Hue: a Partially Decentralized Solution to Music Streaming

Marcus Lee '26

Abstract

ABSTRACT ABSTRACT

1 Introduction

With the advent of music streaming platforms, artists have lost control of their work. Spotify, Apple Music, and essentially all other market players use a monthly subscription model, in which users pay a monthly fee for unlimited access to all music the platform provides. This economic model, though arguably beneficial for users, also allows these platforms to determine themselves how much each stream is worth and how much artists should get paid.

Having a work's value be a function of its stream count isn't inherently bad, but here's an issue: What happens when music platforms lie? Since all data pipelines and stores are owned by the platform, there is incentive to lie about streaming numbers in order to pay artists even less. Perhaps more feasible, what happens when streaming platforms decide to suddenly change how they pay out artists? In fact, Spotify changed their entire royalty structure in April 2024, distributing more for bigger artists while leaving smaller ones with less than they already had. Thus, artists have to relinquish control and power over to the music streamers in order to participate in the industry.

Given this, there is a clear need for platforms that enable and give control back to artists. Through a transparent and trustless data store process, as well as artist-decided pricing, Hue's decentralized approach to music streaming represents an attempt to disrupt the industry and restore control to artists.

2 Related Work

Decentralized music streaming is not an original thought. In fact, there are already well-funded startups with implementation. Through basic research, I found there were two projects with any form of notoriety: Audius and Opus. It is common within the blockchain community to produce "whitepapers" which explain their decentralized design. Through reading Audius and Opus's whitepapers, one issue was very clear to me. These platforms decentralize every single aspect of the system, whether it is the data store, content ledger, or even discovery nodes. Although they successfully create an economic system independent of any custodian or centralized party, this full decentralization leads to a poorer user experience in my opinion. In order for users and artists to fully understand how to navigate and harness their platform, they need to be onboarded to crypto and blockchain technology. To combat this issue, Hue is only partially decentralized in order to deliver a comfortable and familiar user experience for listeners.

Hue is built on top of the Sui blockchain, whose infrastructure is optimized for speed, scalability, and developer experience. Realistically, one could use any programmable blockchain to decentralize a music streaming platform. I personally developed Hue using Sui and its proprietary smart contract language, Sui Move, simply because I have prior experience. Another significant factor for my choice in using Sui is that they recently announced their decentralized blob storage protocol: Walrus. This decentralized storage system not only optimizes for low-cost, and high-availability, but also comes with a HTTP API for both upload and lookup. This easy-to-use API for decentralized storage as well as prior experience are what drove Hue to be developed on top of Sui technology.

3 Design

Hue's guiding design philosophy is simple: combine the familiar music-streaming user experience with transparent data. No one listener on Hue should feel the effects of blockchain, unless they want to. All streaming and audio track data must be trustless and auditable by any third party. Essentially, it should feel like every other music streamer,

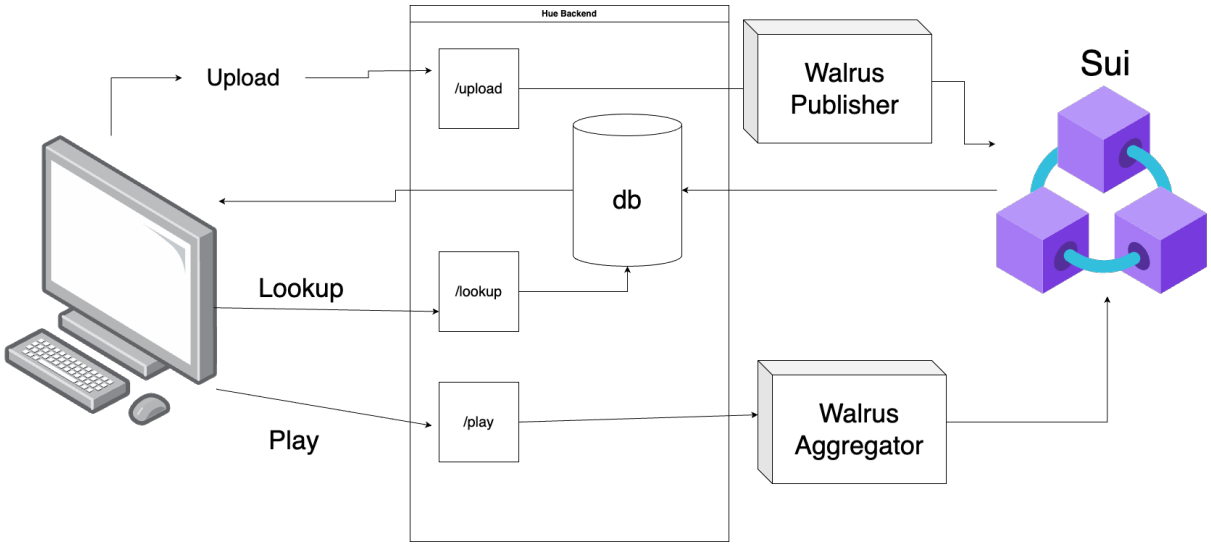


Figure 1: Hue Design Overview

except with accessibility to transparency that empowers artists and users alike.

From a bird's eye view, Hue consists of three parts: a centralized, user-friendly frontend, a backend server that encapsulates connections to both decentralized systems and a centralized database, and finally smart contracts deployed onto the Sui blockchain. One can imagine parallels to the Napster architecture, where a centralized database is used to optimize lookup within a decentralized storage system. This architecture aligns with our design principle of simple interface and user experience above all else. In 2024, who is willing to wait potentially seconds just to see what songs they can listen to? The answer is no one. The following sections will be an overview of the upload, lookup, and play functions to better understand the overarching system design.

3.1 Upload

To start the upload of an audio track, an artist needs to provide a title, cover image, price per stream, and of course the audio file itself. Using these parameters, the frontend then sends a POST request to the backend server. Immediately after receiving this request, the backend sends a POST request to a Walrus publisher, which is simply an HTTP endpoint used to upload to the decentralized storage protocol. Note that it is possible to host your own Walrus publisher, which would be useful for the future of this work. Assuming the publisher is successful in uploading the binary data of the audio data, it then returns a blobId which can be provided to a Walrus aggregator endpoint in order to fetch the data later.

After getting this BlobId, the backend connects to Hue's smart contract on Sui. Through using Hue's admin account on Sui, the backend essentially tells the smart contract to create a digital "Track" object representing the track being uploaded. This object is what will encapsulate the stream count, royalty payment, stream price, and all other data related to this audio track. Here is the Sui Move code for 'create_track' to better understand how this might work.

```
    public fun create_track(
        title: String,
        blob_id: ID,
        cover_url: String,
        publish_date: String,
        cost_per_stream: u64,
        ctx: &mut TxContext
    ): ID {
        let sender = tx_context::sender(
            ctx);

        let track = Track {
            id: object::new(ctx),
            title,
            artist: sender,
            blob_id,
            cover_url,
            publish_date,
            stream_count: 0,
            cost_per_stream,
            royalty_balance: balance::
                zero<SUI>(),
        };

        let track_id = object::id(&track
            );

        event::emit(TrackCreated {
            track_id,
            title,
```

```
            artist: sender
        });

        transfer::share_object(track);
        track_id
    }
```

Though this smart contract is useful in how it can create non-fungible digital assets to represent our Track's data, the most powerful and unique aspect is how this object can store currency. This currency can only be withdrawn by the artist's Sui address which would be associated with the artist's Hue account.

After the audio file is stored and linked to a Sui object, its data is then stored into a PostgreSQL database. As we are only storing its attached data and not the binary itself, this data input is very lightweight. After, a 200 OK is sent back to the frontend to let the artist know that Hue has successfully uploaded their audio track to the system.

If anytime throughout this process there is an error with any given connections or services, the backend returns an error to the frontend and discontinues the upload. This is to ensure Hue does not list any artists' works that do not follow the transparent data paradigm created within the system.

3.2 Lookup

Since we have a centralized database with information on all audio tracks uploaded to Hue, lookup is very simple. In my MVP implementation of Hue, the frontend sends a simple get request to the backend. Then, the backend returns every single instance inside the PostgreSQL database's track table as an array to the frontend. Finally, the frontend takes this array and presents it in a nice list to the user. The choice of a centralized database was deliberate in how it mimics the speed and efficiency of centralized platforms such as Spotify. Once again, this aligns greatly with our design principle of industry-standard UI/UX.

3.3 Play

To initiate playback of an audio track, the frontend sends a GET request to the backend server. This request contains the on-chain Track ID as well as the BlobId of the audio track. Upon receiving this request, the backend takes the BlobId and queries a Walrus aggregator endpoint. This aggregator is again a simple HTTP endpoint that returns binary data previously uploaded to Walrus.

While retrieving the audio data, the backend also connects to Hue's smart contract on Sui to update the Track object. This update includes incrementing the stream count as well as processing the royalty payment based on the track's price per stream. Since this Track object exists on the Sui blockchain, all of these operations are transparent and auditable by any third party. More importantly, as these blockchain operations are atomic with the actual streaming process, we can guarantee that the Track object's data accurately represents real user engagement with the audio track.

After both operations are complete - retrieving the binary data and updating the blockchain - the backend streams the audio data back to the frontend for playback. This architecture ensures that our transparent data paradigm is upheld, as every stream counted on-chain represents an actual delivery of content to a user. Additionally, the transaction hash of every stream is sent to the frontend so users can always independently verify that their stream contributed to the artist's royalty pool.

4 Evaluation

As Hue is an application of decentralized systems, evaluation metrics will focus less on lower-level throughput but more on the user experience. Thus, each function (upload, lookup, play) was executed 100 times in order to see the time duration users have to experience in order to interact with each. See figure 2 for both execution time averages as well as distributions.

Upload (seconds):

Mean: 22.69

Std: 2.71

Lookup (milliseconds):

Mean: 45.22

Std: 9.49

Play (seconds):

Mean: 7.06

Std: 1.04

Upload time is somewhat negligible, as artists are in no rush to upload their music. The idea here is that an artist won't mind the difference between 2 seconds and 20 seconds for their uploads. However, the 7 second average time it requires to execute the play function is certainly the core issue here. Music streaming users are accustomed to nearly instantaneous retrieval of audio files, so these metrics

indicate that the current implementation of Hue does not meet its original design intent of feeling just like market-leading music platforms. This does not mean this system is bottlenecked, however. Not only does Walrus have its own caching mechanism (that was not explored in this project), but in general every music streaming platform requires some type of caching in order to deliver its instantaneous speed. This idea, and many more, will be explored within the 'Future Work' section.

5 Future Work

This implementation of Hue was not meant to be the final product. Rather, it was a proof of concept that decentralized systems can be merged into existing centralized models, rather than having to choose between one or the other, to deliver transparency and control back to its users. That being said, there are many directions of improvement that could be explored for the growth of this project.

5.1 Caching

As mentioned before, the biggest issue with the current implementation of Hue is the efficiency of the actual streaming itself. By having to hit the Walrus aggregator endpoint every single instance of a stream, the function itself is bottlenecked by the Walrus protocol's ability to find every sliced up shard of the audio file and retrieve it back to the client.

Thus, caching is a clear solution. Walrus has its own caching solution, but it is also designed to work well with traditional caching and CDNs. This way, the Walrus aggregator won't be needed every single stream and the backend can circuit straight to its Sui smart contract connection, which doesn't take nearly as long.

5.2 Distributed Backend

Although our centralized database provides efficient lookup times, it is also a single point of failure. To prevent this, there are numerous ways to distribute (not decentralize) out backend services. This could look like load-balancing between virtual machines in different regions, replicating and/or partitioning the database, etc. Either way, we know this is possible because every modern-day application distributes their backend service for high-availability and robustness.

5.3 Frontend / User Interface

This current implementation of Hue has a very basic frontend interface. In order to deliver the familiar user experience to listeners, there needs to be much more work done to deliver a seamless user flow. A good place to start would be an actual search function which allows users to lookup keywords.

5.4 Sharding Royalty Ownership

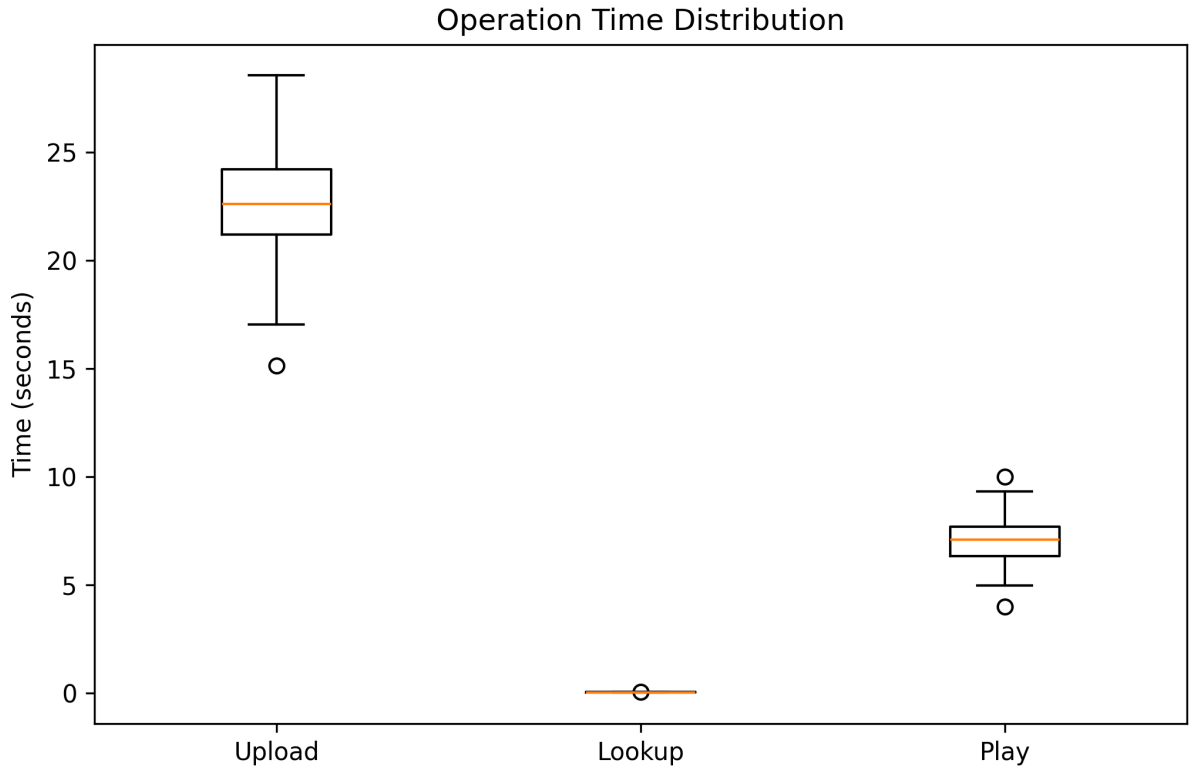
Perhaps the most interesting direction forward, according to the author, is sharding royalty ownership of audio tracks. In the current smart contract implementation, it is only possible for the artists' Sui address to withdraw the royalty balance stored inside the 'Track' object. This royalty transparency is not only disrupting to the music industry and its reliance on music distributors, but can be taken a step further.

Say an artist wants to sell 5% of a track's royalties to the highest bidder. Given current structures in the music industry, there would be lots of contract signing and 3rd parties involved to ensure that the new royalty owner hopefully actually gets 5% of all royalties of this song. However, given Hue's use of blockchain, these 3rd parties and structures can be circumvented.

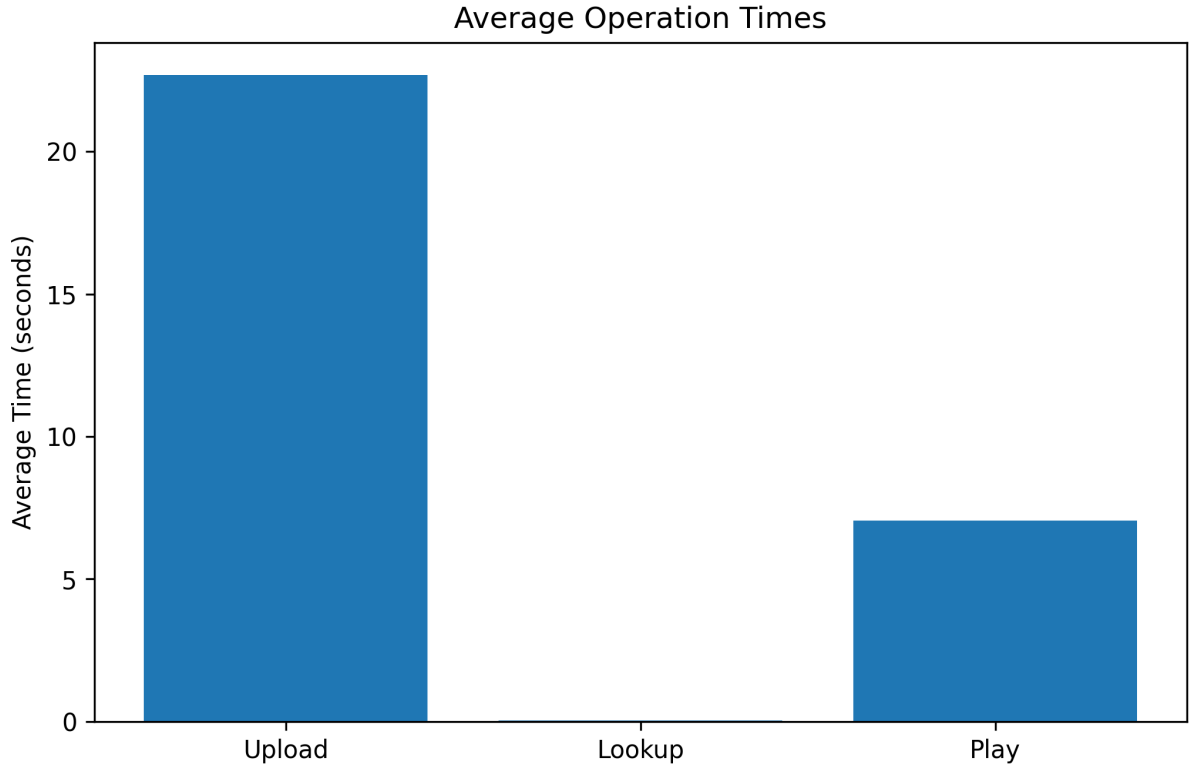
Instead, artists could be given a 'Royalty' object which represents their share of the royalty pool (initially 100%). Then, a 'shard_royalty' function could be created, where only the original artist can create another on-chain object which takes a percentage of the royalty share from their object in order to create a new 'Royalty' object. Then, this new object can be given to another account on Sui, allowing them to receive whatever percentage of the royalties that is defined within the object. This transparent and non-fungible process of selling royalties can empower artists, allowing them to be even more flexible and creative with how they monetize their work.

Acknowledgements

Thank you Jeannie for allowing me to use blockchain. <3



(a) Distribution of operation times across 100 samples



(b) Average operation times

Figure 2: Performance metrics for Hue operations