확장가능한 온체인 SVG 이미지 스토리지

박수한°, 김경은, 이돈영, 김지혜,
윤인균, 김준렬, 김현석, 안석현
CJ올리브네트웍스 NFT LAB
soohan.park@cj.net, kyungeun.kim@cj.net, donyoung.lee@cj.net, jihye.kim33@cj.net,
ik.yoon@cj.net, junryeol.kim@cj.net, hyunseok.kim7@cj.net, elliott.ahn@cj.net

EXPANDABLE ONCHAIN SVG IMAGE STORAGE

SOOHAN PARK°, KYUNGEUN KIM, DONYOUNG LEE, JIHYE KIM,
INKYUN YOON, JUNRYEOL KIM, HYUNSEOK KIM, SEOKHYUN AHN
NFT LAB, CJ OLIVENETWORKS

요 약

대다수의 NFT 프로젝트들이 메타데이터를 중앙 집중형 서버나 IPFS에 저장하고 있어 메타데이터의 손실이 발생할 수 있다. 본 연구는 블록체인 네트워크상에 이미지를 영구적으로 저장하고 변조를 방지할 수 있는 확장 가능한 SVG 이미지 스토리지 모델에 대한 연구이다. 이 스토리지 모델은 SVG 이미지를 구성하는 XML 태그들을 여러 개의 스마트 컨트랙트에 분산 저장하고, 이를 조합하는 방식으로 큰 크기의 SVG 이미지를 저장할 수 있도록 설계하였다. 또한, Property Layer, Assemble Layer, Storage Layer와 같이 총 3개의 계층으로 이루어져 있으며, 계층별 스마트 컨트랙트들이 상호작용하여 블록체인 네트워크상에서 관리의 용이성, 스토리지의 유연성, 그리고 확장성을 확보한 스토리지 모델을 구현하였다.

1. 서 론

블록체인이란 거래 내역들을 블록(Block)이라는 단위로 묶고, 그 블록들을 연결하여 데이터의 위변조를 방지한 P2P 기반 분산 원장 기술이다. [1] 그리고 블록체인 기술을 활용해 디지털 자산을 토큰화하여 소유주를 증명하는 하나의 수단으로 NFT(Non-fungible Token), 즉 '대체 불가능한 토큰'이 있다. [2]

현재 대다수의 NFT 프로젝트들이 블록체인 네트워크에 NFT의 메타데이터를 저장하지 않고, 중앙 집중형 서버나 IPFS(Inter-Planetary File System)에 저장하고 있다. [3]

중앙 집중형 서버의 경우 NFT의 메타데이터를 관리하는 가장 저렴하고 손쉬운 방법이지만, 서버의 소유자 혹은 천재지변 등에 의해 메타데이터가 손상되거나 손실될 가능성이 있다. 또한, IPFS에 저장하여 관리하는 경우, Content-addressing으로 메타데이터의 변조를 방지할 수 있지만, 메타데이터를 저장(Pinned)하고 있는 노드가 없다면 손실될 가능성이 존재한다. [4]

문제점을 해결하는 가장 이상적인 방법 중 하나는 NFT의 메타데이터를 블록체인 네트워크에 저장하는 것이다. 하지만, 이더리움(Ethereum) 네트워크의 경우, 한 번에 배포할 수 있는 최대 크기가 약 24KB로, 일반적으로 수백 KB에서 수 MB의 크기를 갖는 이미지 파일들을 저장하기엔 어렵다는 문제점이 있다. [5]

본 연구에서는 이러한 문제점을 해결하기 위해, 블록체인 네트워크상에 큰 크기의 이미지를 저장하고 나아가 확장 가능성을 갖춘 스토리지 모델을 제시하고자 한다.

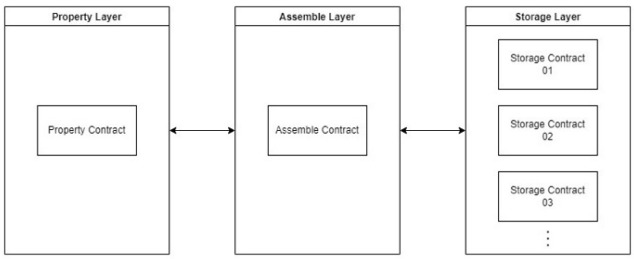
2. 확장가능한 온체인 SVG 이미지 스토리지

2.1. 접근 방향

여러 이미지 파일 확장 중 본 연구에서는 SVG 이미지를 기반으로 연구를 진행하였다. SVG 이미지 확장을 선택한 이유로, 첫째, 벡터 기반 이미지 파일 형식으로 품질 대비 적은 용량을 갖는다. 둘째로 SVG 이미지는 XML 형식의 태그들로 구성되어 있어 편집이 쉽다는 장점이 있고, 이를 바탕으로 이미지를 분할 저장하여 용량 제한이 있는 온체인 상에 저장하기 용이하기 때문이다. [6] [7]

온체인 상에 데이터를 저장하는 데 있어 제약이 있으므로, 여러 개의 스마트 컨트랙트를 배포하여 SVG 이미지를 구성하는 XML 태그들을 분할하여 저장해야 한다. 그리고 이러한 스마트 컨트랙트들을 관리하기 위해 본 연구에서는 그림 1과 같이 스마트 컨트랙트의 역할에 따라 3개의 계층으로 모델을 구상하였다.

2.2. 구조



[그림 1] 스토리지 계층 구조도

그림 1에서와 같이, 본 연구의 스토리지 모델은 Property Layer, Assemble Layer, Storage Layer로 총 3개의 계층으로 이루어져 있으며, 각 계층의 스마트 컨트랙트끼리 상호작용하며 블록체인 네트워크상에서 큰 크기의 SVG 이미지를 저장할 수 있도록 설계하였다.

- **Property Layer:** SVG 이미지를 구성하는 XML 태그마다 부여된 아이디 값을 저장하고 있으며, 사용자와 직접 상호작용을 하는 스마트 컨트랙트 계층. (이하 해당 계층에 속하는 스마트 컨트랙트를 “Property Contract”라 칭한다.)
- **Assemble Layer:** Storage Layer에 분산 저장된 XML 태그들을 조합하여 하나의 SVG 이미지를 만드는 스마트 컨트랙트 계층. (이하 해당 계층에 속하는 스마트 컨트랙트를 “Assemble Contract”라 칭한다.)
- **Storage Layer:** XML 태그들을 분산 저장하는 스마트 컨트랙트 계층. (이하 해당 계층에 속하는 스마트 컨트랙트를 “Storage Contract”라 칭한다.)

2.3. 특징

2.3.1. 대용량 파일 저장 가능

이더리움 네트워크의 경우 배포할 수 있는 최대 크기는 약 24KB로 매우 제한적이다. 본 연구의 스토리지 모델은 큰 크기의 SVG 이미지를 구성하는 XML 태그들을 여러 개의 스마트 컨트랙트로 분산 저장하고, 추후 SVG 이미지가 필요할 때 저장된 태그들을 조합하여 큰 크기의 SVG 이미지를 얻을 수 있도록 하였다.

2.3.2. 비용 효율성

3개의 계층으로 이루어져 여러 스마트 컨트랙트를 배포해야 하는 이 스토리지 모델은 작은 규모와 단기적으로 활용 시 상대적으로 높은 비용이 발생한다. 그러나, 규모가 크고 지속적인 SVG 이미지의 추가 및 수정이 이루어진다면, 변경되는 계층만 새로 배포하면 되므로 비용 절감 효과를 기대해볼 수 있다.

2.3.3. 확장성 및 유연성

기능별로 독립적인 3개의 계층 구조를 통해 확장성 및 유연성을 확보하고자 하였다. 예를 들어, 기존의 SVG 이미지에 새로운 구성 요소를 추가하고자 한다면, Storage Layer 인터페이스에 맞춰 새로운 구성 요소의 XML 태그들을 저장하고 있는 스마트 컨트랙트를 배포하고 Assemble Contract가 이를 추가로 참조하도록 설정해주면 된다.

2.4. 구현

2.4.1. 구현 환경 설정

본 연구에서 구상한 스토리지 모델을 활용해, 블록체인 네트워크 환경에서 이를 구현해 보았으며, 구현 시 환경 설정은 표 1과 같다.

[표 1] 환경 설정

Smart Contract	
Network	Ethereum Rinkeby Network
EVM Version	London
Solidity Version	^0.8.7
Client Script	
Language	Javascript
Platform	Nodejs@16.14.2
Libraries	ethers@5.6.4

구현을 위한 샘플 SVG 이미지로 그림 2를 선정하였으며, 해당 SVG 이미지 파일의 크기는 301KB이다.

[그림 2] 구현을 위한 샘플 SVG 이미지

2.4.2. 구현

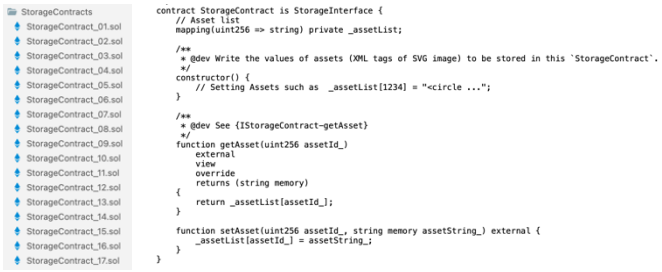
먼저, 본 연구에서 구상한 3개의 계층 구조에 따라, 그림 2의 샘플 이미지를 XML 태그 단위로 분할하였고, Property Contract엔 어떠한 Storage Contract들을 조합해야 하는지에 대한 값들을 저장하고, Assemble Contract에는 Storage Contract들의 주소값들을 저장한다. 이후, 분할된 XML 태그들을 여러 Storage Contract에 저장하였고, 본 연구에선 총 17개의 Storage Contract를 배포하였다.

```
    setAssembleContract(assembleContractAddr_);
}

/**
 * @dev See {AssembleInterface-getImage}
 */
function getImage(uint256 imageId_)
    public
    view
    virtual
    returns (string memory)
{
    return assembleContract.getImage(_attrs[imageId_]);
}

/**
 * @param newAssembleContractAddr_ Address value of `AssembleContract` to be changed.
 * @dev If later changes or extensions are unnecessary, write directly to `constructor` wi
 */
function setAssembleContract(address newAssembleContractAddr_)
    public
    virtual
{
    assembleContract = AssembleInterface(newAssembleContractAddr_);
}

/**
 * @param imageId_ The token ID for which you want to set the attribute value.
 * @dev Set the attribute value of the corresponding `imageId_` sequentially according to
 */
function _setAttr(uint256 imageId_) public virtual {
    for (uint256 idx = 0; idx < assembleContract.getStorageCount(); idx++) {
        uint256 newValue = 0;

        /// @dev Implement the property value setting logic.

        _attrs[imageId_].push(newValue);
    }
}
```

[그림 3] Property Layer의 소스코드

```
/**
 * @dev See {IAssembleContract-getImage}
 */
function getImage(uint256[] memory attrs_)
    external
    view
    virtual
    override
    returns (string memory)
{
    string memory imageString = "";

    imageString = string(
        abi.encodePacked(
            imageString,
            // '<svg version='1.1' xmlns='http://www.w3.org/2000/svg'>'
            '<?xml version="1.0" encoding="UTF-8" standalone="no"?>'
            '<svg xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:cc="http://web.reso
-syntax-ns#" xmlns:svg="http://www.w3.org/2000/svg" xmlns="http://www.w3.org/2000/svg" x
ns:sodipodi="http://sodipodi.sourceforge.net/DTD/sodipodi-0.dtd" xmlns:inkscape="http://
ght="1018" viewBox="0 0 1901 1018" id="svg2" sodipodi:version="0.32" inkscape:version="0
/inkscape/share/examples" sodipodi:docname="gallardo.svg" inkscape:export-filename="C:D
ruresgallardo.png" inkscape:export-xdpi="90" inkscape:export-ydpi="90" inkscape:output_e
ipodi:modified="true">'
        )
    );

    for (uint256 i = 0; i < attrs_.length; i++) {
        imageString = string(
            abi.encodePacked(
                imageString,
                _assets[i].stock.getAsset(attrs_[i])
            )
        );
    }

    imageString = string(abi.encodePacked(imageString, "</svg>"));

    return imageString;
}
```

[그림 4] Assemble Layer의 소스코드



[그림 5] Storage Layer의 스마트 컨트랙트들과 소스코드

구현을 위해 이더리움 Rinkeby 네트워크에 배포한 스마트 컨트랙트의 주소 값들은 표 2와 같다.

[표 2] 이더리움 Rinkeby 네트워크에 배포된 스마트 컨트랙트 주소

Property Layer	01	0x91f1CBee66aaAf53b924372CC5b5eF3552adA8dd
Assemble Layer	01	0xCbD0E4aF557C502aB4823a1F3E273A2BC0Baeee4
Storage Layer	01	0x4dcF357CA7E7F89F5Ee94afff49f2CD0EDeD625B
	02	0x868d875DA976dddEe452ea04D0831721Aa1d0DBd
	03	0xd14DDA64dA78456CB0015D90d7c2270c1459AcCC
	04	0xd894B6eB69FA387c626C0c332D80474a88A01042
	05	0x6098A098E2FE7566ac60bB4a4f9c12A8B8A40153
	06	0xcb5E8d5B35938E2F73542B093495200E74F7D612
	07	0xE42Cf9E0aD35f948EA076fd4244B681DA7EaA77E
	08	0xb035Fa80AC6c73d7c16f6360AcD94636706be7B1
	09	0x24DDdb5cCe386D7F5806cA3D5ed0F71f7985DfEA
	10	0xbC40C6E32FCD2B48759eBf8935030A524B4164F2
	11	0x96ED2b7d5c7faE9AfC9fafd6d69005Dc36c31Ca6
	12	0xcf13bb157370111AFE92f7a2B0cAF9727c728FA6
	13	0x907b1695224E7867532395c0Eb2Bb9a98fE260d6
	14	0x2Dc65FdB0351a9e142E741768882e6b288c17306
	15	0x1995f890564572O7BBa1C119A6E26F39115DBca5
	16	0x5139aEB55BE01b3fFA63aaEC440644B294f427C2
	17	0x240BCd2b759F321778487699429752 3A553F8dFd

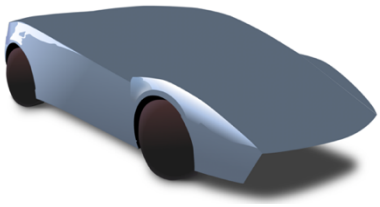
이후, 그림 6과 같이 블록체인 네트워크상에 배포된 샘플 이미지를 가져오기 위한 스크립트를 작성하였으며, Storage Layer에 분할 저장된 XML 태그들을 필요에 따라 유연하게 조합하여 사용할 수 있는지(그림 7, 그림 8, 그림 9)와 배포된 이미지를 온전히 불러올 수 있는지(그림 10)를 확인하였다.

```
import { ethers } from "ethers";
import fs from "fs";
import rpcProvider from "./config/rpcProvider.js";
import abiList from "./config/abi.js";
import addressList from "./config/addressList.js";

// Set Network Provider
const provider = new ethers.providers.JsonRpcProvider(
  rpcProvider,
  4 // rinkeby
);

// Set Contracts (Property)
const EOSIS = new ethers.Contract(
  addressList.property,
  abiList.property,
  provider
);

// Get Image
const imageId = 0;

EOSIS.getImage(imageId).then((result) => {
  fs.writeFileSync("./assembled.svg", result, { encoding: "utf-8" });
});
```

[그림 6] 이미지를 가져오기 위한 스크립트

표 2의 Storage Layer에 포함된 17개의 스마트 컨트랙트 중, 그림

7은 1~5번째, 그림 8은 6~10번째, 그림 9는 1~10번째 스마트 컨트랙트에 저장된 XML 태그들을 조합하여 생성한 결과 이미지이다.



[그림 7] Storage Contract 1~5번째 스마트 컨트랙트 조합한 결과

[그림 8] Storage Contract 6~10번째 스마트 컨트랙트 조합한 결과

[그림 9] Storage Contract 1~10번째 스마트 컨트랙트 조합한 결과

그림 10은 표 2의 Storage Layer에 속한 모든 스마트 컨트랙트에 저장된 XML 태그들을 조합하여 생성한 결과이다.

[그림 10] Storage Layer의 모든 스마트 컨트랙트를 조합한 결과

3. 결 론

본 연구는 블록체인 네트워크상에 이미지를 영구적으로 저장하고 변조를 방지할 수 있으면서, 확장 가능한 SVG 이미지 스토리지 모델을 구상 및 제시하였다. 이 스토리지 모델을 활용해 큰 용량의 이미지 파일을 온체인 상에 저장하고, 3개의 계층으로 분리함에 따라 스토리지의 확장성과 유연성을 확보할 수 있을 것으로 기대한다.

그러나, 301KB 크기의 이미지 파일을 온체인 상에 기록하는 데 있어, 17개의 스마트 컨트랙트와 약 0.8 ETH (3 Gwei 기준)의 가스비가 소모된 만큼, 배포되는 스마트 컨트랙트 수와 그에 따른 비용을 절감할 수 있는 개선 방안에 대한 추가 연구가 필요하다.

4. 참고 문헌

[1] Vitalik Buterin “Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform” (2014)

[2] William Entriken, Dieter Shirley, et al. “EIP-721: Non-Fungible Token Standard” (2018) Accessed: 2022-05-10, https://eips.ethereum.org/EIPS/eip-721

[3] Juan Benet “IPFS – Content Addressed, Versioned, P2P File System (DRAFT 3)” (2014)

[4] IPFS “Persistence, permanence, and pinning”, Accessed: 2022-05-12, https://docs.ipfs.io/concepts/persistence/#persistence-versus-permanence

[5] Vitalik Buterin “EIP-170: Contract code size limit” (2016) Accessed: 2022-05-10, https://eips.ethereum.org/EIPS/eip-170

[6] Chris Lilley, Dean Jackson “About SVG – 2d Graphics in XML” W3C (1999) Accessed: 2022-04-18, https://www.w3.org/Graphics/SVG/About.html

[7] Jake Giltsoff “SVG on the web” (2015) Accessed: 2022-04-22, https://svgontheweb.com/ko/