Questions for FFS and LFS:

1. *What are the pros and cons of using large blocks.*

   + .More data in each file can be represented by direct data pointers in the i-node;

   + More sequential read, less seek/rotational delay, higher throughput;

   - waste disk space, especially when most files are small; .

2. *Why does LFS have better writer performance than FFS?*

   LFS does not update a block in its original location in disk; instead, all updates are organized and buffered in a segment, and then written to disk one segment at a time. Since LFS has turned random writes to sequential writes, its write-access performance is much better than FFS.

Questions for RPC/LFS:

1. *What does the binding process accomplish in RPC?*

   The binding process helps RPC calling server to identify and locate the RPC service provider.

2. *Why does LFS have worse read performance than FFS?*

   LFS makes sure that all file updates are sequential. As a result, even a sequential read of a file (i.e., reading the file from the first byte to the last byte following the in-file position order) may end up visiting disk blocks that are far away from each other ---- these blocks may be updated at different time and hence are in different segments on disk.

   Instead, sequential read of a file typically would read blocks inside the same cylinder group in FFS, and hence could be much faster than that in LFS.

   (Of course, when we say `worse', we are just referring to a likely trend. It is definitely possible to create a specific workload so that LFS has better read performance than FFS.)

Questions for NFS:

1. *Briefly explain how caching is used to improve the performance of NFS.*

NFS caches data blocks to improve performance. Before using a data block, it will check whether this data block is still up to date. NFS uses time-out mechanism and attribute-block checking to decide whether to use the cached data blocks.

AFS caches the whole file. After the first open, later file accesses will get local-disk access throughput. In later versions of AFS, call-back mechanism is used to determine whether a file needs to be discarded and fetched again from the server.

2. *What is a file handle in NFS?*

   A file handle includes three parts: the i-node number, the i-node generation number, and the file system ID. It is used for NFS client to reference a file on the server.

Questions for AFS and GFS:

1. *Under what type of file access patterns, will AFS offer better performance than NFS?*

   If it is a relatively big file, and the client machine ends up needing to access most blocks in this file. In this case, AFS would offer better performance than NFS.

2. *Which system offers more reliability, AFS or GFS?*

   GFS, because AFS does not offer any data replication.

Questions for MapReduce, DryadLINQ:

1. *Discuss how Map-Reduce systems handle node failures..*

   Master periodically pings every worker node to detect worker failures.

   Once a worker fails, all the finished or on-going mapper tasks, and all on-going reduce tasks running on it will be re-executed on another node. Reducer tasks that are supposed to read data from the re-executed mappers are notified. If the reducer task hasn't finished reading the data, it will instead read data from the new map-worker.

   Finished reduce tasks do not need to be re-executed.

   Finally, if the master fails, the job will be re-executed.

2. *What are the advantages of DyadLINQ over MapReduce?*

   DryadLINQ provides a richer set of APIs for developers to write their distributed computing program; it also provides more static and dynamic optimization support.