1. What is the definition of a p2p system given by the authors in sec 1?

The term “peer-to-peer” refers to a class of systems and applications that employ distributed resources to perform a critical function in a decentralized manner. The resources encompass computing power, data (storage and content), network bandwidth, and presence (computers, human, and other resources). The critical function can be distributed computing, data/content sharing, communication and collaboration, or platform services. Decentralization may apply to algorithms, data, and meta-data, or to all of them.

2. In Fig 2 (pg 3), the authors compare some aspects of the client-server and the p2p computing models. List and explain these aspects.

There is no clear border between a client-server and a P2P model. Both models can be built on a spectrum of levels of characteristics (e.g., manageability, configurability), functionality (e.g., lookup versus discovery), organizations (e.g., hierarchy versus mesh), components (e.g., DNS), and protocols (e.g., IP), etc. Furthermore, one model can be built on top of the other or parts of the components can be realized in one or the other model. Finally, both models can execute on different types of platforms (Internet, intranet, etc.) and both can serve as an underlying base for traditional and new applications.

3. What is a hierarchical and what is a flat client-server model?

A client-server model can be hierarchical (mainly for improved scalability). In this type of model the servers of one level are acting as clients to higher level servers. Examples of a hierarchical model is DNS server and mounted file systems.
On the other hand, the client-server model can be flat where all clients only communicate with a single server (possibly replicated for improved reliability), or it can be hierarchical for improved scalability. Examples of a flat model include traditional middleware solutions, such as object request brokers and distributed objects.

4. What is a super peer?

A super-peer is a node in a peer-to-peer network that operates both as a server to a set of clients, and as an equal in a network of super-peers.

5. What is the difference between a compute-intensive and a componentized application? How does this relate to vertical and horizontal distribution?

The general idea behind a compute-intensive application is that idle cycles from any computer connected to the Internet kai be leverage to solve difficult problems that require extreme amounts of computation. Most often, the same task is performed on each peer using different sets of parameters. messaging is one subclass of this class of application. On the contrary componentized applications run different components on each peer.

6. What is according to the authors the main challenge of communication in p2p?

The P2P model covers a wide spectrum of communication paradigms. At one end of the spectrum are desktop machines mostly connected via stable, high-speed links over the Internet. At the other end of the spectrum, are small wireless devices such as PDAs or even sensor-based devices that are connected in an ad-hoc manner via a wireless medium. The fundamental challenge of communication in a P2P community is overcoming the problems associated with the dynamic nature of peers. Either intentionally (e.g., because a user turns off her computer) or unintentionally (e.g., due to a, possibly dial-up, network link failing) peer groups frequently change. Maintaining application-level connectivity in such an environment is one of the biggest challenges facing P2P developers.

7. What is the most common solution to reliability across p2p systems.

Reliability in P2P systems is a hard problem. The inherently distributed nature of peer networks makes it difficult to guarantee reliable behavior. The most common solution to reliability across P2P systems is to take advantage of redundancy. For example, in case of compute-intensive applications upon a detection of a failure the task can be restarted on other available machines. Alternatively, the same task can be initially assigned to multiple peers. In file sharing applications, data can be replicated across many peers. Finally, in messaging applications, lost messages can be resent or can be sent along multiple paths simultaneously.

8. What are the advantages/disadvantages of the centralized directory, the flooded requests, and the document routing models.

- Centralized directory

This model requires some management infra-structure (the directory server), which hosts information about all participants in the community. This can cause the model to show some scalability limits, because it requires bigger servers when the number of requests increase, and larger storage when the number of users increase. However, Napster' experience showed that – except for legal issues – the model was very strong and efficient.

- Flooded requests

This model, which is used by Gnutella, requires a lot of network bandwidth, and hence does not prove to be very scalable, but it is efficient in limited communities such as a company network. To circumvent this problem, some companies have been developing "super-peer" client software, that concentrates lots of the requests. This leads to much lower network bandwidth requirement, at the expense of high CPU consumption.

- Document routing

Although the document routing model is very efficient for large, global communities, it has the problem that the document IDs must be known before posting a request for a given document. Hence it is more difficult to implement a search than in the flooded requests model. Also, network partitioning can lead to an *islanding* problem, where the community splits into independent sub-communities, that don't have links to each other.

9. In the centralized directory approach, after the best peer is located, the file exchange occurs directly between it and the requesting peer. What are the advantages/disadvantages of this?

10. What can be considered as a closure mechanism in Gnutella?

New nodes must know the address of another Gnutella node or use a host list with known IP addresses of other peers. The node joins the network of peers by establishing a connection with at least one peer currently in the network. Then, it can begin discovering other peers and cache their IP addresses locally.

11. What are the factors that affect scalability, give one example for each.

Scalability is limited by factors such as the amount of centralized operations (e.g, synchronization and coordination) that needs to be performed, the amount of state that needs to be maintained (e.g., in Gnutella networks the new nodes must know the address of another Gnutella node or use a host list with known IP addresses), the inherent parallelism an application exhibits, and the programming model that is used to represent the computation.

12. Given the ad-hoc nature of connectivity in p2p, comment on what type of (message-oriented) communication (i.e., synchronous/asynchronous, transient/persistent) would be more appropriate.

Because of the ad-hoc nature of connectivity in p2p (some of the nodes will be available all the time, some will be available part of the time, and some will not be available at all) I would use a transient asynchronous communication scheme. Transient because we need the messages to be stored only as the sending and receiving application are executing, and asynchronous because we want for the sender to immediately continue working after submitting its message.

14. What is the goal of caching in p2p? What are the advantages/disadvantages of caching the reply at all nodes in the return path? Can you think of any alternatives? Is this possible in Gnutella?

The goal of caching is to minimize peer access latencies, to maximize query throughput and to balance the workload in the system. Caching the reply at all the nodes in the return path (like in Freenet) has a major advantage. It reduces the path length requried to fetch a file (and therefore the number of messages exchanged between the peers) for further queries only. The main disadvantage is the need for enormous cache space. An alternative is to cache the reply every 5 or so nodes (depends on the length of the return path). This approach doesn't need so much caching space and it still reduces latencies. In Gnutella a node broadcasts and rebroadcasts a request until it reaches a node with a response. Hence this approach cannot work in Gnutella since the request may arrive to the node with the response using several different paths.