Descrizione
BitCoin Miners' Interactions Viewer

Introduzione

Il progetto si pone l'obiettivo di visualizzare le interazioni tra i minatori di bitcoin, analizzando l'intero storico delle transazioni. E' stato richiesto di realizzare sia una visualizzazione basata su fenomeni macroscopici, analizzando le interazioni temporali a cadenze mensili/annuali, che microscopici, a fasce orarie della singola giornata.

Utilizzo

Per utilizzare l'applicazione è sufficiente scaricare il progetto da github ed avere nodejs e mongodb installati sulla propria macchina. Dalla cartella Visualizzazione è sufficiente avviare un server node in tal modo "nodejs app.js" e recarsi su localhost:3000 dal browser per accedere da client.

Se si è interessati ad aggiornare le informazioni è necessario scaricare ed installare bitcoind ed avviare il file main.py all'interno della cartella Preprocessamento.

Preprocessamento

Le informazioni dei bitcoin sono accessibili attraverso l'installazione di bitcoind, il software ufficiale con cui è possibile sia partecipare alle minate e gestire il proprio wallet, sia effettuare query nello storico. Una volta che la copiosa mole di dati (in binario) è stata scaricata dalla rete, è stato realizzato un sistema in python che raccogliesse i dati relativi ai miners e che li memorizzasse all'interno di MongoDB[1].
Il sistema è così costituito:

- main.py: è sufficiente avviare tale file per aggiornare le informazioni del database
- nextBlockToRead: file testuale che detiene il prossimo blocco che il sistema deve leggere. Se si desidera riprocessare, è sufficiente inserire il numero 1 al suo interno.
- macro.py: gestisce le interazioni con la shell, permettendo di effettuare le query a bitcoind, restituendo l'output
- transaction.py: modella l'oggetto "transazione", memorizzandone le informazioni rilevanti (blocco di appartenenza, bitcoin in ingresso, bitcoin in uscita, indirizzi in ingresso, indirizzi in uscita)

[1] E' stato selezionato questo tipo di database sia per l'efficienza temporale delle query sui dati, sia perchè il formato di dati è comune con l'output di bitcoind (JSON)

- address.py: modella l'oggetto "indirizzo/minatore", memorizzandone le informazioni rilevanti (indirizzo hash, numero di minate, transazioni di mining, transazioni di pagamento, transazioni di credito)
- dao.py: classe che gestisce le interazioni con MongoDB.
- bitcoinParser.py: costituisce il controller del progetto.

Visualizzazione

Una volta raccolte, le informazioni devono poter essere visualizzate. In particolare, è stato scelto di rappresentare due grafi, uno per il macro (minatori come nodi e archi come interazioni) ed uno per il micro (i nodi verdi costituiscono i blocchi, i nodi rossi i minatori ed esiste un arco tra un nodo verde ed uno rosso se il minatore ha minato in quel blocco), accompagnati da diversi filtri che permettessero di filtrare le informazioni visualizzate.
In particolare sono stati realizzati i seguenti filtri:

- **Time Stamp Blocco**
  Il primo filtro agisce solo sul primo grafo e permette di scegliere un range temporale sulla base del quale verranno filtrati i blocchi da analizzare
- **Mining Count dell'Address**
  Questo filtro permette di filtrare i nodi, sia del primo che del secondo grafo, sulla base di quante volte, un minatore abbia minato, nella storia.
- **Miner nel blocco**
  Quest'altro filtro sui blocchi, agisce anch'esso su entrambi i grafi, e permette di filtrarli in base al numero di minatori che l'hanno "minato"
- **Numero di collaborazioni tra miner**
  Questo filtro agisce solo sul primo grafo ed il primo filtro che agisce solo sugli archi. Infatti permette di definire un numero minimo di interazioni che due nodi devono aver avuto per far sì che l'arco sia visualizzato.
- **Blocks Date e Fascia Oraria**
  L'ultimo filtro serve a definire una fascia oraria sul secondo grafo. Ne è stato realizzato uno diverso dal primo, perché il tipo di informazioni visualizzate sul secondo grafo devono essere riferite ad una fascia temporale più piccola. Ovvero, al massimo, di una giornata.

E' stata realizzata una architettura client-server che permettesse l'uso della libreria D3js.
L'intera logica server è stata raccolta tutta all'interno del file app.js, mentre quella client è stata suddivisa in 3 file:

- graphD3.js: costituisce la logica relativa alla visualizzazione, ossia all'inizializzazione dell'SVG ed al disegno dei nodi e degli archi del grafo
- graphLogic.js: preleva le informazioni dal server (che a sua volta le estrae da mongodb) e le filtra sulla base dell'input
- filters.js: gestisce la logica relativa alla rappresentazione degli sliders