1. HDFS Federation:

The namenode keeps a reference to every file and block in the filesystem in memory, so if there are very large clusters with many files, memory becomes the limiting factor for scaling.  
HDFS federation divides the namednode to sub-namenodes and adding them to clusters, each of which manages a portion of the filesystem namespace. each namenode manages a namespace volume, which is made up of the metadata for the namespace, namespace volumes are independent of each other, which means namenodes do not communicate with one another, and furthermore the failure of one namenode does not affect the availability of the namespaces managed by other namenodes.

2. HDFS High Availability; fallover and fencing:

the replication of namenode metadata on multiple filesystems and using the secondary namenodes to protect data loss don't provide high availability of the filesystem, the namenode is still a single point of failure (SPOF).  
Hadoop solved this problem. In HDFS high availability (HA) implementation, there are a pair of namenodes in an active-standby configuration.In the event of the failure of the active namenode, the standby takes over its duties to continue servicing client requests without a significant interruption.

failover: The transition from the active namenode to the standby is managed by a new entity in  
the system called the failover controller.

fencing: a method which ensure that the previously active namenode is prevented from doing any  
damage and causing corruption.

3. Read and Write sequence in HDFS:  
Read:   
In hadoop reading data can be done by many ways:  
- Reading Data from a Hadoop URL: One of the simplest ways to read a file from a Hadoop filesystem is by using a  
java.net.URL object to open a stream to read the data from.  
- Reading Data Using the FileSystem API: Sometimes it is impossible to set a URLStreamHandlerFactory for your application. In this case, you can use the FileSystem API to open an input stream for a file.  
A file in a Hadoop filesystem is represented by a Hadoop Path object, because the fileSystem is a general filesystem API, so the first step is to retrieve an instance for the filesystem we want to use.

Write:   
The FileSystem class has a number of methods for creating a file. The simplest is the  
method that takes a Path object for the file to be created and returns an output stream  
to write to. also we can append to existing file or copy existing files.

4. Primary and Secondary Namenodes

The namenode stores the HDFS filesystem information in a file named fsimage. Updates to the file system (add/remove blocks) are not updating the fsimage file, but instead are logged into a file, so the I/O is fast append only streaming as opposed to random file writes. When restaring, the namenode reads the fsimage and then applies all the changes from the log file to bring the filesystem state up to date in memory. This process takes time.  
The secondarynamenode job is not to be a secondary to the name node, but only to periodically read the filesystem changes log and apply them into the fsimage file, thus bringing it up to date. This allows the namenode to start up faster next time.  
The secondarynamenode service is not a standby secondary namenode, despite its name. Specifically, it does not offer HA for the namenode.