Q1. Assume that you have complete control to reorganize and distribute the dataset. What is the fewest number of MapReduce jobs that you would need to implement the complete set of tasks in HW3-PC? Note that your program should emit outputs for each task into a separate file. [300-400 words]

Although I was not able to fully complete the last few questions, I believe the fewest number of MapReduce jobs would be two, assuming that I have complete control to reorganize and distribute the dataset. From my experience in the third assignment, one of the biggest difficulties in working with the datasets came from the Analysis files lacking the “title” field that the Metadata files had. For the questions that required the song names and artist names they belonged to with only the song_id as comparisons between the two, there were many bad records where the title did not exist in the reduced dataset and was null. I am still not sure if this occurred because due to the analysis files lacking the song_id that the metadata files had or vice versa. I think an easy way to fix this and reduce the amount of time needed for the reducer to compare both files for the song_id would be to simply add a “title” field to the analysis files.

Additionally, since the analysis file is already a very large file relative to the metadata files due to the large amount of fields, it would be more efficient to run jobs if this dataset were split between into one with the general song information and one with the specific song analysis, which would hold fields such as tempo and segements_pitches. I did not have to use the specific song analysis pieces that analyzed the parts of a song. The specific song analysis datasets would also have song_id and title as fields to help with avoiding bad records.

As mentioned before, I believe it would only take two jobs because of the way I implemented this assignment. Although I did not complete every question, the planned methodology I wrote out explained what would be needed to answer the question, and I would at most use three mappers for the metadata, general song analysis, and specific song analysis files. This made it simple to create lookup tables for the reducer to use for each question. Question 10 would require another job due to requiring another dataset to be used (check README.txt for more information).

Q2. You are designing a new streaming service that can scale to millions of users and stream songs that are personalized to a user. How would you extend assignments HW1-PC, HW2-PC, and HW3-PC to accomplish this goal? [400-500 words]

In order to design a new streaming service that can scale to millions of users and stream songs personalized to a user, I would begin extending the assignments by using HW2-PC as the base for the entire program.

The second assignment utilized thread pools and micro-batching to manage, load, and balance active network connections in a scalable server design, which is essential when considering that the number of accesses to the streaming service will constantly change with users logging in and out and with people creating new accounts. However, as of now, the second assignment restricted the number of threads in the thread pool to 10 and supported up to 100 active clients. This is not enough to support a streaming service with a growing user base, so the overlay will need to be upgraded.

To upgrade the overlay, the server itself will need to become a messaging node, and there will need to be multiple messaging nodes created as needed for the service. There will

also need to be a messaging node manager that is able to call on certain messaging nodes from a messaging node pool. There will be one overall server working with the messaging node manager to connect to and receive data to send to select messaging nodes. Each of the messaging nodes can then process information for multiple clients and send it back to them.

This is where HW1-PC needs to be implemented. Since assignment two's server became a messaging node itself and there are now multiple messaging nodes to be able to support more clients, this means that the overlay of messaging nodes needs to be managed and optimized. The messaging node manager mentioned above will become a registry for the messaging nodes in the messaging nodes pool. The registry and messaging nodes will communicate with each other to register and deregister. Instead of assigning link weights to messaging nodes like in assignment one, the overlay will need to keep track of the actual link weight between nodes. Once the network overlay is set up, dijkstra's algorithm can be implemented to get the shortest path between nodes.

As for extending HW3-PC, since the songs need to be personalized to a specific user, there will need to be a separate dataset created that contains information on specific users and their streaming history. This dataset will limit information by refreshing each month and keep the dataset from growing too large. From the history, MapReduce can be used to pull data from an available music dataset. This will allow the messaging node to complete a specific user's request after obtaining the data.

Q3. What if a family has a shared account on your streaming service? How would your streaming service recommend songs so that each family member is satisfied? Please design your own measure for satisfaction. [300-400 words]

As mentioned in the second question above, information on the user accounts will be kept in a separate dataset and be updated monthly. In the dataset, in order to differentiate between different family members under the same user account, each entry can have an account_id field as well as a unique user_id for each family member under the account. This will allow the data on each family to be grouped and will allow for the information to be mapped per family and per family member.

In the case that a new family member makes a new profile under the user account, there will be no history of what was listened to. With no previous history, the streaming service can go through the million songs dataset or other music dataset and use MapReduce to pick the artists that have the highest hotness rating, artists with the greatest average song hotness, songs with the highest hotness rating, and artists with the highest familiarity rating (similar to the methodology in answering HW3-PC Question 8 for the most generic and most unique artists). This will at the very least be able to recommend the most popular songs for users with no history.

As for the users in a shared account who do stick around and build up their user history, the million song dataset can be used to grab "similar_aritists" from the Metadata file for each song that the user has listened to. The recommended songs can be changed on a weekly basis so that the list is not exhausted too quickly, and the amount of recommended songs can be limited to perhaps 30 or 40 songs per week. The satisfaction from the recommended songs

feature can be measured by tracking the number of plays the user made and tracking at what time the user changed songs or if they finished the song.

Q4. You are starting a music production company and you are working with local artists in town. How would you perform micro-adjustments to a song so that it is a commercial success in different countries? How would you avoid concept drifts - what is in vogue today may not be in the near future? [300-400 words]

In order to perform micro-adjustments to a song so that it is a commercial success in different countries, it will be necessary to first analyze highly rated songs in specific countries. The metadata file in the million songs database is contains locations in the fields artist_latitude, artist_longitude, and artist_location. This will allow for every song in the dataset to be categorized. From there, similar to question eight from assignment three for finding the most generic artists and most unique artists, I interpreted that question to mean generic implies low ratings overall whereas unique implies higher ratings. This interpretation required the artists to be sorted with the highest hotness rating, highest average song hotness, and highest familiarity rating. Along with sorting the artists in those three different ways, they can also be further divided by location with the artist location fields in the metadata.

These results will give insight into what is currently in vogue today by examining the artists that are the most popular at the time. If the results were then used to analyze whether the local artists' music will become a commercial success in a different country, then it would be useful to pick the top ten artists from each of three of the results (highest hotness, highest average song hotness, highest familiarity) from the target country. I could grab the songs from these sets of artists and use MapReduce again to this time look through the analysis file and grab the averages of the various fields such as key, tempo, danceability, energy, etc. for each song in all three sets of artists. This should give the local artists an idea of what composes a highly rated song in select different countries.

If we want to further make sure to avoid concept drifts, the results could be further grouped by the year field in the metadata, which would allow the company to keep track of the highly rated songs and artists throughout the years and notice emerging and disappearing trends.

Q5. The kind of music a person likes may change/evolve over time. Describe a potential scheme to design "interest trajectories" that allows you to recommend songs that a person may like in the future. [300-400 words]

If it is assumed that only the Million Songs dataset can be used, one potential scheme to design "interest trajectories" that allows you to recommend songs that a person may like in the future would be to recommend songs based on different fields found in the datasets. The user can be recommended songs in different groupings based on the fields. For example, one simple way would be to look at the metadata files for the fields similar_artists. This would allow an easy way to recommend artists that are similar to the user's history (as mentioned in previous questions).

Another way to recommend songs from the metadata file would be to use the artist_terms, artist_terms_freq, and artist_terms_weight fields. The field "artist_terms" are a list

of terms describing the artist, so one way to recommend artists to the user would be to check each song and see which ones share the same descriptive terms with others. There could be a specific grouping of songs that recommend users on artists that have at least n similar terms to the songs the user frequents or has favorited.

Another approach to recommending songs is to use the analysis file to recommend songs. This recommendation would be based on fields such as danceability and energy and would be compared to the data from the user's history. Since this does not directly give comparisons to other artists such as from the similar_artist field, this recommendation would be considered as more of an explorative option for the users to find new songs. The same could be done if fields from the segments, sections, beats, bars, and tatums fields from the analysis files were compared to the songs listened to on the user's history, but it would be much more difficult to get a recommendation and would therefore be more of an explorative option for users.

Essentially, in order to make give a user proper recommendations and a chance to expand their interests, it would be beneficial to give them various groupings of songs similar to their play history in different ways.