# A jornada inicial
# de um node Bitcoin

Discord - Bitcoin Coders
https://discord.com/invite/e5qUsNWgQg

Anfitrião:
Rafael Penna

bitcoinCoders bitups

# O Que Veremos!

- Introdução
- É só baixar arquivos?
- O primeiro contato com a rede
- Reconstrução do estado
- Por que sincronizar demora tanto?
- Por que reconstruir tudo desde 2009?
- Novas propostas

# O Que Veremos!

- Introdução
- É só baixar arquivos?
- O primeiro contato com a rede
- Reconstrução do estado
- Por que sincronizar demora tanto?
- Por que reconstruir tudo desde 2009?
- Novas propostas

# Introdução

- Você instala o Bitcoin Core
- Abre o programa conectado à internet
- Software
  - Encontra peers
  - Começa a baixar dados
  - Exibe o progresso (horas ou dias)
- Etapa parece simples
  - Node está baixando a blockchain

# Introdução

- Processo complexo
  - O que um node Bitcoin precisa aprender para se tornar parte da rede?
  - Por que a sincronização demora tanto?
  - Por que desenvolvedores estão discutindo projetos?
    - AssumeUTXO
    - Utreexo
    - SwiftSync

# Introdução

- Vamos:
  - Acompanhar um node recém-instalado
  - Estado atual do Bitcoin
    - Entender como ele:
      - Descobre
      - Valida
      - Reconstrói

# O Que Veremos!

- Introdução
- É só baixar arquivos?
- O primeiro contato com a rede
- Reconstrução do estado
- Por que sincronizar demora tanto?
- Por que reconstruir tudo desde 2009?
- Novas propostas

# É só baixar arquivos?

- Rodar um node → armazenar uma cópia da blockchain
  - Basta baixá-la e pronto?
  - Node faz muito mais que isso
- Quando um bloco é recebido o node verifica
  - se o Proof of Work é válido
  - se o bloco referencia corretamente o bloco anterior
  - se as transações obedecem às regras de consenso
  - se nenhuma moeda está sendo gasta duas vezes
- O node não está tentando descobrir quais blocos existem
- Está tentando descobrir qual é o estado correto do sistema

# O Que Veremos!

- Introdução
- É só baixar arquivos?
- O primeiro contato com a rede
- Reconstrução do estado
- Por que sincronizar demora tanto?
- Por que reconstruir tudo desde 2009?
- Novas propostas

# O primeiro contato com a rede

- Quando um node é executado pela primeira vez
  - Precisa encontrar outros participantes da rede
  - Estabelece conexões
  - Começa a baixar os cabeçalhos dos blocos

- Somente depois → Initial Block Download (IBD)
  - O node percorre toda história do Bitcoin e cada bloco é
    - Baixado
    - Verificado
    - Aplicado localmente

# O primeiro contato com a rede

- O node refaz por conta própria todas as verificações que milhares de outros nodes já fizeram ao longo dos anos
- Podemos observar esse processo através do Bitcoin Core:

```
bitcoin-cli -datadir="." getblockchaininfo
```

- Entre os campos retornados estão:

```
{
  "blocks": 952562,
  "headers": 955807,
  "verificationprogress": 0.9905729213921609,
  "initialblockdownload": true,
}
```

# O primeiro contato com a rede

- initialblockdownload == true
  - Node ainda considera que está sincronizando
- Quantos blocos foram baixados?
- O que exatamente o node está tentando construir ao longo desse processo?

# O Que Veremos!

- Introdução
- É só baixar arquivos?
- O primeiro contato com a rede
- Reconstrução do estado
- Por que sincronizar demora tanto?
- Por que reconstruir tudo desde 2009?
- Novas propostas

# Reconstrução do estado

- Bitcoin → Normalmente pensamos na blockchain/timechain
- Blockchain → histórico
- O que realmente importa para um node
  - Conjunto de UTXOs

# Reconstrução do estado

- UTXO / Unspent Transaction Output
  - Todas as moedas que ainda podem ser gastas
  - Imagine
    - Toda a história do Bitcoin desaparecendo
    - Lista perfeita contendo todos os UTXOs existentes naquele instante
    - Daria para saber exatamente
      - Quem pode gastar moedas
      - Quanto existe disponível
      - Qual é o estado atual do sistema

# Reconstrução do estado

- Objetivo final do IBD
  - Não é armazenar blocos
  - É reconstruir o UTXO Set

- Podemos observar esse estado através do comando:

  bitcoin-cli -datadir="." gettxoutsetinfo

- A saída inclui informações como:

```
{
  "height": 953672,
  "txouts": 165789719,
  "total_amount": 20042498.01493958
}
```

# Reconstrução do estado

- A blockchain é o histórico
- O UTXO Set é o estado
- E a sincronização existe para transformar o primeiro no segundo

# O Que Veremos!

- Introdução
- É só baixar arquivos?
- O primeiro contato com a rede
- Reconstrução do estado
- Por que sincronizar demora tanto?
- Por que reconstruir tudo desde 2009?
- Novas propostas

# Por que sincronizar demora tanto?

- Desafio é baixar gigabytes de dados?

- O verdadeiro custo está na validação
  - Cada bloco contém transações
  - Cada transação contém entradas e saídas
  - Cada entrada precisa provar que está gastando moedas legítimas
  - Cada assinatura precisa ser verificada
  - Cada UTXO precisa ser criado ou removido do estado

# Por que sincronizar demora tanto?

- Ao longo de mais de quinze anos de história
  - Processo executado bilhões de vezes.
- Baixar dados é relativamente barato
- Verificar tudo é o que custa tempo

# Por que sincronizar demora tanto?

- Como o UTXO Set muda conforme novos blocos são validados?

- Em uma rede regtest recém-criada
  - Após minerarmos os primeiros blocos

```
bitcoin-cli -datadir="." gettxoutsetinfo
{
  "height":101,
  "txouts":101,
  "total_amount":5050.00000000,
  "transactions":101
}
```

# Por que sincronizar demora tanto?

- Nesse momento o estado da rede é composto por:
  - 101 UTXOs
  - Totalizando 5050 BTC
- Agora criamos algumas transações:

```
bitcoin-cli -datadir="." sendtoaddress $(bitcoin-cli -datadir="."
getnewaddress) 1

bitcoin-cli -datadir="." sendtoaddress $(bitcoin-cli -datadir="."
getnewaddress) 1

bitcoin-cli -datadir="." sendtoaddress $(bitcoin-cli -datadir="."
getnewaddress) 1
```

# Por que sincronizar demora tanto?

- As transações entram na mempool
  - Mas ainda não alteram o estado confirmado da blockchain

- Precisamos minerar um novo bloco:

  bitcoin-cli -datadir="." generatetoaddress 1 $(bitcoin-cli -datadir="." getnewaddress)

# Por que sincronizar demora tanto?

- Após a mineração:

```
bitcoin-cli -datadir="." gettxoutsetinfo
```

- Resultado:

```
{
  "height":102,
  "txouts":104,
  "total_amount":5100.00000000,
  "transactions":103
}
```

# Por que sincronizar demora tanto?

- Observe que o estado mudou:
  - Altura da blockchain aumentou
  - Novas transações foram incorporadas
  - Quantidade de UTXOs cresceu
  - Conjunto de moedas disponíveis para gasto foi atualizado
- É esse processo que um node executa durante o IBD
- A diferença é que:
  - Em vez de processar alguns blocos e transações
  - Repetir para toda a história da rede desde 2009

# O Que Veremos!

- Introdução
- É só baixar arquivos?
- O primeiro contato com a rede
- Reconstrução do estado
- Por que sincronizar demora tanto?
- Por que reconstruir tudo desde 2009?
- Novas propostas

# Por que reconstruir tudo desde 2009?

- Se o objetivo final do node é chegar ao UTXO Set atual, por que ele precisa percorrer toda a história da rede desde o bloco gênesis?
  - Node não quer apenas obter o estado
  - Quer saber se aquele estado é consequência válida de todas as regras do Bitcoin

# Por que reconstruir tudo desde 2009?

- Um snapshot do UTXO Set pode dizer:
  - “estas são as moedas que existem agora”
  - Mas ele não prova, sozinho, que:
    - Essas moedas foram criadas corretamente
    - Nenhum gasto inválido aconteceu no caminho
    - Todas as regras de consenso foram respeitadas desde 2009

# Por que reconstruir tudo desde 2009?

- Por isso que o IBD tradicional é tão importante
- Ele não é apenas um mecanismo de sincronização
- Ele é o processo pelo qual o node constrói confiança sem precisar confiar em ninguém
- Mas essa segurança tem um custo:
  - Tempo
  - Disco
  - CPU

# O Que Veremos!

- Introdução
- É só baixar arquivos?
- O primeiro contato com a rede
- Reconstrução do estado
- Por que sincronizar demora tanto?
- Por que reconstruir tudo desde 2009?
- Novas propostas

# Novas propostas

AssumeUTXO

# Novas propostas - AssumeUTXO

- Permitir que o node comece a partir de um snapshot conhecido do UTXO Set

- Em vez de esperar toda a validação histórica terminar para só então se tornar útil

- Node pode:
  - Carregar um estado inicial
  - Começar a operar mais rapidamente
  - Continuar validando o passado em segundo plano

# Novas propostas - AssumeUTXO

- AssumeUTXO não troca validação por confiança cega
- Ele muda a ordem do processo
- No modelo tradicional:

  Validar toda a história

  ↓

  Construir o UTXO Set

  ↓

  Node fica utilizável

# Novas propostas - AssumeUTXO

- Com AssumeUTXO:
  Carregar snapshot do UTXO Set
  ↓
  Node fica utilizável mais cedo
  ↓
  Validar a história em segundo plano
  ↓
  Confirmar que o snapshot estava correto

# Novas propostas - AssumeUTXO

- Diferença sutil, mas muito importante
- Usuário ganha tempo → node começa a funcionar antes
- Software ainda preserva a ideia central do Bitcoin
  - Verificar por conta própria
- Se validação histórica não bater com o snapshot assumido:
  - Problema
- O snapshot:
  - Não é tratado como uma verdade absoluta
  - É uma hipótese inicial
  - Precisa ser confirmada

# Novas propostas

## Utreexo

# Novas propostas - Utreexo

- Se o UTXO Set não precisasse ser armazenado localmente?
- Lembrando...
- Todo node:
  - Mantém uma base contendo milhões de UTXOs

# Novas propostas - Utreexo

- Quando transações tentam gastar moedas, o node precisa:
  - Consultar essa base
  - Verificar se aquele UTXO realmente existe
  - Verificar se ainda não foi gasto

- O desafio é que:
  - Esse conjunto cresce continuamente
  - Quanto mais o Bitcoin é utilizado, mais saídas não gastas precisam ser armazenadas e consultadas

# Novas propostas - Utreexo

- A ideia do Utreexo é:
  - Substituir essa enorme base de dados
  - Um acumulador criptográfico
  - Estrutura para representar um conjunto gigantesco de elementos → Quantidade muito pequena de informação

- Uma analogia útil:
  - Árvore de Merkle
  - Não armazena todos elementos de forma explícita
  - Armazena apenas um hash raiz que representa todo o conjunto

# Novas propostas - Utreexo

Milhões de UTXOs

↓

Árvore criptográfica

↓

Hash raiz compacto

- O node passa a armazenar apenas esse acumulador

# Novas propostas - Utreexo

- Se o node não guarda os UTXOs, como ele sabe que uma moeda existe quando alguém tenta gastá-la?
  - Provas criptográficas
- Quando uma transação deseja gastar um UTXO:
  - Precisa fornecer uma prova mostrando que aquele UTXO faz parte do conjunto representado pelo acumulador
  - O node então verifica essa prova utilizando apenas o hash raiz armazenado localmente
- O trabalho que antes era feito consultando uma grande base de dados passa a ser feito verificando provas criptográficas

# Novas propostas - Utreexo

- No Bitcoin atual:
  Node
  ↓
  Armazena todos os UTXOs
  ↓
  Verifica gastos localmente
- Com Utreexo:
  Node
  ↓
  Armazena apenas o acumulador
  ↓
  Recebe provas
  ↓
  Verifica criptograficamente

# Novas propostas - Utreexo

- Benefícios:
  - Armazenamento necessário para operar um node pode cair drasticamente
  - Muito mais fácil executar nodes em dispositivos com poucos recursos

- Dificuldades:
  - UTXO Set removido localmente
    - Distribuir e atualizar continuamente essas provas

# Novas propostas

SwiftSync

# SwiftSync

- Qual é a menor quantidade de informação necessária para reconstruir o estado atual do Bitcoin?

- Quando um node executa o IBD tradicional:
  - Percorre milhões de transações
  - Quais moedas ainda existem?

- SwiftSync
  - Talvez seja possível chegar ao mesmo resultado sem reexecutar toda a história

# SwiftSync

- A ideia central gira em torno de um arquivo auxiliar chamado hints file
  - Conjunto de pistas sobre o estado final da rede
  - Em vez de armazenar
    - Blocos
    - Provas necessárias para reconstruir o UTXO Set
  - Contém apenas as informações mínimas necessárias para identificar quais elementos precisam sobreviver até o estado final

# SwiftSync

- Tradicional:
  Histórico completo da blockchain
  ↓
  Processar tudo
  ↓
  Descobrir os UTXOs sobreviventes

- SwiftSync
  Hints file
  ↓
  Identificar os sobreviventes
  ↓
  Reconstruir diretamente o estado final

# SwiftSync

- Grande parte da inspiração veio justamente das pesquisas envolvendo o Utreexo
- Os desenvolvedores perceberam:
  - Muitos UTXOs que já foram criados e posteriormente gastos, o efeito líquido sobre o estado final é nulo.
- Ou seja:
  Criar um UTXO
  ↓
  Gastar esse UTXO
  ↓
  Resultado final: nada mudou

# SwiftSync

- Se o objetivo é apenas reconstruir o estado atual
  - Precisa carregar todas informações intermediárias?

- Quanta informação pode ser descartada sem comprometer a capacidade de reconstruir e verificar o estado final?

- Números apresentados nas discussões recentes:
  - Primeiras versões do hints file → ~450 MB
  - Otimizações → ~116 MB
  - Pesquisas recentes → ~90 e 100 MB

# SwiftSync

- Ainda é uma linha de pesquisa em desenvolvimento
- Não substitui o IBD tradicional
- Ainda existem diversas questões em aberto
  - Distribuição dos arquivos
  - Validação
  - Integração com projetos como o Utreexo e o Kernel
- Quanto da história realmente precisamos processar para provar que o estado atual está correto?

# FIM

Obrigado!

Discord - Bitcoin Coders
https://discord.com/invite/e5qUsNWgQg

Rafael Penna
rapennas@gmail.com

bitcoinCoders bitups