1. How do you represent a sound as a wave?

Reference Video Link: https://youtu.be/mEhNOz_CxcY

You can see that the sound wave has different frequencies at different times. We plot these points on a graph of time vs. frequency. This continuous function is then called a wave. We can similarly plot the graphs of time vs. amplitude, etc...

2. Why does this video only talk about the sound search algorithm? Where is an explanation of the architecture?

This video is about analysing an algorithm in depth. The considerations of server load, fault tolerance, etc… are discussed in detail in the other videos.

3. Is this algorithm capable of handling millions of requests searching amongst millions of songs?

The database has only insert and select operations. We do not require any transactional guarantees during querying.

The requests will be processed by the mobile clients and sent as a set of interesting points to the server.

This means there is little left to do on the server except matching a point map to the database. Even for millions of requests a day, this system should be scalable.

4. What if two chunks have very similar signatures?

As long as the hash function is chosen appropriately, this is highly unlikely.

5. What if the song has no interesting points in it? Similarly, what happens if the clipping has no interesting points?

The first scenario is highly unlikely, to the point of being assumed impossible. If there are no interesting points in the clip, it usually means one or more of the following:

a) The clip is too short

b) The audio is too faint

c) The noise is too much.

6. Is it possible that two servers give different results for the same audio clip?

If the process is deterministic, then no. If the algorithm changes, a new deployment will be needed. This might lead to multiple versions of code in the production environment.

When this happens, the same song clip can give different song match results.

7. What happens if the API has to return the most likely matches?

We can modify the response to return the songs having the most hash matches.