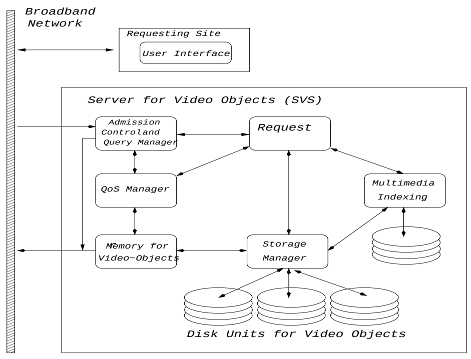


Fig. 2. The SVS Baseline Architecture

The described architectures serve as a basis for the queuing models presented in Section 4 and the experimental evaluation of Section 5.

We assume that the role of a server is essentially that of a “storage and streaming” point in the various network configurations discussed. As soon as a server is ready to furnish a video clip/movie that a client machine has requested, we assume that the appropriate communication protocol, such as RTP, is in place to facilitate timely delivery of continuous data. We build upon a significant amount of work in this area [15,39,40,47,51,26].

Fig. 1 outlines a possible layout for the delivery of multimedia objects from a server “near” a requesting client site. The Internet Protocol (IP) facilitates video streaming at the network-layer. The UDP transport protocol offers the network transport functions [15] and as long as it functions in dedicated networks, data loss remains negligible [29]. The RTP (real-time transport) and RTCP (real-time control) protocols support real-time delivery and control functionalities for QoS (Quality of Service) respectively. More specifically, RTP offers source identification, time-stamping, sequence numbering, and payload identification for streaming data [39]. RTCP is the control protocol that functions in parallel with RTP to provide feedback on the quality of data received and membership information for data. RTCP offers QoS sender/receiver reports on the quality of data transported, participant identification, scaling packets, and inter-media synchronization [39]. A session control protocol such as the RTSP [40] defines the messages and the functions that control the delivery of continuous data in the context of an established session. RTSP assist in the streaming of continuous data in a number of distinct ways: a) establishes control streams for video and audio data between the server nodes and client sites b) allows choosing the delivery channel including unicast UDP and multicast UDP, and c) supports fundamental VCR-like operations including pause, fast-forward, fast-backward, stop, play, and resume.

The architectures we present next are designed to take advantage of the specific characteristics of different *P2P* network models. Two main approaches have emerged for constructing *P2P* networks: *structured* and *unstructured* networks. Structured P2P are networks organized in such a way that the data objects are located at specific nodes in the network, and nodes maintain some state information to enable efficient retrieval of the objects. In this case a global distributed index of all the objects in the network can be created. Structured *P2P* networks are better suited for environments where the rate of peer departures, arrivals and failures (churn) is low. Unstructured *P2P* networks on the other hand can consist of any number of peers. These make no assumptions about the location of the objects or the availability of the peers. More advantages of unstructured *P2P* networks include their ability for self-organization, for adaptation to different loads and for resiliency to node failures. Due to the possibly high rate of peer departures, arrivals and failures, these cannot be used to create any kind of global organization.

3.1 **Sole Video Server (SVS)**

In this architecture, a single video server maintains all movies and/or clips existing in the network. In addition, the server maintains the necessary metadata for the objects in discussion as well as the necessary indexing mechanisms for easier retrieval of data. Upon check-in, users interact with the system via a thin-client interface that essentially provides the IP address of the server. By contacting the server, users are able to browse and/or search for clips and/or movies of interest. This functionality is achieved with the following three commands/messages:

- `query`: this is the primary mechanism for searching the server's content. Should a number of matches are found, the server will notify the query-originating site accordingly. Otherwise, a message indicating an empty result will be dispatched by the server.
- `query_reply`: provides an enumeration of matching multimedia objects that comply with the constraints imposed by a `query`. In the case where no such outcome exists, the query reply consists of an empty message.
- `download`: initiates the transfer of an object from the server to the requesting site.

Users are able to request video-objects by sending the name of the object, or terms of the object's name (via the `query` message) for look-up to the server. We assume throughout the paper that the name of each multimedia object is unique, and is used as a key. The server can use traditional database indexing structures such as B-Trees (for indexing the file date and size), or inverted files (for indexing filename terms) to efficiently index the objects. The dynamics of the server's object population can be handled easily as inclusion of new multimedia items occurs in a single location. Updates imposed on the indexes can be carried out in a straightforward manner at the server.

Fig. 2 depicts two sites -a server and a requesting peer- in the SVS architecture. The communication substrate that connects the various sites is assumed to have the capability of delivering video streams. As discussed earlier, a combination of appropriate protocols (such as the RTSP-RTCP-RTP/UDP/IP) helps in transporting the required data payloads in a timely fashion to requesting sites. As the number of concurrent queries increase, the server will ultimately suffer from problems such as missing QoS guarantees, experiencing delays at disk units, service disruptions in light of bursty workloads, etc. In order to avoid such situations, the peer with the video storage features an *Admission Control and Query Manager*. The latter in collaboration with the *Quality of Service* module overlooks the rate by which frames arrive in main-memory and are getting ready for shipment over the network. Peer requests for downloads of streams are only accepted if the QoS parameters of these streams can be guaranteed [2,36]. The `query` and `query_reply` messages are handled by the *Admission Control and Query Manager* in cooperation with the *Storage* and *Multimedia Indexing* modules. The ultimate delay observed by a user depends on the number of movies concurrently requested by different sites as well as the utilization of resources at the server peer and the network.

There are obviously no reliability guarantees in the SVS architecture. Once the server, the source of all video-objects, fails there is no alternative for fetching data. It is worth pointing out the high level of required initial investment in order to realize such a configuration. Clearly, this architecture cannot be classified in our *P2P* context, but it serves as baseline for our discussion and experimental work.

3.2 Single/Multiple Index site(s)–Multiple Servers (SIMS/MIMS)

In this architecture, a number of independent *data*-servers constitute the basis for managing the storage and retrieval of video objects. Each *data*-server can only hold up to a number of video objects and for each such object, the server counts the number of accesses and the time of last request for that object. Initially, the number of the participating data servers in the network can be considered fixed. However, this restriction can be easily relaxed as soon as the data-server connection protocol is introduced by nodes in the system. This model is essentially the Napster [33] model.

Data-Server Connection Protocol: Upon initialization, the data-servers upload the meta-data for each of their video objects to some indexing server (see Fig. 3). As mentioned earlier, such meta-data includes titles of segments, producers, ownership, cast, release dates, etc. The uploaded meta-data uses a *soft state* protocol in which objects are removed from the indexing server unless their data-server owner sends periodically a refresh message indicating in that way that it is still alive and willing to serve the object. The *indexing* server can be either a single node ($SIMS$) or multiple nodes ($MIMS$). In $SIMS$, all data-servers select the same index server while in $MIMS$, nodes select several different indexing servers.

Ultimately, the indexing servers maintain distributed “hooks” regarding the movies and clips stored in the server peers. One could envisage such hooks as triplets including an object-ID (associated with search terms in the *Multimedia Indexing* module), an IP address (in which the object is resident) and a port number (through which the object is fetched). The final service provided by the “indexing” node is the brokering of connections between site of users and data-servers. These approaches provide the SIMS/MIMS architecture with an advantage over the SVS architecture in which all video objects have to be stored in the same server. Although the main advantage of MIMS is that it avoids overloading a single indexing node in the system, it also significantly increases the number of messages required to maintain the index entries.

Client Connection Protocol: The clients of the system connect upon initialization to any of the available data-servers in the network. The initial connection can occur in a number of ways: 1) by a simple selection by the user's client program (e.g. connect to a server that a client was connected in the past), 2) by proximity of geographic location (e.g. by using the binning scheme proposed in [37]), or 3) by exploitation of simple yet effective load-balancing heuristics that could take into consideration number of concurrent outgoing streams in a specific server, number of connections, availability of main-memory, and utilization of resources. The above process is facilitated by two protocol messages:

- `ping`: discover server peers in the network.
- `pong`: any server site that obtains a `ping` is expected to reply with `pong` messages. Information about the load, number of concurrent operations per server, geographic location, as well as number of open connections could be piggybacked to the requesting peer.

Search Protocol: We assume that once a connection is established, it remains open throughout the entire time of the interaction regarding a video object and that a user can search for available movies/clips that are located on that server. If a video object is located at the data-server then the video is directly streamed back to the user. On the other hand, if the video object the user is looking for is not located on the data-server, the server consults its associated indexing server using the following search mechanism:

- `query_index`: questions the *Indexing Node* for the existence of multimedia objects.

- `query_index_reply`: sends back either distributed hooks for sought video objects or a null message indicating no result.

Upon receiving the `query_index_reply`, a data-server can follow one of the following strategies:

(i) *First Cache–Then Deliver:* The item is first cached to the data-server and then distributed back to the client that initiated the request. The condition that enables such a copy is that an object has become popular. The latter is quantified by constraints that indicate that an object has received $\delta\%$ of the most recent Δ requests. The *Indexing Node* has to also be alerted to this effect. Therefore the client will perform the object download from its data-server rather than the original holder of the object. Transparent replication of popular objects is also performed in the Freenet system [14], with the difference that popular objects get replicated on all intermediate nodes from the owner to the requester.

(ii) *Forward Object*: The data server managing the item streams the object via the network directly to the requesting site. In this case, users download copies from the original holder of the object and the data-server of a client is skipped.

If more than two servers can furnish the sought data objects, simple heuristic policies can be applied depending on the least loaded peer, proximity of the user to peer (as this is manifested by the number of hops needed in the network), as well as on-going traffic at the segments of the network. This information could be easily provided to the *Indexing* site with piggybacked messages from the respective data-servers. To efficiently index the objects, the indexing node can use the same techniques with the SVS architecture (B-trees and inverted files). The indexing structures associate with the name of each object the nodes that have this object. Since the index is centralized, updates can also be handled efficiently by the indexing node.

Replication and Caching Protocol: To ensure reliable operation of the network in light of node failure and/or time outs, we propose a simple yet effective replication policy that calls for the mirroring of an object in the network to at least another additional node. The selection of the new node can be done randomly or based on load-balancing heuristics. In conjunction with the possible caching and subsequently floating copies (due to First Cache–Then Deliver operation above), the system will be able to recover from more than two site failure. As peers reach their capacity, there is obviously need for house-keeping by doing garbage collection. Objects can be eliminated in a least recently requested object fashion and provided that at least two copies exist in the network. The latter can be determined by issuing a `query_index` to

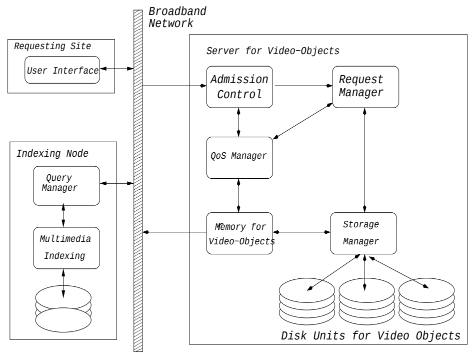


Fig. 3. SIMS/MIMS Architecture.

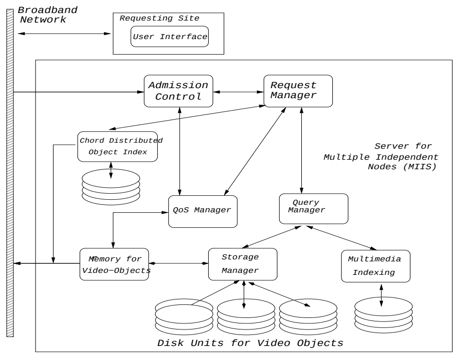


Fig. 4. Multiple Independent Indexing Servers (MIIS).

the indexing node and receiving more than two object hooks. The additional messages needed by the SIMS model are:

- `download(siteIP)`: this message either downloads directly an object or transparently generates a copy from the peer bearing the object and then continues with the downloading.
- `updateIndex(indexIP)`: sent by a peer that has updated its database.

New videos/clips can join the network by having a site publish available data objects and following an admittance policy. In this paper, we are not concerned with the latter as this policy is the mechanism that gauges the service content of a particular *P2P* network. Assignment policies of the video/clip to a peer server may range from completely random assignment to heuristics based on the intention to keep utilization as minimal as it is possible. As soon as a home server is determined and the appropriate storage takes place, the *indexing* node has to be provided with the meta-data of the newly arrived segment. Subsequently, the access structures have to be refreshed in order to allow for accurate retrieval.

3.3 Multiple Independent Indexed Servers (MIIS)

In this architecture, clients maintain their own multimedia objects which makes them in that way also data-servers. Because of this symmetric role, we will refer to these nodes as *peers*.

The MIIS architecture is similar to the SIMS/MIMS architecture, the main difference being that in MIIS each *peer* features *distributed* access structures that give them the opportunity to efficiently retrieve objects of interest located at other peers. As a result, in the description of the MIIS architecture we focus on the areas that it differs from SIMS/MIMS.

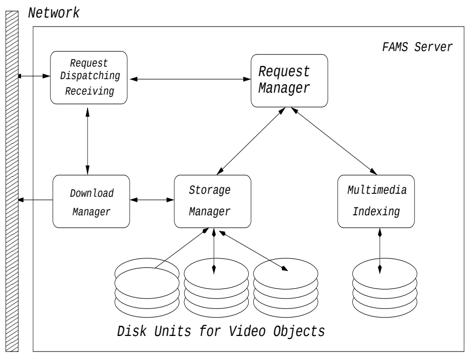


Fig. 5. Fragmented And Multiple Servers (FAMS).

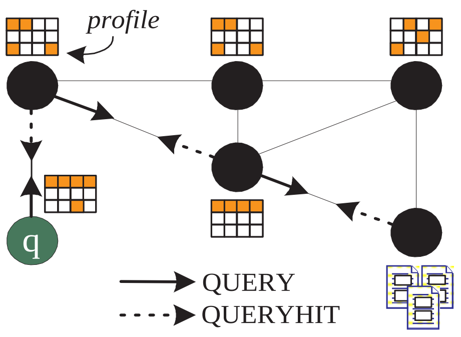


Fig. 6. Searching in FAMS using the Intelligent Search Mechanism (ISM).

Specifically, the MIIS architecture utilizes a scalable and distributed index to provide means for efficient indexing of object names and object lookup in logarithmic time. There are many techniques for building a distributed index, such as Chord [41], CAN [38] and Tapestry [52]. In the description we focus on the Chord Algorithm. This architecture is designated for environments where either the rate of *peer* arrivals and departures is moderate (structured *P2P* networks), or for environments where the available communication bandwidth is sufficient for keeping the *distributed* access structures in a consistent and updated state.

Peer Connection Protocol: Chord provides a mechanism for mapping a key to a node; in MIIS, the key is the name of the object, and the mapping is used to identify one node that is responsible for keeping the locations of all copies of the object in the network. Chord uses *consistent hashing* in which peers and objects are hashed upon initialization into an *m-bit key* and then each object key k is assigned to the first node n that is equal to or follows k (n is called the successor of k). Object keys are organized in key circle of numbers in the range $[0..2^m - 1]$ and the first node clockwise from a node k is called the *successor* of k. One advantage of using consistent hashing is that with high probability all nodes receive approximately the same number of keys. Furthermore, Chord minimizes the number of key redistributions in the network when the network dynamically expands or contracts by using $O(log^2 N)$ messages.

Fig. 4 depicts this architecture term Multiple Independent Indexing Servers (MIIS). The storage peers feature modules pertinent to quality of service delivery of data including *Admission Control*, *QoS Manager* and *Memory and Download Manager*, query and storage managers as well as the Chord Distributed Video Object Index. Peers, as in the SIMS/MIMS configurations, maintain open connections with all their peers as the assumption is that their networking substrate displays low-latency and high bandwidth characteristics.

Search Protocol: To efficiently retrieve the multimedia objects based on the name of the object, each node first hashes the name of the object. It then utilizes Chord to locate the node that contains the key of the object it is looking for. A node maintains a *finger table* which contains pointers to several nodes. More specifically the i^{th} entry in the finger table contains the identity of the first node s that succeeds n by at least 2^{i-1} on the keys circle and object lookups can efficiently be resolved with only $O(logN)$ messages.

Although Chord was initially proposed for lookup operations based on the object identifier of an object (e.g. filename) recent work in [18] shows that content-based query resolution is also feasible. More specifically the framework proposes the registration of the content (i.e. attribute-value pairs that describe the content) at *Rendezvous Points RP*s. Queries might then be routed, using Chord, to a predefined set of *RPs* which consequently resolve the query.

Replication and Caching Protocol: The MIIS architecture features, similarly to the SIMS/MIMS architecture, a transparent replication protocol of popular objects. It is invoked when an object o receives $\delta\%$ of the most recent Δ requests. A *peer* then downloads and caches o locally and further updates the Chord entry which is located at some node in the network (i.e. the successor of o). Using this mechanism we can guarantee that the network always contains k copies of object o, since each node can check (through the successor of o) the number of copies before removing the o from its local storage.

3.4 Fragmented And Multiple Servers (FAMS)

This scheme follows the fundamental design choices of the Gnutella [32] model, in which a large number of low-bandwidth nodes (i.e. dial-up users) form an adhoc overlay network structure which serves as the communication medium among them. In this type of architecture, there are no global indexing sites for the multimedia objects and no peer has a global view of the system at any particular moment [20]. Each node is essentially a video-service provider, which maintains a local set of video clip files along with their meta-data, and a client which actively seeks for new video objects. This architecture places the smallest possible requirements to each peer. Since no centralized or distributed index is maintained by the system, peers are free to join and leave the network independently, thereby creating an unstructured *P2P* network. There is also no upper limit on the number of participating peers and that the peers are not fully connected. In addition, there is no explicit quality of service modules on the peers and timely delivery of multimedia objects relies solely on restricting the number of respective connections per peer. Such an architecture has no scalability limitations, like the centralized SVS architecture and the SIMS/MIMS, because there is no global index to be maintained. Fig. 5 depicts the organization of a peer.

Peer Connection Protocol: A node becomes a member of the network by establishing a connection with at least one peer in the network (often no more than 3-5). This number is typically limited by the peer's resources and the type of network links that exist between the peer and the rest of the network. The connectivity information (i.e. IP and port) of the initial neighbors is either retrieved from a *hostcache*, which maintains a partial list of active nodes, or from a local list of nodes to which the peer was connected in the past. The next step is to actively discover other nodes in the network as current nodes might leave at any point of time. This is achieved by using at regular intervals the following messages:

- `ping`: sent to all neighbors requesting to discover other nodes/peers in the network.
- `pong`: furnishes information about the nodes which are willing to accept new incoming connections. Such information includes, the IP-address, ports available, types of links, and some aggregate type of meta-data that outline the data stored locally.

Basic Search Protocol: To search the network for an object, a node can send a `query` message to all of its peers including a "constraint" that essentially articulates the search operation. Typically this constraint is a set of keywords (meta-data) attempting to outline what is being sought. A peer receiving a `query` message evaluates the constraint locally against the meta-data in its own local storage manager. If the evaluation is successful, the peer will generate a `reply` message back to the originating node which includes the object(s) corresponding to the constraint. Otherwise, a new `query` is initiated from the peer in discussion to the nodes in its own *Peer List*. If the segment is available in the network, this recursive operation guarantees ultimate retrieval. However, in practice we limit the depth of the recursion by using some *Time-to-Live* (*ttl*) parameter, which determines the maximum number of hops that the given lookup should be forwarded. The *ttl* parameter, which is used in many networked applications, starts out from a pre-defined value and decreases each time the lookup message is forwarded until *ttl* becomes zero. The *ttl* parameter avoids flooding the network with messages.

The described protocol can be summarized with the following messages:

- `query`: this message designates in terms of a string specific information about a sought movie and/or video clip. In addition, it can specify requirements for the nodes to be searched for the information.
- `query_reply`: an enumeration of video/clip objects that comply with a `query`.

Once the object is located, a user initiates either a download request from the respective peer. If the original holder of the file is behind some firewall (and hence not accessible from outside users), the requester issues an `upload` request, which includes the IP and port on which the requester can accept the requested object. This procedure can be summarized with the following messages:

- `download`: this initiates the transfer of an object from the requesting site to the peer-server.
- `upload`: this initiates the transfer of an object from a peer-server to the requesting site.

Improving Search Performance: In order to minimize the overwhelming amount of messages that are generated by querying all neighbors (also known as *message flooding* or *Breadth-First-Search (BFS)*), a node can utilize a number of improved search techniques that are based on local knowledge and that were recently proposed. It is important to mention that distributed lookup indexes, such as Chord, are not appropriate in the context of the FAMS architecture as transient user populations will impose a large number of repair operations (i.e. key redistributions) which are extremely expensive for low-bandwidth nodes [11].

Examples of alternative approaches include *Random BFS* [24,50], in which a node probabilistically queries a random subset of neighbors. If a node can keep information on the result of past queries, the FAMS architecture can also employ other search techniques, including *>RES* [49] in which a node queries the neighbors that returned the most results in the recent past, the *Random Walkers* [27] approach, in which the query is forwarded to one random neighbor at each hop, or the *APS* [42] approach, in which a node utilizes feedback from previous searches to probabilistically guide future walkers (rather than forwarding the walker at random). A node can also employ the *ISM* [24,50] mechanism in which a node intelligently queries a subset of neighbors that returned the most similar answers in the past. The ISM mechanism (see fig. 6) is an efficient, scalable yet simple search technique designated for unstructured *P2P* networks. It consists of four components: A *Profiling Structure* which logs queryhit messages coming from neighbors, a *Query Similarity* function which calculates the similarity, *RelevanceRank* which is an online neighbor ranking function and a *Search Mechanism* which forwards queries to selected neighbors. Using ISM, a node continuously monitors queryhit messages coming from neighbors and builds using this information some local knowledge about its neighbors. If the query locality is high (i.e. similar queries are repetitively posted) then the ISM mechanism performs extremely well.

Recent studies in [50] show that by using ISM and some of the other presented search techniques in random graphs, a node can significantly reduce the number of messages and time used by the brute-force technique while retaining high degrees of recall (i.e. finding the expected objects).

Assumptions in a Realistic Environment: The fact that the topology of the network is not known and can change dynamically over time creates both problems and opportunities. Every node only knows about its first-line peers, but in light of site failures, the overall system can still function (almost certainly with longer response times). In order to make our FAMS model more pragmatic, while providing video services, we impose the following two assumptions:

(i) We disallow `downloading` of multimedia objects through low bandwidth connections that may appear in *Peer-List*s if a faster connection is available. Although a connection to a peer may exist, it might be not viable in order to sustain the presumed quality of service requirements.

(ii) We assume that any video segment is available from at least two peers in line with the reliability rule suggested in the SIMS/MIIS architectures. This policy permits "new releases" to have at least one replica randomly created in some node in the *P2P* network. The `downloading` option allows for further caching of objects and propagation of their corresponding meta-data.

4 Experimental Methodology

In this section, we discuss our experimental methodology of the presented peer-to-peer architectures for video-service provision. By employing a number of different workloads, we have estimated performance indicators and carried out sensitivity analyses. In this context, our main goals were to:

- investigate the "average" behavior of the suggested configurations in the presence of uniform and skewed requests.
- examine the reliability features of each architecture and the effect of the proposed replication/caching policy as well as to experimentally gauge the levels for continuous operation despite failure of multiple nodes.
- carry a competitive scalability analysis and quantify the number of requests unable to be serviced by every architecture.
- estimate the resource requirements and evaluate the timeliness guarantees of each architecture.

In order to carry out our experimental objectives, we developed detailed queuing network models for all architectures (discussed in the following subsection) and based on those models we created four extensive simulation packages. The software was developed in C++ and the size of the packages ranges from 2k-2.5k lines of source code; the packages run on the Linux RedHat 7.1 distribution. The key parameters used across all simulation packages are outlined in Table 1 and their definition is self-explanatory.

Table 1
Key Simulation Parameters

Parameter	Definition
NumPeers	Number of Servers (Storage-capable Peers)
NumObjects	Population of Multimedia Objects in *P2P* Network
ConsPServer	Number of Connections allowed Per Server
FracObjects	Fraction of Objects Allocated to a Server
RepliDegree	Degree of replication of an Object
NumIndxSites	Number of *indexing* sites
VicinityObjs	Number of Objects in the Vicinity of a Server (partial indexes)
NetworkType	Type of network (random, fully connected)
RandNetDgr	Degree of Random Network
ServOpConn	Maximum Number of Concurrent Connections allowed per Server
RespTime	Response Time for downloading an Object

In the SVS architecture, we used a single server that operates both as indexing and source of all the video objects in the network. Fig. 7 depicts the queuing model for SVS. The streaming server runs on a powerful machine and has the capability of storing all movies and clips whose number is set to 1,000. Users request objects by dispatching `query` messages to the server. The *Admission Control and Query Manager* determines whether a new user request can be accepted based on the site's communication capacity (maximum number of downloads it can support) and the number of movies that are currently being streamed. User requests are routed to the network via the *Request Queue* and arrive at the *Admission Queue* of the server. If the request gets accepted by the *Admission Control and Query Manager*, the request is queued at the *Query queue* of the *Request Manager*. The *Admission Control and Query Manager* works in collaboration with the *QoS Manager* to ensure QoS guarantees for the accepted requests. Subsequently, the request is queued at the *Multimedia Indexing* component that is responsible for finding the movie at the movie server. If the movie is found locally, the *Admission Control and Query Manager* creates a *Query Reply* message that describes the matching results and is queued at its *Reply queue*. If there are no matching results, the query reply message consists of an empty message. The message travels through the network and reaches the *Reply queue* at the client side. For simplicity, all the queues in the model use the FIFO discipline.

To initiate the transfer of the movie/clip object, the client creates a *download* message which passes through the *Admission Queue* and the *Query Queue* and finally arrives at the *Download Queue*. For each user request, the server dynamically creates a new thread that starts the user session. The movie is finally displayed at the user interface.

Fig.8 shows the queuing network for the SIMS/MIMS architecture. Users obtain the movie directly from the local server, if it is available. Otherwise, a `query_index` message/request is routed to the network via the *Request Queue*. At the *Indexing node*'s site the request is placed on a *Query queue* while awaiting processing. The *Indexing node* identifies which streaming peer maintains a

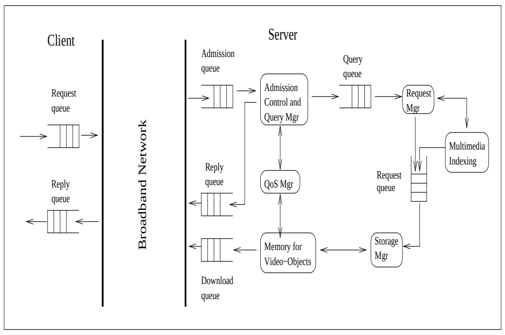


Fig. 7. Queuing Model for SVS Architecture.

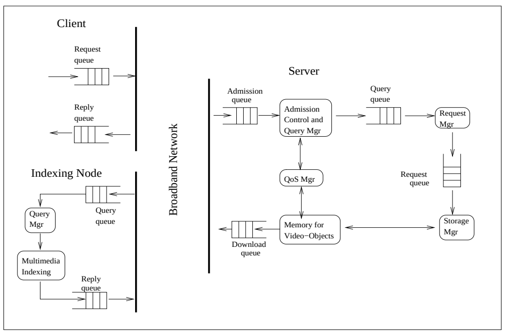


Fig. 8. Queuing Model for SIMS/MIMS Architecture.

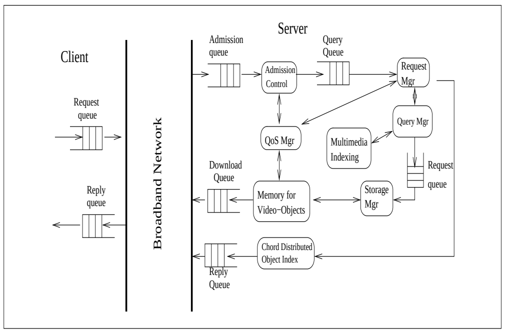


Fig. 9. Queuing Model for MIIS Architecture.

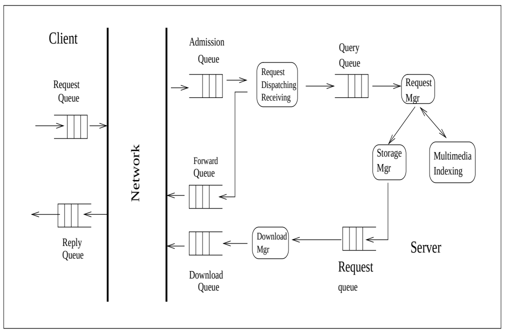


Fig. 10. Queuing Model for FAMS Architecture.

copy of movie object in question and notifies the client with a recommendation of such a server through the *Reply queue*. This recommendation is based on load balancing policies obtained through piggybacked messages that maximize the probability that the user request will be met. Subsequently, the user issues a `download` message which travels through the network and reaches the *Admission Queue* of the appropriate streaming peer. As soon as the *Admission Control Manager* accepts the query, it forwards it via the *Request Manager* and *Storage Manager* to the *Download queue* for streaming.

In our baseline experiments with the SIMS/MIMS architecture, we used 10 dedicated streaming servers connected to one *indexing* server. We assumed that one *indexing* server was sufficient to support all incoming requests. To guarantee reliable operation in the network despite failures of the *indexing* site, we assume that there is a second *indexing* site functioning as a backup unit. Since the users are serviced by the dedicated servers even if the primary *indexing* site fails, object streaming is not interrupted (as opposed to SVS). If `query_index_reply` acknowledgments have not been received after the elapse of a predetermined time-out period, users have to re-issue `query_index` messages. The availability of additional *indexing* servers will benefit the SIMS/MIMS architecture as it can contribute to a noteworthy reduction in their response time. In our experiments, 1,000 multimedia objects were randomly distributed among the servers. Each object was replicated twice, so each server maintained in total 200 movies/clips. 10,000 requests for movies/clips were made to the servers. All the servers have equal capacity in terms of CPU, networking and storage. The servers are connected only to the *indexing* server, and are independent of each other. The *Admission Control* manager determines the number of concurrent object streams the server can support. We assume that all streaming peers are identical and therefore can accept the same number of connections.

Fig.9 depicts the queuing network for the MIIS architecture. A user's request, through the client's *Request queue*, is forwarded to the network and finally arrives at the *Admission queue* at the client's local server. If the *Admission Control* component accepts the user request, the request is queued at the *Query queue* of the *Request Manager*. The functionality of the *Request Manager* is modified so that it also searches the network using Chord, as described in section 3.3, for objects not locally available. Therefore, the *Query request* first goes through the *Query Manager* component and if a match is not found there, the *Request Manager* searches the network using Chord. If a match is found, there are two options: the server can either cache the movie locally before it starts downloading to the client or can forward the Query request to its server peer. In the first case a node is also required to update the Chord index that contains the keys for the cached movies. Then, the movie starts getting transmitted at the client machine.

While experimenting with the MIIS architecture, we assume 10 servers, fully connected to each other. There is no centralized *indexing* node; each peer maintains its own local index along with its multimedia objects. In our experiments, each server maintains 200 multimedia objects. During the operation of the system, popular movies are cached to more servers for future requests, so the replication degree for some of the objects increases. All the streaming servers have similar resource features. The server's *Admission Control* module controls the number of concurrent data streams allowed.

Fig. 10 shows the queuing network for the FAMS architecture. When the user submits a *Query Request* through her/his interface, the request is forwarded to user's closest streaming peer. Streaming peers are essentially user machines and they can be highly diversified. Therefore, there is no admission control module in the FAMS queuing network. Requests received by servers are placed at *Query Queue* of the *Request Manager* for evaluation. If the retrieval is successful and the request can be satisfied locally, the server peer creates a *Query Reply* message which via the *Reply Queue* is sent to the querying node. In addition, the server dispatches the request through the *Forward Queue* to its own peers in the network.

The FAMS model reflects a fully distributed architecture; there is no single indexing or dedicated storage server. Essentially, each server in this architecture represents an end-user machine. In our main body of experiments, we assume 100 such peers; the formed network among the peers constitutes a random graph with average degree varied from five to twenty (for brevity, we report results only for graphs average degree ten). We randomly distributed 1,000 multimedia objects among the sites with the only condition being that objects are replicated twice during their onset in the *P2P* network. In total, 1,000 users make requests for objects. FAMS servers are the least powerful in comparison with all other architectures.

We ran experiments with 10,000 requests for continuous objects. We have used as the main threshold value of user requests to be 10,000 because for this value and for values above the average remains very similar. In all configurations, the requests "arrive" sequentially. Each time interval, a random user selects a movie and submits the request. We used two distributions to model the movie selection. In the first case, the distribution of the requests is uniform. In the second, one tenth of the multimedia objects are popular movies and represent half of the requests. The other half of the requests is uniformly distributed to the rest of the objects. The popular objects are randomly distributed to the servers. Also, we assumed that all the multimedia objects have the same size, and take the same time to download. We varied this download duration of the multimedia objects between 100 and 1,000 time units in different experiments. Each movie/clip download, keeps one server connection busy. In all architectures, we assume that there is an upper limit in the number of users that can be simultaneously served by single site. This number is largest in the SVS architecture (typically 50 to 200), smaller in the SIMS/MIMS and MIIS architectures (10 to 20) and smallest in the FAMS architecture (2 to 10).

5 Experimental Evaluation

Search Performance: In the first set of experiments, we evaluated the efficiency of each of the architectures. This was done by measuring the average number of messages that have to be exchanged between users and servers for each request before downloading commences. Here, we do not consider the time it takes for a server to search its index. For datasets in the range of $10^4 - 10^6$ movies/clips, we expect the logarithmic search time to be reasonably short compared to the communication time among servers.

It is evident that the SVS architecture calls for the smallest number of messages (two messages) to start downloading: the user sends a `query` message to the server, the server searches through its index and if any matches are found sends a `query_reply` message that includes the corresponding results. The user initiates the transfer of the movie by issuing a `download` message.

In the SIMS/MIMS architecture, the number of messages also remains limited. Upon login, a user sends a `ping` message to connect to one of the well-known servers in the network. The payload of the message contains the user's IP address. The server's *Admission Control* manager uses the location of the user, along with the current number of downloads in the system, to determine whether it can accept the request in question. The server replies with a `pong` message only if the *Admission Control* manager on that node is willing to accept the connection. If the movie/clip is available, the download process can begin. Otherwise, the user queries the *indexing* node for movies/clips in the network. Therefore, for each user request, only `two` or `four` messages are needed to find a movie. If the user request is denied by the *Admission Control* manager and the user re-issues his request, the latter is viewed as an entirely new request.

Fig. 11 shows that the average number of messages per user request in the MIIS architecture.[2] To search for a movie, the user sends a `query` request to its local server. If the peer has the movie, or has its location in the local index, then it replies with a `query_reply` message that contains the location of the movie for download. On the other hand, if the server has no information about the location of the movie, it has to initiate a search in the network. The figure also shows that the average number of messages decreases with the number of user requests over time. As more users request movies from the server, the server get populated with more copies of replicated objects, which therefore results in fewer peer lookup messages. The replication algorithm will be described in further detail in the next subsection.

The average number of messages per user request is much higher in the FAMS architecture (Fig. 12) compared to the other architectures. The reason is that in this architecture there are no dedicated servers where the users find movies/clips. Also, since there is a larger number of servers in the network, each peer has only a partial index of the movies of its "vicinity". When the server has zero-knowledge of the location of a multimedia object, it initiates a Breadth-First-Search in the network. As a result, the number of messages propagated in the network is large. Similar to the MIIS architecture, the performance of the FAMS architecture improves as the server "learns" about the location of the movies in the network by using the ISM search mechanism [24,50]. However, the improvement is small because the server learns about only a small proportion of the movies that are available in its peers (compared to the MIIS architecture, and for the same replication degree of the movies).

[2] In the experiments with the MIIS architecture we did not implement Chord or a similar distributed index since the number of nodes was small. Instead each node sends query messages to all its peers.

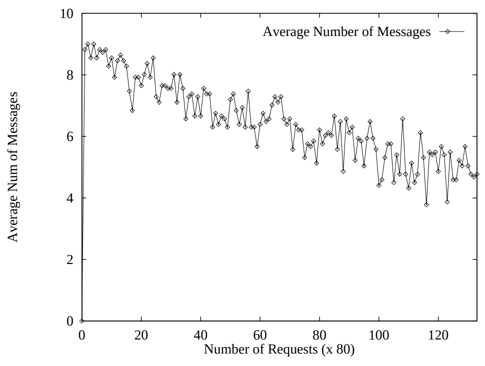


Fig. 11. MIIS Architecture: Average number of messages per request to find an object. (10 servers, 1,000 movies/clips, degree of replication 2.)

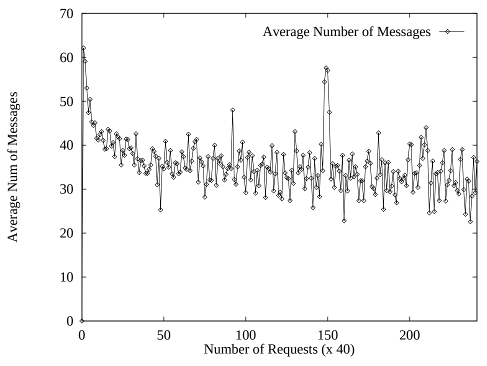


Fig. 12. FAMS Architecture: Average number of messages per user request to find an object. (100 servers, average connectivity 10, 1,000 movies/clips, degree of replication 2.)

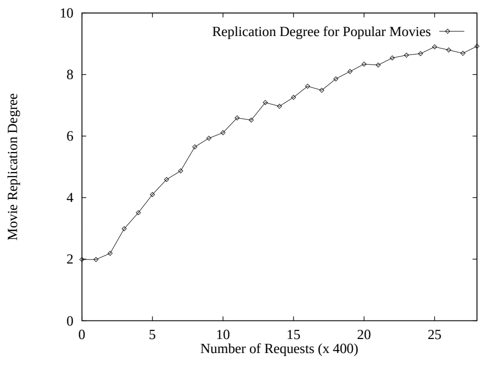


Fig. 13. MIIS Architecture: Average Replication Degree for Popular Movies. (10 servers, 1000 movies/clips, 100 popular movies/clips, initial replication degree 2.)

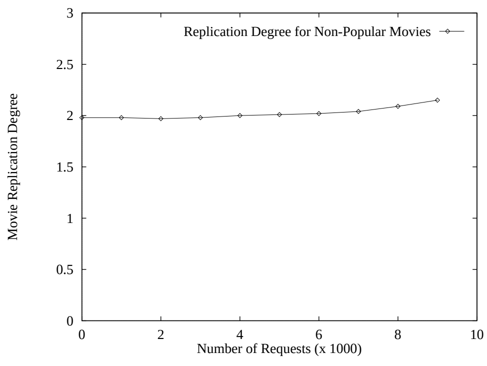


Fig. 14. MIIS Architecture: Average Replication Degree for Non-Popular Movies. (10 servers, 1000 movies, 900 non-popular movies, initial replication degree 2.)

Clearly the SVS and SIMS/MIMS architectures have the best search performance. However our experiments show that the performance of the MIIS architecture approaches that of SIMS/MIMS as the local indexes index a larger number of movie locations. On the other hand, the performance of the FAMS architecture is quite lower than the other three.

Replication Algorithm: In the second set of experiments, we evaluated the performance of our replication algorithm. We ran experiments in the context of the MIIS architecture. Our goal was to investigate the replication degree of the popular and the non-popular movies as the number of requests increases.

Fig. 13 depicts the average Replication Degree of the popular movies in the MIIS architecture as a function of the user requests. At first, the replication degree for all movies is 2. If a server peer receives more than three requests for a movie not locally available, it caches the multimedia object. The latter is an

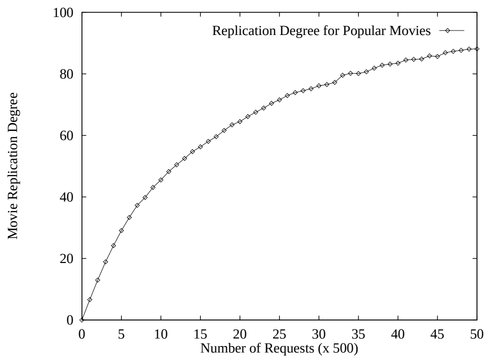


Fig. 15. MIIS Architecture: Average Replication Degree for Popular Movies. (100 servers, 10,000 peers, 10,000 movies/clips, 1,000 popular movies/clips, initial replication degree 5.)

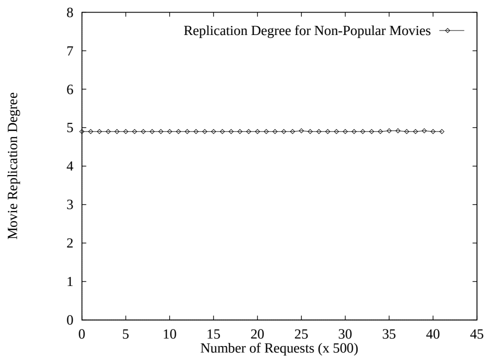


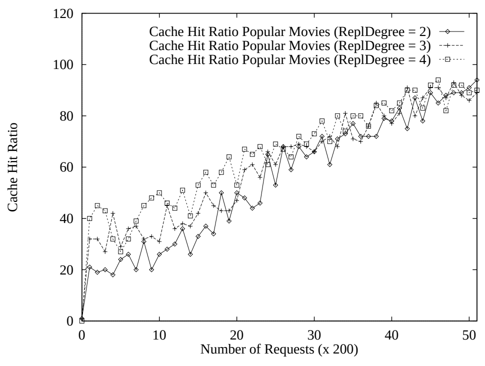
Fig. 16. MIIS Architecture: Average Replication Degree for Non-Popular Movies. (100 servers, 10,000 peers, 10,000 movies, 9,000 non-popular movies, initial replication degree 5.)

indicator that the clip/movie in discussion is a popular one. As Fig. 13 shows, the replication degree of the popular movies increases quickly. Eventually, most of the Servers cache a copy of the popular movies. As a result, the number of messages needed to find a movie decreases continuously.

Fig. 14 shows the average Replication Degree for the non-popular movies. Our experimental results indicate that only a few peers cache locally a copy of the non-popular movies. The reason is that these movies are requested less frequently and a maximum of four replicas seems to be sufficient to satisfy the user requests.

We verified the above results by running experiments in a larger network with 100 dedicated movie servers, 10,000 movies/clips and 10,000 peers. Figures 15 and 16 show the average replication degree for the popular and the non-popular movies respectively. Fig. 15 indicates that although we start with a small number of popular movies in the network, the replication degree of the popular movies increases quickly and eventually most of the servers obtain a copy of these movies. For the non-popular movies we notice (Fig. 16) that the original replication degree does not change with the user requests. Therefore, five replicas for the non-popular movies seem more than enough to satisfy the user requests. Both of these results validate our earlier results (Figures 13 and 14) where we used a smaller network.

In this set of experiments, we did not run the corresponding experiments for either SIMS/MIMS or FAMS. In the first, the algorithm cannot be directly applied as we assume that servers are not directly connected with each other. So the servers cannot cache additional copies of objects. In the second, we note that the connections from a graph with relatively low degree, and servers are not connected to most of the other servers directly. On the other hand, to do



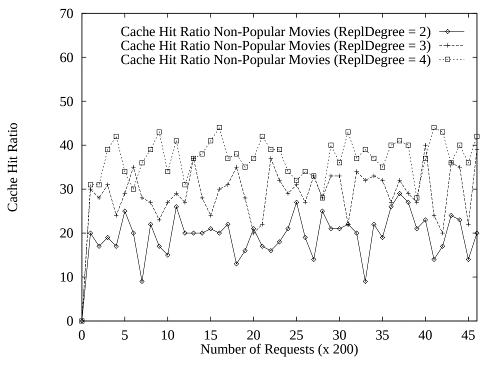


Fig. 17. MIIS Architecture: Cache Hit Ratio for Popular Movies. (10 servers, 1000 movies/clips, 100 popular movies, initial replication degree 2.)

Fig. 18. MIIS Architecture: Cache Hit Ratio for Non-Popular Movies. (10 servers, 1000 movies/clips, 100 popular movies, initial replication degree 2.)

the replication efficiently, we would have to open direct connections between servers, and change the topology of the network dynamically.

We also measured the Cache Hit Ratio for the popular and non-popular movies over different replication degrees. Fig. 17 shows that the Cache Hit Ratio for the popular movies increases with the replication degree of the movies and the number of user requests. For example, at user request 4000 and at movie replication degree four, the Cache Hit Ratio is 60%. The Cache Hit Ratio for the popular movies increases as more servers cache copies of the popular movies. This reduces the number of messages in the network and the response time to the user requests.

Fig. 18 shows that the improvement of the Cache Hit Ratio for the non-popular movies is smaller and remains stable as the number of user requests increases. The reason is that these movies are less frequently requested by the users, so there is small benefit in caching them in many nodes. The experimental results suggest that a replication degree of at most four is sufficient to meet all user demands.

Reliability: To evaluate the reliability of the different architectures, we measured the number of movies that are no longer available in the system as servers fail. Clearly, the SVS architecture provides no reliability guarantees; if the server fails, then the system stops operating. In the remaining architectures, when only one server fails no objects are lost. The reason is that each movie is replicated twice or more, so there is at least one more server that has a copy of the movie. Naturally, if we expand the initial degree of replication, we can guarantee tolerance to multiple failures (at least proportional to the degree of replication allowed).

Fig. 19 shows the number of movies lost in the SIMS/MIMS architecture when 20% servers (2 out of 10) fail over various replication degrees for the movies.

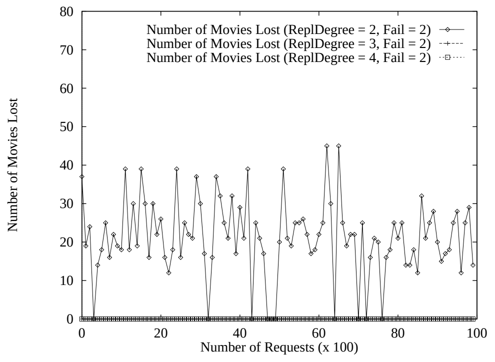


Fig. 19. SIMS/MIMS Architecture: Number of movies lost when 20% of the Servers fail at movie replication degrees 2, 3 and 4. (10 servers, 1000 movies.)

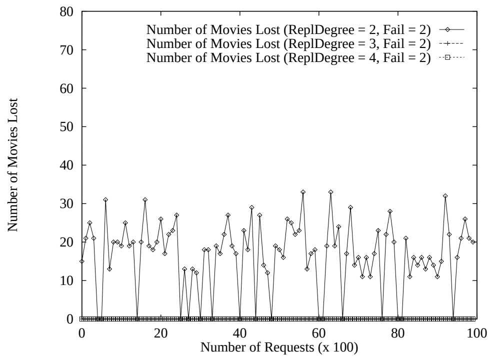


Fig. 20. MIIS Architecture: Number of movies lost when 20% of the Servers fail at initial movie replication degrees 2, 3 and 4. (10 servers, 1000 movies.)

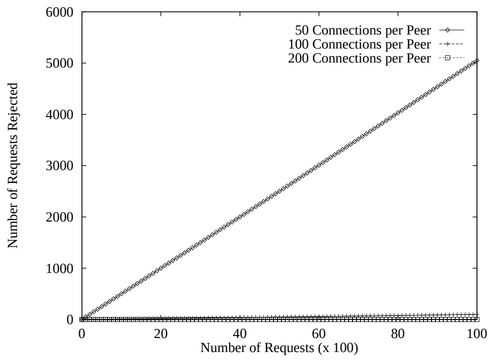


Fig. 21. SVS Architecture: Number of Requests Rejected for Maximum Number of Open Connections 50, 100 and 200.

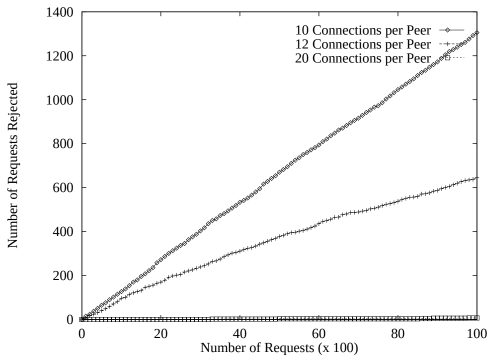


Fig. 22. SIMS/MIMS Architecture: Number of Requests Rejected for Maximum Number of Open Connections 10, 12 and 20. (10 servers, 1000 movies, replication degree 2.)

For example, at replication degree = 2, less than 5% of the movies are lost. The reason that we loose movies in this case is because both the servers that have a copy of the same movie may fail. As the replication degree for the movies increases, no movies are lost. In the MIIS architecture, the number of movies lost is initially the same compared to SIMS/MIMS for the same replication degree and same number of faulty servers, as shown in figure 20.

Scalability: In the fourth set of experiments, we evaluated the scalability of the architectures by measuring the number of user requests that each architecture rejects. The *Admission Control* manager of the Server node rejects a request when the maximum number of connections allowed in the mode is exceeded. The *Admission Control* manager decides whether a new user request is accepted based on the communication capacity of the server (maximum number of downloads it can support) and the number of multimedia transfers that currently take place.

In all the figures in this section we report the number of dropped requests cumulatively, so at the i^{th} request we show the total number of requests that were dropped from the 1^{st} request to the i^{th} request (including the i^{th} request if indeed it was dropped).

In the SVS architecture, we varied the maximum number of (open) connections that the SVS can support from 50 to 200. These connections correspond to the number of movies that are concurrently being downloaded from the server. Fig. 21 shows the number of rejected requests for the SVS architecture. When the maximum number of connections that the server supports is 200, then a large number of user requests are rejected (50% of the requests) from the server. The first requests are always accepted by the server, but as more requests are made, the *Admission Control* manager denies more requests. The results can improve if the Server becomes more powerful to accept more user requests. But still, this illustrates the scalability limitations of the SVS architecture.

Fig. 22 shows the number of rejected requests for the SIMS/MIMS architecture. To simulate the fact that peers are less powerful (than the SVS), the server can facilitate 10 to 20 concurrent open connections. Our results indicate that the architecture can service 87% of the requests (approximately 1,300 rejections) when the maximum number of connections per server is 10. Also, the results show that as the maximum number of connections for the server increases to 15 and more, almost no requests are rejected.

In Fig. 23 we show the number of failed requests for the MIIS architecture over different number of open connections. The duration of each movie was set to 100 time steps (i.e., when a server accepts a request for a movie download, has to keep this connection on for 100 time steps). The architecture can service 88% of the requests (1,200 rejections) when the maximum number of open connections per server is 10, but 95% when this number raises to 12 and almost 100% for 20. We observe that the SVS architecture would have to be able to keep at least 95 connections open at the same time in order to be able to service 95% of the requests.

In Fig. 24, we show the number of failed requests in FAMS. Since the servers in this architecture are likely to be less powerful, we assume 5, 7 or 10 maximum number of simultaneously open connections per server. Allowing only 2 connections per server results to 2,000 dropped requests. However, increasing the limit (connections) quickly reduces the number of dropped requests, and in fact even allowing 5 connections per server results in 0.25% requests rejected.

Response Time: In the last set of experiments, we evaluate the timeliness guarantees of the proposed architectures by measuring the average response time per user query request. The response time depends on the following

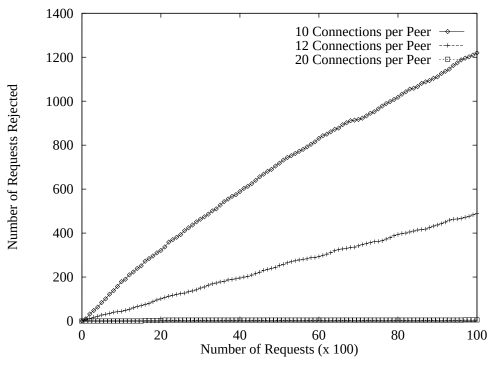


Fig. 23. MIIS Architecture: Number of Requests Rejected for Maximum Number of Open Connections 10, 12 and 20. (10 servers, 1000 movies, initial replication degree 2.)

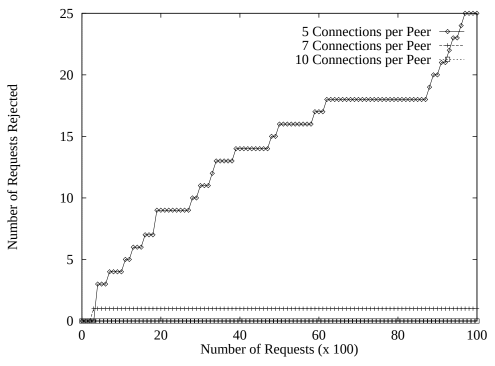


Fig. 24. FAMS Architecture: Number of Requests Rejected for Maximum Number of Open Connections 5, 7, and 10. (100 servers, avg. connectivity 10, 1000 movies, replication degree 2.)

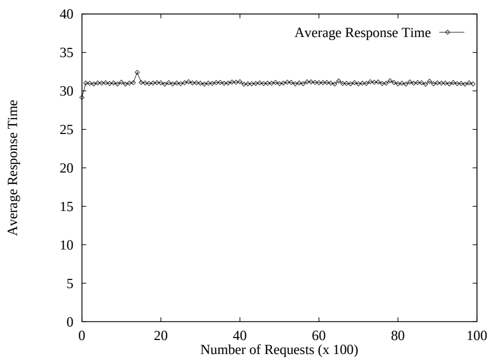


Fig. 25. SVS Architecture: Average response time per request (in milliseconds) to start downloading an object, 10000 peers, maximum 200 connections per server, 100,000 movies/clips.

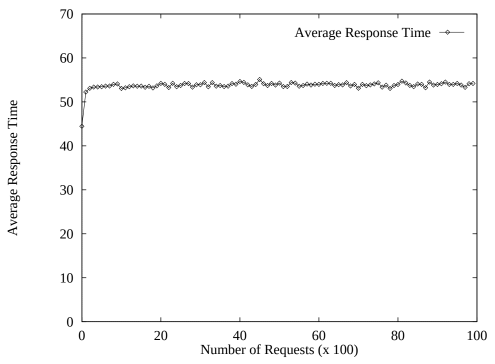


Fig. 26. SIMS/MIMS Architecture: Average response time per request (in milliseconds) to start downloading an object, 10 servers, maximum 100 connections per server, 10,000 peers, 100,000 movies/clips, degree of replication 3.

factors: (1) the time for the user to send the query message to the indexing server, (2) the time for the server to evaluate the query locally against the movies in its database, (3) the time for the server to initiate a search request, if the movie is not locally available (in the MIIS and FAMS architectures), and (4) the time for the transfer of the movie clip to begin.

Fig. 25 shows the average response time for our baseline architecture, the SVS architecture. The figure indicates that the response time is very small, in the range of 30 milliseconds. The SVS architecture has minimal end-to-end delay because it drops a large number of user requests (as shown in Section 4.4). But if a request is accepted by the *Admission Control Manager*, the SVS architecture guarantees very small end-to-end delay.

Fig. 26 depicts that the response time per user request for the MIIS architec-

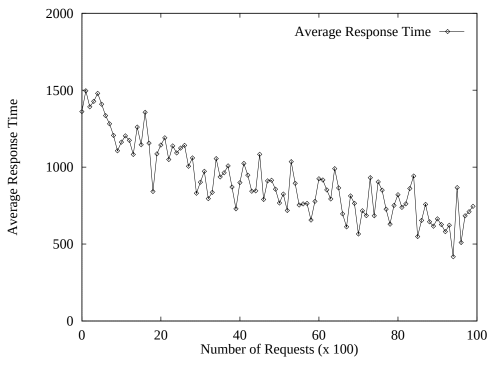


Fig. 27. MIIS Architecture: Average response time per request (in milliseconds) to start downloading an object, 10 servers, 1000 peers, maximum 10 connections per server, 1,000 movies/clips, degree of replication 3.

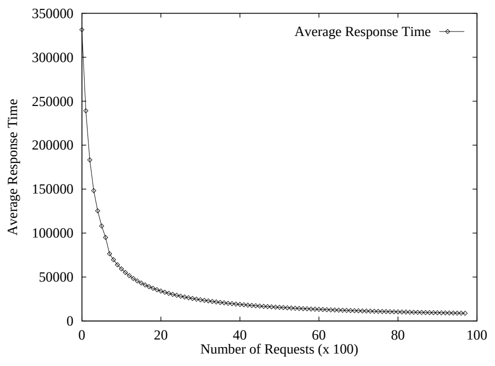


Fig. 28. FAMS Architecture: Average response time per request (in milliseconds) to start downloading an object, 100 peers, 1000 movies/clips, maximum 10 connections per peer, degree of replication 10.

ture. The overall rates are higher compared to the SVS architecture although they remain relatively short in the order of 52 milliseconds. This value represents the average response time only for the requests that are accepted by the *Admission Control Manager*. In general one can argue that compared with its SVS counterpart, the SIMS/MIMS architecture offers a better performance as it can service a larger number of user requests with minimal extra delay.

Fig. 27 shows the average response time for the MIIS architecture. The figure indicates that the average response time is consistently higher compared to the rates of SVS and SIMS/MIMS. The reason is that if the server cannot find the movie locally available, it will send search requests to its server peers and wait until it receives a hit. It is noteworthy that not only the reliability but also the performance of the MIIS architecture improves with the number of user requests over time. As the users request more movies, the server caches copies of the popular movies (as shown in Section 4.2) instead of searching in the network and therefore the average response time decreases. To estimate an upper bound for the average response time of the MIIS architecture, we also run experiments with a larger network (100 servers, 10,000 peers, 100,000 movies, 50,000 use requests). Our experiments indicated that the response time does not increase above 2,500 milliseconds.

Fig. 28 depicts the average response times attained in the FAMS architecture. Clearly, this constitutes the worst performance. There are two reasons: (1) these servers are actually diverse type machines (which can be less powerful than the dedicated movie servers and with lower bandwidth connections), and (2) each server has to initiate either an ISM search each time it needs to find a movie from its server peers. Consequently, FAMS is unable to offer any timeliness guarantees.

6 Conclusions and Future Work

The timely and reliable streaming of video-on-demand, clips, and audio files to end-users may yield gains for a wide range of fields including telemedicine, news delivery, digital libraries, enhanced security of public and private venues, and distance learning. The emerging peer-to-peer (*P2P*) distributed computing protocols in conjunction with ever increasing network capabilities offer new and exciting opportunities for the aforementioned application areas. In this paper, we have sought to take advantage of these developments and have evaluated a number of distributed software architectures whose objective is to offer the required support for the provision of video service on *P2P* networks.

The experimental evaluation shows that FAMS has a number of unique advantages. It is likely to be the most inexpensive option as no particular computational features are required of the participating sites. It also provides a fault-tolerant environment that is not likely to be affected by the failure of a peer. Obviously, SVS is on the other side of the spectrum. It is our belief however that schemes similar to MIIS will benefit the area of multimedia delivery the most as the close collaboration of some dedicated storage peers helps in the timely and efficient delivery of multimedia objects. It is worth mentioning that MIIS requires more powerful peers than FAMS, and better reliability guarantees, in order for the distributed index to work. On the other hand the advantages, which include a much faster response time, and a better utilization of the peers' resources, are worth having. In our experiments, all the distributed architectures (SIMS/MIMS, MIIS, and FAMS) have shown impressive reliability and scalability furnished by only a small degree of replication.

In the future, we plan to extend our work by pursuing a number of issues. They include, use of just-in-time pre-staging of multimedia objects for improved adherence to quality of services requirements in *P2P* architectures; investigation of virtual peer clustering for the facilitation and better support of specialized interest requests/groups; examination of low-entropy protocols for updating meta-data structures/lists; and finally, suggesting load sharing and replica snooping techniques using more effectively the notion of vicinity in *P2P* architectures in order to further maximize overall throughput of object requests.

References

[1] K. Aberer, M. Punceva, M. Hauswirth, and R. Schmidt. Improving Data Access in P2P Systems. *IEEE Internet Computing*, 6(1):58-67, January/February 2002.

[2] S. Adali, K.S. Candan, S.-S. Chen, K. Erol, and V.S. Subrahmanian. Advanced Video Information Systems. *ACM Multimedia Systems Journal*, 4(4):172-186, 1996.

[3] L.A. Adamic, R.K. Lukose, A.R. Puniyani, and B.A. Huberman. Search in Power-Law Networks. Technical report, Xerox Parc Research Center, http://www.parc.xerox.com/istl/groups/iea, Palo Alto, CA, 2000.

[4] Akamai Home Page. Akamai.Com. http://www.akamai.com.

[5] W.G. Aref, I. Kamel, and S. Ghandeharizadeh. Disk Scheduling in Video Editing Systems. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):933-950, 2001.

[6] S. Banerjee, B. Bhattacharjee and C. Kommareddy. Scalable Application Layer Multicast. *Proceedings of ACM SIGCOMM'02*, Pittsburgh, PA, August 2002.

[7] S. Berson, S. Ghandeharizadeh, R. R. Muntz, and X. Ju. Staggered Striping in Multimedia Information Systems. In *Proceedings of the 1994 ACM SIGMOD, Minneapolis, MN, May 24-27, 1994*, pages 79-90.

[8] J. Carter, A. Ranganathan, and S. Susarla. Khazana: An Infrastructure for Building Distributed Services. In *Proceedings of the 18th IEEE Int. Conf. on Distributed Computing Systems*, Amsterdam, The Netherlands, May 1998.

[9] S.R. Carter, J.F. Paris, S. Mohan, and D.D.E. Long. A Dynamic Heuristic Broadcasting Protocol for Video-On-Demand. In *Proceedings of the 21st IEEE Int. Conf. on Distributed Computing Systems*, Phoenix, CA, May 2001.

[10] M. Castro, P. Druschel, A-M. Kermarrec, A. Nandi, A. Rowstron and A. Singh, SplitStream: High-bandwidth content distribution in a cooperative environment, IPTPS'03, Berkeley, CA, February, 2003.

[11] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, S. Shenker Making gnutella-like P2P systems scalable. In *Proceedings of ACM SIGCOMM 2003*, ACM Press, 2003, pp. 407 - 418.

[12] M.S. Chen, D.D. Kandlur, and P.S. Yu. Storage and Retrieval Methods to Support Fully Interactive Playout in a Disk-Array-Based Video Server. *ACM Multimedia Systems Journal*, 3(3):126-135, July 1995.

[13] Y. Chu, S. Rao, S. Seshan and H. Zhang. A Case for End System Multicast. *IEEE Journal on Selected Areas in Communications (JSAC)*, 20(8), 2002.

[14] I. Clarke, O. Sandberg, B. Wiley, T.W. Hong. Freenet: a distributed anonymous information storage and retrieval system *Proceedings of the ICSI Workshop on Design Issues in Anonymity and Unobservability*, 2001, pp. 46-66.

[15] D. Comer and D. Stevens. *Internetworking with TCP/IP:Volume III Client-Server Programming and Applications.* Prentice-Hall, Englewood Cliffs, New Jersey, BSD Socket Version edition, 1993.

[16] Microsoft Corporation. Hailstorm Software Architecture. http://www.microsoft.com/net/hailstorm.asp.

[17] H. Fahmi, M. Latif, S. Sedigh-Ali, A. Gafoor, P. Liu, and L.H. Hsu. Proxy Servers for Scalable Interactive Video Support. *IEEE Computer*, 34(9):54-60, September 2001.

[18] J. Gao and P. Steenkiste. "Design and Evaluation of a Distributed Scalable Content Discovery System." *IEEE J. on Selected Areas in Communications*, Special Issue on Recent Advances in Service Overlay Networks, 22(1):54-66, 2004.

[19] L. Golubchik, R.R. Muntz, C.-F. Chou, and S. Berson. Design of Fault-Tolerant Large-Scale VOD Servers With Emphasis on High-Performance and Low-Cost. *IEEE Transactions on Parallel and Distributed Systems*, 12(4):363-386, 2001.

[20] Z.J. Haas and S. Tabrizi. On Some Challenges and Design Choices in Ad-Hoc Communications. In *Proceedings of IEEE MILCOM*, Bedford, MA, October 1998.

[21] G.B. Horn, P. Knudsgaard, S.B. Lassen, M. Luby, and J.E. Rasmussen. A Scalable and Reliable Paradigm for Media on Demand. *IEEE Computer*, 34(9):40-45, September 2001.

[22] J. Hsieh, M. Lin, J.C.L. Liu, D.H-C. Du, and T. Ruwart. Performance of a Mass Storage System for Video-On-Demand. In *Proceedings of the IEEE INFOCOM Conference*, Boston, MA, 1995.

[23] V. Kalogeraki, A. Delis, and D. Gunopulos Peer-to-Peer Architectures for Scalable, Efficient and Reliable Media Services, In *Proceedings of IEEE IPDPS 2003*, IEEE Computer, 2003, p. 29b.

[24] V. Kalogeraki, D. Gunopulos, and D. Zeinalipour-Yazti. A Local Search Mechanism for Peer-to-Peer Networks, In *Proceedings of ACM CIKM 2002*, ACM Press, 2002, pp. 300 - 307.

[25] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of ASPLOS*, Cambridge, MA, 2000.

[26] R. Lienhart, M. Holliman, Y-K. Chen, I. Kozintsev, and M. Yeung. Improving Media Services on P2P Networks. *IEEE Internet Computing*, 6(1):73-77, January/February 2002.

[27] Q. Lv et al., Search and Replication in Unstructured Peer-to-Peer Networks, In *Proc. of ICS 2002*, ACM Press, 2002.

[28] Sun Microsystems. Jxta. http://www.jxta.org.

[29] S. Milliner and A. Delis. Networking Abstractions and Protocols Under Variable Length Messages. In *Proceedings of the 1995 IEEE International Conference on Network Protocols (ICNP-95)*, Tokyo, Japan, November 1995.

[30] B. Özden, A. Biliris, R. Rastogi, and A. Silberschatz. A Disk-Based Storage Architecture for Movie On Demand Servers. *Information Systems*, 20(6):465-482, 1995.

[31] B. Özden, R. Rastogi, P.J. Shenoy, and A. Silberschatz. Fault-tolerant Architectures for Continuous Media Servers. In *Proceedings of the 1996 ACM SIGMOD*, pages 79-90.

[32] Gnutella Home Page. Gnutella.Com. http://www.gnutella.com.

[33] Napster Home Page. Naspter.Com. http://www.napster.com.

[34] SETI Project Home Page. SETI@home. http://sethiathome.ssl.berkeley.edu.

[35] C. Partridge. *Gigabit Networking.* Addison-Wesley, 1993.

[36] P.V. Rangan, H.M. Vin, and S. Ramanathan. Designing an On-Demand Multimedia Service. *IEEE Communications Magazine*, 1(1):56-64, 1992.

[37] S. Ratnasamy, M. Handley, R. Karp, S. Shenker. Topologically-Aware Overlay Construction and Server Selection In Proceedings of IEEE INFOCOM 2002, New York, USA, 2002.

[38] S. Ratnasamy, P. Francis, M. Handley, R.M. Karp, S. Shenker A Scalable Content-Addressable Network In Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, 2001, San Diego, CA, USA. Pages 161-172, ACM, 2001

[39] H. Schulzrinne, S. Carter, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. Technical report, Internet Engineering Task Force, RFC 1889, January 1996.

[40] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). Technical report, Internet Engineering Task Force, RFC 2326, April 1998.

[41] 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 ACM SIGCOMM Conference*, San Diego, CA, August 2001.

[42] D. Tsoumakos and N. Roussopoulos, Adaptive Probabilistic Search for Peer-to-Peer Networks, In *Proceedings of P2P 2003*, IEEE Computer Society, Pages 102-110, 2003

[43] C. Vassilakis, M. Paterakis, and P. Triantafillou Video Placement and Configuration of Distributed Video Systems Based on Cable TV Networks, *ACM/Verlag Multimedia Systems Journal*, vol. 8, no. 3, pp. 92-104, March 2000.

[44] M. Vernick, C. Venkatramani, and T. Chiueh. Adventures in Building the Stony Brook Video Server. In *Proceedings of the Forth ACM International Conference on Multimedia*, Boston, MA, November 1996.

[45] A. Victor, J. Stankovic, and S. H. Son. QoS Support for Real-Time Databases. In *IEEE Workshop on QoS Support for Real-Time Internet Applications*, Vancouver, BC, June 1999.

[46] S. Waterhouse, D.M. Doolin, G. Kan, and Y. Faybishenko. Distributed Search in P2P Networks. *IEEE Internet Computing*, 6(1):68-72, January/February 2002.

[47] D. Wu, Y.T. Hou, W. Zhu, H.-J. Lee, T. Chiang, and Y.-Q. Zhang. On End-To-End Architecture for Transporting MPEG-4 Video Over Internet. *IEEE Transactions on Cirtcuits and Systems for Video Technology*, 10(6):923-941, September 2000.

[48] D. Xu, M. Hefeeda, S. Hambrush and B. Bhargava. On Peer-to-Peer Media Streaming. *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'02)*, pp. 363-371, Vienna, Austria, July 2002.

[49] B. Yang and H. Garcia-Molina. Comparing Hybrid Peer-to-Peer Systems. In *Proceedings of the 27th International Conference on Very Large Data Bases*, Rome, Italy, Pages 561-570, 2001.

[50] D. Zeinalipour-Yazti, V. Kalogeraki and D. Gunopulos. Exploiting Locality for Scalable Information Retrieval in Peer-to-Peer Systems *Information Systems Journal*, Elsevier Publications, Volume 30, Issue 4, Pages 277-298, 2005.

[51] Z.L. Zhang, Y. Wang, D.H.C. Du, and D. Shu. Video Staging: a Proxy-server-based Approach to End-to-End Video Delivery over Wide-area Networks. *IEEE/ACM Transactions on Networking*, 8(4):419-442, August, 2000.

[52] B.Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment, In IEEE Journal on Selected Areas in Communications, Vol 22, No. 1, January 2004.