Võ Trường Trung Hiếu
Github: https://github.com/UPI05/Proof-of-training-Quality

Bản tóm tắt hiện thực giao thức Proof of training Quality

Bài báo tham khảo: https://ieeexplore.ieee.org/abstract/document/8843900
Source code hiện thực: https://github.com/UPI05/Proof-of-training-Quality

1. Future works & Notes:

1. Future works

- Longest-chain rule, chain sẽ bị split thành 2 chain đúng khi có 2 node trong mạng cùng broadcast block hợp lệ. Handle theo kiểu của giao thức nakamoto hơi phức tạp nên khi có 2 node đồng thời gửi block hợp lệ thì mình sẽ chọn node nào đăng kí trước trong blockchain làm proposer.
- Cần xử lý ngoại lệ try - catch để tránh node bị crash.
- Những message liên quan đến 1 request lưu trong pool cần xóa hoạt unSpent khi thực hiện xong (Garbage collector).
- Hiện tại, chain data đang lưu trong memory, chưa ghi ra đĩa.
- Một số chỗ cần giảm data lưu vô pool. Ví dụ như getChainRes chỉ cần lưu giá trị hash của chain vào pool.
- Khi nhận message dataSharingRes, một node có thể gửi nhiều lần blockVerifyReq bằng với số lượng message dataSharingRes nhận được. Cần cải tiến để chỉ gửi 1 lần. Tương tự cho blockVerifyRes.

2. Notes

- Trong bài báo, họ chọn ra node proposer là node có giá trị MAE^u nhỏ nhất:

$$MAE^u(P_j) = \gamma \cdot MAE(m_j) + \frac{1}{n}\sum_i MAE(m_i),$$

  nhưng trong phần implement, mình bỏ giá trị Gama cho đơn giản vì nó liên quan đến topology, vẫn đảm bảo được node nào có model chính xác nhất làm proposer.
- Vì cặp (publicKey, privateKey) được sinh ra từ SECRET nên SECRET tốt nhất phải unique. Có thể sử dụng crypto hash của {một giá trị ngẫu nhiên + timestamp}.
- Khi mạng ít hoạt động, longest-chain rule có bị attack? Nếu 1 node cố tình tạo chain sai bằng cách xem như các node khác trong committee đều không hoạt động, như vậy chain của attacker sẽ dài ra rất nhanh tuy nhiên validators lại chưa có cách nào để kiểm tra vì data nằm trong chain là của quá khứ.

- Nếu không node nào trong mạng có category ứng với dataSharingReq.requestCategory thì sẽ không có dataSharingRes message gửi về, tức là request đó sẽ không được thực thi.
- Trong code, việc gọi đệ quy để tạo message và verify giúp code nhìn gọn hơn nhưng dễ gây ra nhầm lẫn giữa transaction, block và message. Block được sinh ra từ message.
- Nếu publicKey cho dataRetrieval đã tồn tại, coi như cập nhật category.

2. Hiện thực giao thức PoQ:

Note: Trước khi đọc phần này nên đọc phần 4.

Mô tả quá trình gửi, nhận message (PoQ):

Trong bài báo đề cập 2 loại transaction là Data Retrieval và Data Sharing.

- Data retrieval transaction: Đăng kí một node mới tham gia vào mạng. Dùng để lưu thông tin của node khi tham gia vào mạng: public key, category, data size, data type…
- Data sharing transaction: Dùng để tạo request lấy model. Trong block nó là một trường để lưu trữ tất cả data phát sinh trong quá trình thực hiện một request từ client. Bao gồm các message, các model và số MAE của các node trong committee, các chữ kí, … Khi một node response với model và MAE, mình không xem nó như một transaction mà xem nó như một message trong quá trình đồng thuận của các p2p server.

Để implement, mình chia thành các message sau (để xử lý các tín hiệu qua lại giữa các node):

Các loại message:

- Cho data retrieval transaction:
  - dataRetrieval: Đăng kí node mới vào mạng, bao gồm public key và category cho node. (Theo bài báo thì có thể có thêm Type và Size của data).

- Cho data sharing transaction:
  - dataSharingReq (Data sharing request): Gửi data query và category
  - dataSharingRes (Data sharing response): Gửi lại model và giá trị MAE
  - blockVerifyReq (Block verify request): Node proposer sẽ tạo block và yêu cầu các node khác trong committee verify block đó.
  - blockVerifyRes (Block verify response): Mỗi node trong committee nếu verify hợp lệ sẽ gửi message response có kèm chữ kí của mình.
  - blockCommit (Block commit): Nếu proposer thu thập đủ chữ kí của tất cả các node trong commitee (những node bị crash thì không cần) thì nó gửi message này để yêu cầu các node trong mạng lưu block vào blockchain của node.

Khi một message được gửi đến một node, node đó phải kiểm tra xem nó đã từng nhận message đó hay chưa, nếu chưa nhận thì mới xử lý (đồng thời broadcast cho node khác xử lý). Như vậy mình cần một nơi để lưu trữ các message vì nếu đã nhận rồi mà còn broadcast message đó đi cho các node khác thì message đó sẽ bị broadcast vô tận trong mạng, hơn nữa mình cũng cần lưu để khi nhận đủ message mới xử lý bước tiếp theo (chẳng hạn như muốn tạo block thì phải nhận đủ số message loại dataSharingRes). Ở đây mình sẽ dùng 1 cái pool gọi là messagePool để lưu tất cả message nhận được.

Trong bài báo, request chỉ gửi từ client đến các node trong committee, committee node được tìm thông qua quá trình Multi-party retrieval. Tuy nhiên, implement thì mình sẽ broadcast toàn mạng, node nào nằm trong committee thì nó sẽ response, để tránh bị attack thì mình sẽ verify lại ở các bước sau. *Chỗ này là một vấn đề nếu muốn cải tiến vì trong bài đề cập topology khá phức tạp (chưa đọc kĩ).*

3. Flow khi implement:

- Khi một node tham gia vào mạng, nó phải broadcast message dataRetrieval kèm với public key, category của nó, đồng thời gửi kèm chữ kí của tổ chức thứ 3 ngụ ý rằng nó đã được đồng ý cho vào mạng. Khi các node trong mạng nhận được message này, chúng sẽ verify chữ kí hợp lệ thì lưu message này vào messagePool và đánh dấu là unSpent.
- Khi client gửi request lên hệ thống, tại node client (client muốn tương tác đến hệ thống thì nó phải là 1 node đã được xác thực, tức là client là 1 p2p server và giữ 1 blockchain; khác với bài báo client phải gửi request đến Super Node), nó sẽ broadcast dataSharingReq message kèm với data query mà nó cần, category, chữ kí…
- Khi các node nhận được dataSharingReq được gửi ở trên, nó sẽ kiểm tra trong messagePool đã tồn tại hay chưa. Nếu tồn tại rồi thì không xử lý. Nếu verify message (bao gồm chữ kí, data…) không hợp lệ cũng không xử lý. Nếu chưa tồn tại thì nó lưu vào messagePool và đánh dấu là unSpent, đồng thời broadcast message mà nó nhận đi để các node khác có thể nhận được. Sau đó, nó check thêm nếu requestCategory của message gửi đến trùng với category của nó hay không, nếu trùng tức là message đó gửi đến committee của nó nên nó phải xử lý. Ở đây, nó sẽ sinh ra model, đánh giá ra số MAE và kí vào, gửi kèm dataSharingReq mà nó nhận để broadcast đi cho toàn mạng, message dùng là dataSharingRes.
- Khi các node nhận dataSharingRes message được gửi ở trên, nó cũng sẽ kiểm tra trong messagePool xem đã tồn tại hay chưa. Nếu tồn tại rồi thì không xử lý. Nếu verify message không hợp lệ cũng không xử lý (bao gồm chữ kí, số MAE…). Nếu chưa tồn tại thì nó sẽ lưu vào messagePool đồng thời đánh dấu là unSpent. Nếu nó có category liên quan và nó đã nhận đủ số dataSharingRes ứng với tất cả các node còn sống của committee thì nó phải kiểm tra xem số MAE (đại diện cho độ chính xác của model, càng nhỏ càng chính xác) của nó có phải là nhỏ nhất hay không, nếu nhỏ nhất tức là nó là node proposer. Khi đó nó sẽ tạo block mới và add tất cả data liên quan từ lúc

client gửi request bao gồm tất cả các message liên quan từ messagePool rồi broadcast đi toàn mạng với message blockVerifyReq.

- Khi các node nhận blockVerifyReq message được gửi ở trên, nó sẽ kiểm tra trong messagePool xem đã tồn tại hay chưa. Nếu tồn tại thì không xử lý. Nếu verify message không hợp lệ cũng không xử lý (bao gồm chữ kí của tất cả transaction trong block, MAE, preHash…) Nếu chưa tồn tại thì lưu vào messagePool và đánh dấu unSpent. Nếu message requestCategory thuộc committee của nó thì nó sẽ verify block đó (bao gồm chữ kí của proposer, transactions, MAE,...). Nếu hợp lệ nó sẽ kí và broadcast đi toàn mạng với message blockVerifyRes.
- Khi các node nhận blockVerifyRes message được gửi ở trên, nó sẽ kiểm tra xem trong messagePool đã tồn tại hay chưa. Nếu tồn tại rồi thì không xử lý. Nếu verify message không hợp lệ cũng không xử lý (bao gồm chữ kí của tất cả transaction trong block, MAE, preHash…), đặc biệt kiểm tra chữ kí của node gửi message xem có cùng committee hay không (các thông tin này được đăng kí trên chain thông qua Data retrieval transaction). Nếu chưa tồn tại thì lưu vào messagePool và đánh dấu là unSpent. Nếu block mà message blockVerifyRes gửi về mà có trường proposer là public key của nó, tức là nó là người request verify cho block đó. Lúc này nó cần đếm xem số blockVerifyRes cho block mà nó request để verify nhận được đã đủ chưa (chữ kí đôi một khác nhau), nếu đủ và hợp lệ, nó sẽ gôm chữ kí lại, broadcast kèm block với message là blockCommit.
- Khi các node nhận blockCommit message được gửi ở trên, nó cũng sẽ kiểm tra xem trong messagePool đã tồn tại hay chưa. Nếu tồn tại rồi thì không xử lý. Nếu verify message không hợp lệ cũng không xử lý (bao gồm chữ kí của tất cả transactions, proposer, MAE, preHash, MAE của proposer có phải là nhỏ nhất, kiểm tra chữ kí của committee cho block,...). Nếu mọi thứ hợp lệ thị bất kể nó có phải thuộc committee đó hay không cũng phải add block đó vào blockchain của mình, đồng thời broadcast message đó đi cho node khác xử lý.
- Sau khi 1 block được add xong, để tiếp tục hoạt động nó cần xử lý các message trước đó đã add vào messagePool bằng cách gán unSpent = false. Mình không xóa các message mà chỉ đánh dấu nó.

Note: Cần sử dụng heartBeat để kiểm tra các node còn sống (Xem phần 5).

Đương nhiên là khi các node nhận một message đến nó phải verify message đó có hợp lệ hay không thông qua chữ kí, các transaction ở trong… và khi gửi một message nó phải kí vào đó. Các message khi add vào pool cũng phải có trường messageType hay gì đó để phần biệt các loại message với nhau.

Khi một số message bị hủy, không đạt được kỳ vọng, nó sẽ tồn tại mãi trong pool. Vì vậy mình cần có một cơ chế nào đó để đánh dấu unSpent = false cho nó. Ví dụ như đợi 10 phút mà chưa được Spent thì đánh dấu nó Spent để hủy chẳng hạn.

Cần một bên thứ ba kí thì Data retrieval transaction mới hợp lệ (tức là node đó phải được tổ chứ thứ 3 cho phép tham gia). Giải pháp? Mình sẽ đóng vai trò là bên thứ ba cho hệ thống. Một cặp (public, private) key sẽ được sinh ra và public key được lưu thẳng vào genesis

block. Khi muốn cho node mới tham gia, mình sẽ dùng private key để kí và gọi REST API cho data retrieval transaction. Đây là giải pháp tạm thời vì blockchain hoạt động mãi nhưng private key có thể bị mất.

Thế nào là message tồn tại? Dựa vào giá trị hash và loại message.
Chữ kí hợp lệ? Chữ kí khớp với public key và public key đó đã được đăng kí trong blockchain với category hợp lệ.

4. Note: *Phải phân biệt được transaction vs message:*

Phần này do mình định nghĩa để implement dễ dàng hơn, có thể là không chính xác trong một số trường hợp.

Transaction thường liên quan đến client, dùng để thay đổi giá trị trong blockchain. Khi client tạo request loại Retrieval hoặc Data sharing thì đó là transaction.

Message (trong bài báo gọi là event) liên quan đến giao thức đồng thuận, nó là loại tín hiệu mà các p2p server gửi qua lại với nhau để đạt được sự đồng thuận cho 1 Transaction.

Tóm lại, transaction là khái niệm khi người dùng call REST APIs đến p2p server trong khi message là khái niệm được sử dụng giữa các p2p server với nhau.

- *1 block sẽ lưu 1 transaction (Vì khi 1 transaction request được gửi lên thì ngay lập tức node nào có MAE response nhỏ nhất sẽ đóng block cho transaction đó nên mình không thể để nhiều transaction trong 1 block được). Một transaction sẽ bao gồm tất cả message liên quan đến transaction đó: dataRetrieval, dataSharingReq, dataSharingRes, blockCommit…*

- *Để dễ cài đặt, mình sẽ xem Data retrieval transaction như là một massage. Tức là khi một Data retrieval transaction được gửi lên, mình sẽ không trigger quá trình đồng thuận để đóng block mà thay vào đó mình sẽ lưu nó dưới dạng message trong 1 Data sharing transaction. Như vậy chỉ khi nào có Data sharing transaction thì Data retrieval transaction đó mới được add vào blockchain.*

5. Thêm node mới vào network / Restart node

Khi 1 node tham gia vào mạng hay restart, nó sẽ tạo ra 1 chain với 1 genesis block và 1 wallet chứa cặp (public key, private key). Sau đó broastcast tín hiệu trong mạng để lấy chain đúng nhất về (chain đúng nhất là chain dài nhất) so sánh với chain nó đang giữ (chỉ có 1 block). Nếu là thêm node mới, client cần call API để truyền signature của tổ chức thứ 3 (ngụ ý node đã được cấp phép tham gia mạng).

Nếu xem dataRetrieval là một transaction thì khi nó được gửi đi, node nào sẽ là proposer? Trong bài báo đề cập khi gửi dataSharing transaction thì node nào có model đúng nhất sẽ là proposer, tuy nhiên lại không đề cập đến dataRetrieval.

Nếu xem dataRetrieval là một message được add kèm theo dataSharing transaction thì khi có một transaction loại dataSharing được tạo, dataRetrieval sẽ được add theo vào trong chain. Tuy nhiên, để làm theo cách này, mình cần tạo sẵn một số node trước khi hệ thống hoạt động. Bao nhiêu node là đủ? => 2.

5.1 Kiểm tra các node còn sống:

Khi một node nhận được message loại dataSharingRes, nó sẽ cần lấy thông tin của những node còn sống trong committee. Vì vậy mình sẽ sử dụng thêm 2 loại message:

- heartBeatReq: Yêu cầu các node còn sống gửi lại mesage heartBeatRes.
- heartBeatRes: Các node còn sống reply lại khi nhận heartBeatReq.

Như vậy, khi nhận được dataSharingRes, node cần gửi heartBeatReq và đợi các message heartBeatRes trong một khoảng thời gian HEARTBEAT_TIMEOUT.

5.2 Lấy chain đúng nhất:

Khi cần lấy chain đúng nhất (trường hợp restart hoặc muốn propose block mới), node trong mạng sẽ cần dùng 2 loại message sau:

- getChainReq: Yêu cầu các node gửi lại chain của mình đang giữ.
- getChainRes: Các node reply kèm theo chain của mình.

Sau khi nhận được các chain, mỗi node chỉ cần validate chain đó hợp lệ thì chain nào dài nhất sẽ được sử dụng.

6. REST APIs:

REST APIs chạy ở port 300**x.**

GET /blockchain
Lấy chain ở node hiện tại.

GET /message
Lấy tất cả các message trong messagePool ở node hiện tại.

POST /register
{
“signature”: *string*
}

Đăng kí một node mới vào mạng.
signature là chữ kí của tổ chức thứ ba (bên cấp phép tham gia). Là giá trị: **sign(node.publicKey())**. Với cặp (private key, public key) do node đó tự tạo.

POST /request
{
	“requestModel”: *model*,
	“requestCategory”: *string*
}

Request một model ứng với *category* và truyền input là *data* cho model.

7. Cấu trúc block:

7.1: Genesis block


```
{
        "timeStamp": "010101010101",
        "msgType": "GenesisBlock",
        "publicKey": "Hieu Vo",
        "other": {
            "registerPublicKey":
"d0233e2fbc91aeb97d462d71b05b6522a01c838249fb93b0ac57bb6965062cb4"
        },
        "transaction": {
            "messages": [
                {
                    "publicKey":
"5864f1e7d7e37e95882b398c21ca29b314ebfc6ea1286c8dc1201214cc0d0686",
                    "category": "chicken",
                    "msgType": "DataRetrieval"
                },
                {
                    "publicKey":
"276ab32fb62ccc18fcb9c0e792092b9b6911e53b75f951211b232cf84de061bf",
                    "category": "tiger",
                    "msgType": "DataRetrieval"
                }
            ]
        },
        "hash": "0xDEADBEEF",
        "signature": "VH"
}
```

Trong đó `registerPublicKey` dùng để verify một node mới tham gia vào mạng đã được cấp phép hay chưa. Tức là tổ chức nắm giữ *registerPrivateKey* đã kí nhận.

7.1: Block thường:

Xem thêm ở https://github.com/UPI05/Proof-of-training-Quality

8. Setup và thêm node:

8.1 Setup

Vì là permissioned blockchain, tổ chức thứ ba setup hệ thống cần một cặp (publicKey, privateKey) để sau này cấp phép cho các node mới muốn tham gia. Giá trị publicKey cần gán vào `GENESIS_OTHER.registerPublicKey` trong file *config.js*. Để hệ thống hoạt động, cần khởi tạo trước 2 node với privateKey (SECRET) của node cũng do tổ chức đó nắm giữ:

```
CATEGORY='chicken' SECRET='NODE1' HTTP_PORT=3000 P2P_PORT=5000 node
app.js
CATEGORY='tiger' SECRET='NODE2' HTTP_PORT=3001 P2P_PORT=5001
PEERS='ws://localhost:5000' node app.js
```

8.2 Thêm node vào mạng

Trước tiên, giá trị `GENESIS_OTHER.registerPublicKey` trong file *config.js* phải trùng với publicKey của tổ chức cấp phép.

Sau đó node sẽ setup với lệnh:

```
CATEGORY='lion' SECRET='node_secret_key' HTTP_PORT=3002 P2P_PORT=5002
PEERS='ws://localhost:5000,ws://localhost:5001' node app.js
```

CATEGORY và SECRET, cũng như các PORT và PEERS do mỗi node tự customize. Tuy nhiên cần phải truyền ít nhất 1 giá trị trong PEERS (là địa chỉ của những node đang hoạt động có thể kết nối đến).

Bây giờ, node đã connect đến mạng nhưng vẫn chưa thể tạo request - ví dụ như lấy chain (Vì chưa được cấp phép).

Để grant permisstion, tổ chức thứ ba cần kí vào publicKey của node muốn tham gia bằng privateKey của họ.

Signature này sau đó được node gửi đến mạng với dataRetrieval message khi gọi API /register của node (Xem phần 6).

Đến bước này, client có thể tạo request đến node thông qua REST APIs và hoạt động bình thường.

8.3 Production and Simulation:

Xem thêm ở https://github.com/UPI05/Proof-of-training-Quality

9. Longest-chain rule vulnerability:

Khi một node nhận được message response cho dataSharingReq hoặc blockVerifyReq, nó cần gửi heartBeatReq để kiểm tra xem đã đủ message response của các node còn sống gửi về hay chưa.

Tuy nhiên, khi các node trao đổi chain trong mạng, mình không thể kiểm tra một block nào đó có nhận đủ message response của các node còn sống tại thời điểm block đó được tạo hay không. Điều này gây khó khăn khi sử dụng longest-chain rule. Giả sử một node cố gắng tạo ra chain dài nhất để thay thế cho chain của hệ thống. Việc này rất đơn giản bằng cách tạo request cho chính category của nó, khi đó nó sẽ xem như các node khác trong category đó đều đã ngưng hoạt động nên tự mình làm proposer cho block đó. Node attacker cứ lặp lại công việc trên đến khi nào chain đủ dài hơn chain của hệ thống thì gửi chain đó lên mạng. Các node trong mạng chẳng có cách nào để verify tại một thời điểm nào đó (thời điểm block được tạo) có bao nhiêu node còn sống cả.

Ý tưởng:

Mỗi block không được tạo tại thời điểm ngẫu nhiên nữa mà thay vào đó mình sẽ tạo block vào các thời điểm 5p, 10p, 15p, 20p, …60p (tương tự cho request).

Sử dụng longest-chain rule đi kèm với tiêu chí khác là mỗi block trong chain phải có càng nhiều message response càng tốt, mỗi request cũng có càng nhiều response càng tốt.

10. Models similarity:

Khi một node gửi model để các node trong committee thực hiện transfer learning, nó phải gửi kèm test data để đảm bảo các node có thể đồng bộ trong việc xác định model nào chính xác nhất. Tuy nhiên một node cố tình cheat hệ thống có thể cố tình train ra model mới để overfit với test data. Vì vậy chúng ta cần xác định miền giá trị mà nằm ngoài vùng đó, ta xem như model đó không tốt. Ở đây, mình sẽ lấy trong đoạn:

$$[MEAN - STD - \varepsilon, MEAN + STD + \varepsilon]$$

Với MEAN là giá trị trung bình MAE của các model, STD là độ lệch chuẩn. Mình sử dụng thêm ε để tránh những model tốt bị loại bỏ (trường hợp tất cả các model đều tốt). Ví dụ, MAE có thang điểm 10 thì mình có thể lấy ε khoảng từ 0.5 đến 1.

11. Thực nghiệm:

1. Đánh giá thời gian sinh ra 1 block (tính từ lúc client tạo request):

Ước tính:

Δ = transferLearningTime + modelAggregationTime + modelsMAEVerificationTime + ε

Với **ε** là thời gian kiểm tra, xác thực của chương trình, độ trễ mạng, heartbeat_timeout…

Kết quả thực nghiệm (không tính thời gian thực hiện Federated Learning):