1.

There would need to be a central process that keeps track of a few things. The first will be the total number of files in the system. The second will be the average size of the files. Last it will keep track of the ideal total size of files that each node should be processing. The central process will broadcast this information to each node. If a node has a larger size than it should, it sends files right to its neighbor until it reaches the ideal size plus or minus a certain percentage of that size. If a node receives a file, and it cannot hold it, it would simply continue passing it along to its neighbor. If a node ever receives a file that it has already passed off, it will hang onto it and override the size requirement. The reason for this is because it would mean that the rest of the nodes are at their max capacity as well. Every time a new file or node is added to the cluster, a signal will be sent to the central process, and it will recompute the ideal load balance again, and then broadcast that information to the nodes. This re-computation would also take place whenever files or nodes were removed from the cluster. Because the file passing is handled by the nodes and not the central process, this cluster should be able to scale very well.

2.

For this problem, I would use two rounds of mapping and reducing. On the first pass, the mapper will collect the five pieces of data, and send them to the reducer. In the first round of reducing, the reducer will make some decisions about what to pass on based upon the values it receives. For the information to be passed on, it will have to pass the following check:

```
if((scrobbled || radioPlay) && !skip) map2(userID, trackID);
```

In the second round of mapping, the mapper will need to keep track of which trackIDs have already been associated with the userID. If it comes across a trackID that has been seen before, it would send a

pair like (0, userID) to the reducer. If the trackID has not been seen, the pair (1, userID) will be sent to the reducer.

In the final round of reducing, the reducer will simply keep track of the userIDs, and total the numbers for each userID. Because a 0 is sent on repeat trackIDs, the final result will be the count of the unique tracks played for each user, ignoring repeats.

3.

Although MapReduce and DBMS both operate in parallel, each one of them is better suited for a particular set of problems. MapReduce is better suited for performing complex operations on a set of data than a DBMS. A DBMS on the other hand, is great for filtering information based on a query, and doing so in a very short amount of time. When MapReduce is used to perform in a manner similar to a DBMS, it does so much slower than the DBMS. Similarly, a DBMS can perform map reduce operations, but much less efficiently. I would choose to use MapReduce in a situation where I needed to parse data from a large number of files that also contained a great deal of irrelevant information. This extra information would have to be separated, likely in a complex manner, to get down to what was important. A situation where I would use a DBMS is one where all the information I needed was contained in files that were well formatted, which would make the retrieval process very quick. Even though these two technologies are not great at emulating the other, in conjunction, they can be very powerful. A large set of data could be reduced by MapReduce, and then sent to a DBMS for later retrieval.