Audio Search Engine - High Level Design

In an audio search engine users send a sound clip to us and we need to identify the sound clip is part of which song.

Let's start with a **simple solution**

Comparing waveforms

A song/sound clip is just a wave. For each song we can save its waveform in our database. To make our search results more accurate we can store multiple waves of **frequency v/s time, amplitude v/s time** etc. Whenever we get a new sound clip we compare the sound clip's waves with each and every song's waves. The song which matches the maximum with the clip's waves is our answer.

Now this approach works however there are two major problems

- **Storage requirements are high** For each song we are storing its multiple waves. So we need a lot of storage.
- **Slow response time** We are comparing each and every wave form of the sound clip with each and every wave form of every song. This process is very slow and inefficient.

Let's come up with a better solution

Point of Interests

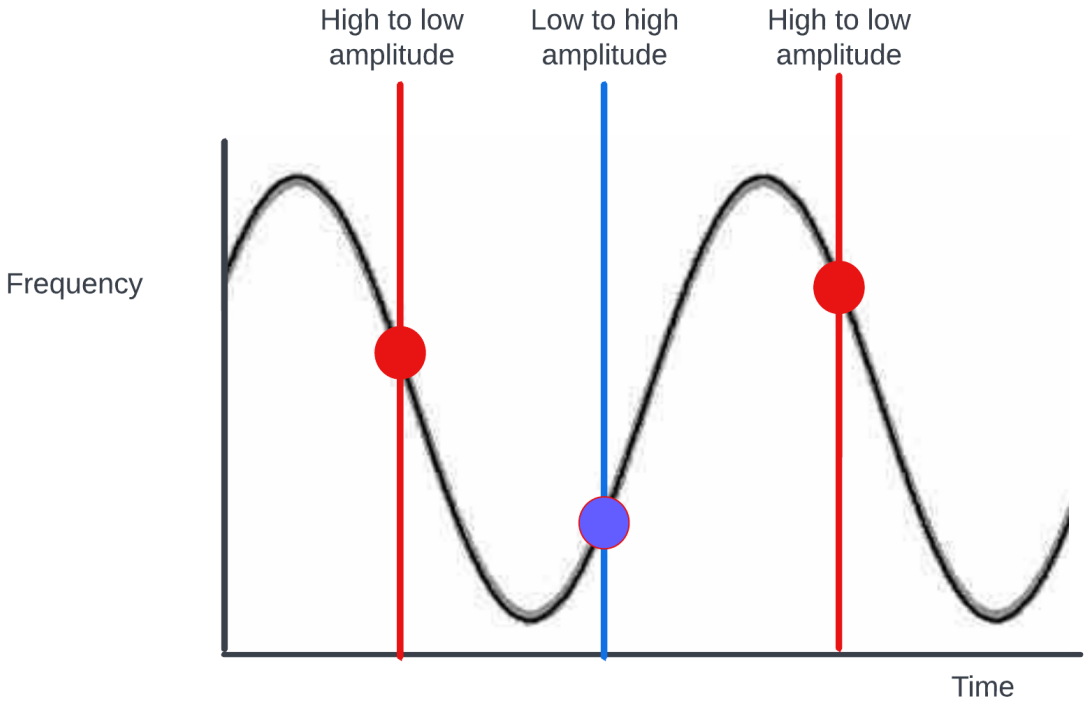
We want to identify song without storing the entire waveform. So to do that we can store multiple **point of interests** in a song.

Point of Interest : is a point where there is a sudden change in the amplitude. (*Maybe the amplitude suddenly increases or there is pin drop silence*).

It is almost impossible for two different songs to have the same number of point of interests and at the same time. So now we can **uniquely identify a song by storing only its point of interests.**

What values do we need to store for each point of interest?

We will store the frequency and time for each POI. **Note :** We do not need to store the amplitude because we are only concerned about the **significant change in amplitude** not the exact value. And the fact that it is POI tells that there is a change.



Okay, now we have stored the POIs for each song. But **how do we compare the song and the sound clip?**

First it is important to point out that we cannot compare the POIs by the time component because it is not fixed. We are not sure what is the offset of the clip. Let's understand this with an **example**

We have song with POIs at **20s** and **50s**. User starts recording the sound clip at the **10s** mark, so for the sound clip the POIs will be at **10s** and **40s**. If we compare the frequency at **20s** match then it won't be a match.

To solve this issue

We will store **pairs of POIs and the time duration between the them**. So the uniquely identifying characteristics for a song are **(F_1, F_2 , Time interval between F_1 and F_2)**

Now if the find the same frequency with the same time delta in the clip that means there isa match. The song which has maximum number of matches is our result.

But can we optimize this any further?

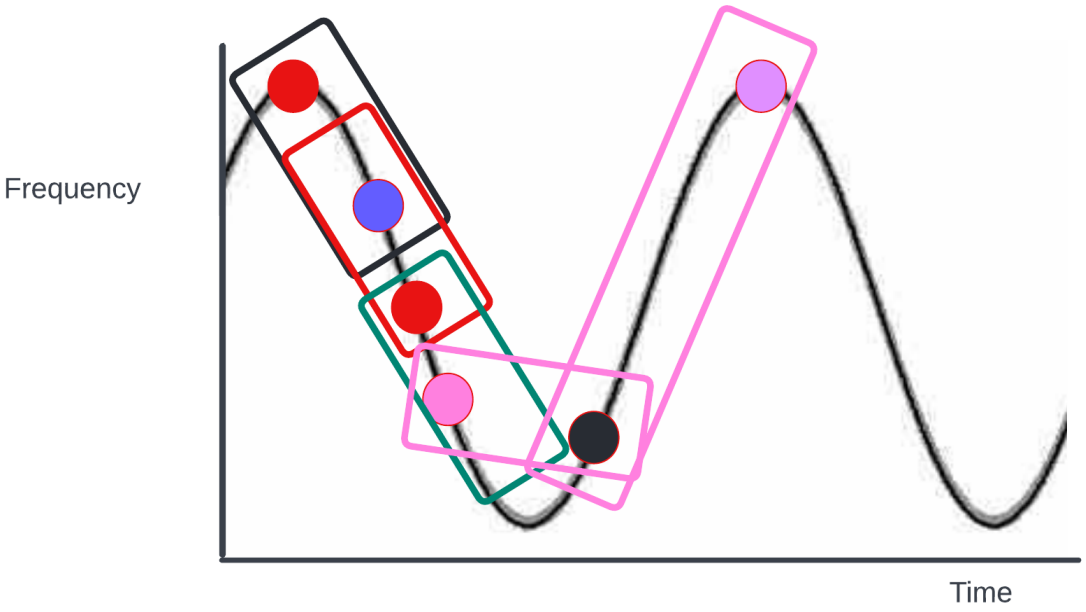
If there are **n** POIs there there will be **n*(n-1)/2** pairs. So each comparison will have worst case time complexity **O(n^2)**.

We can optimize this process by dividing the song into chunks and then compare the chunks of song and sound clip. If the chunks match then we have our song.

Problems arises when the two points are in different chunks.

We repeat the same process for the sound clip as well. Then we **find the chunks having maximum number of matches with our clip**. The song which contains this chunk is our answer. (*Notice that instead*

of comparing two pairs of POIs, we are comparing chunks).



N = 5 , p = 2

Time complexity of this solution

Let's say we have **n** points and each chunk has **p** points. So there will be a total of **(n-p)** chunks.

So the time complexity will be: **(n-p) * (p*(p-1))/2** = **O(p^2 * (Number of blocks))**

We can optimize this process even more. We can **take the points in the chunks and convert it into a hash**. Each hash is like a key. So when searching for a chunk we just search for the key. The song having the most number of matches with keys for its chunks is the song we are looking for. We can search for a key in **O(1)**, so the search will be very fast.

Capacity Estimation

Storage estimation

Assumptions

- Total number of songs = 1,000,000
- Average length of each song = 3 min
- Average length of each cip = 10s
- Number of interesting points per chunk = 10
- Average number of requests per second = 1000
- Search time for a pair of points ~ 5ms

Size of each point (F1,F2,TimeInterval) = 4 bytes + 4bytes + 4bytes = 12bytes

Number of pairs per chunk = (10*(10 -1))/2 ~ 50 pairs

Average number of chunks per song = 3*60 / 10 ~ **20**

Total storage = Number of songs * Number of chunks * Number of pairs * Size of each pair ~ 12 GB

Taking into account replication for fault tolerance and performance we multiply it with a factor of 3: **12*3 ~ 36 GB**

Process estimation

Number of pairs in sound clip = (10*(10-1))/2 ~ **50 pairs**.

Therefore, total processing time per second = Number of requests * Number of pairs in sound clip * search time = 1000 * 50 * 5ms ~ **250 seconds/second**

We want to keep the load of the processors at about 60% , so number of processors required = 250 / 0.6 ~ **400 processors**

That's it for now!

You can check out more designs on our video course at InterviewReady.