Implementação de sistema de arquivos distribuído (baseado em Hash Tables) para aplicações multimídia

Lenin Cristi, Daniel Byoung Koo Jung, Gustavo Prado de Freitas, Carolina Riccomi, Vitor Sanches

CMCC – Universidade Federal do ABC (UFABC)
Santo André – SP – Brasil

{lenin.cristi, daniel.jung, p.gustavo, riccomi.carolina, carmignoli.v}@aluno.ufabc.edu.br

Resumo. *O presente artigo trata da viabilidade, aspectos de arquitetura e de implementação do uso de tabelas de hash distribuídas na disponibilização de conteúdo multimídia.*

Abstract. *This article addresses the viability, architectural and implementation aspects of distributed hash tables in the provision of multimedia content.*

Sumário

Objetivos

O desafio da distribuição multimídia em cenários cada vez mais complexos

Existe um desafio considerável na distribuição de conteúdo multimídia, desde entrega eficiente em plataformas variadas com consumo diversificado a redes de diferentes latências e de topologia variável.

A proliferação de dispositivos e variedade de consumo, que vai desde smartphones e tablets a TVs, a dispositivos domésticos conectados e dispositivos de realidade virtual.

Redes de topologia variável

A qualidade da conexão de rede varia significativamente em diferentes regiões, diferentes ambientes e mesmo diferentes períodos do dia para locais com grande demanda. Mesmo a rota que o dispositivo usa para recuperar conteúdo pode ter trechos e especificações alteradas durante um fluxo ou transferência de bloco de dados e é necessário otimizar e racionalizar a distribuição garantindo uma experiência de usuário satisfatória, mesmo em condições de rede desafiadoras.

A qualidade de conteúdo distribuído que aumenta sensivelmente a cada ano, na última década, o avanço nos padrões de rede, tanto cabeadas quanto sem fio, permite um padrão de uso e consumo de dados considerado impossível antes disso, considere dois cenários: Uma transmissão em tempo real em resolução 4k usando rede celular; Duas estações de trabalho conectadas a uma rede local sem fio que colaboram num mesmo projeto de edição de vídeo com dezenas de horas gravadas em 8k que precisam renderiza-lo localmente na GPU e transferir os artefatos para um NAS na mesma rede. Eram ambas tarefas inatingíveis, mas tornadas possíveis hoje por tecnologias como a 5G e a WiFi 6 respectivamente.

Tamanhos de arquivo e fluxos

A evolução dos padrões de rede e técnicas de distribuição portanto, que no limite é o que tratamos aqui, foi acompanhada por uma rápida evolução nos padrões de uso e especificação de multimídia sendo consumidas, tornando semi obsoletas tecnologias de distribuição, compressão e transmissão inovadoras em não mais que um ano.

Considere a seguinte tabela, que trata da evolução de dois dos padrões mais usados para transmissão de sinal digital:

Versão do DisplayPort	Ano	Largura de banda	Versão do HDMI	Ano	Largura de banda
DisplayPort 1.0	2008	10,8 Gb/s	HDMI 1.0 e 1.1	2004	5 Gb/s
DisplayPort 1.1 e 1.1a	2008	10,8 Gb/s	HDMI 1.2 e 1.2a	2005	5 Gb/s
DisplayPort 1.2 e 1.2a	2010	21,6 Gb/s	HDMI 1.3	2006	10 Gb/s
DisplayPort 1.3	2014	32,4 Gb/s	HDMI 1.4	2009	10 Gb/s
DisplayPort 1.4 e 1.4a	2018	32,4 Gb/s	HDMI 2.0, 2.0a e 2.0b	2016	18 Gb/s
DisplayPort 2.0	2019	80 Gb/s	HDMI 2.1, 2.1a e 2.1b	2023	48 Gb/s
DisplayPort 2.1	2022	80 Gb/s			

Larguras de banda máximas do DisplayPort e HDMI [5], adaptado de https://tecnoblog.net/responde/qual-a-diferenca-entre-hdmi-e-displayport/

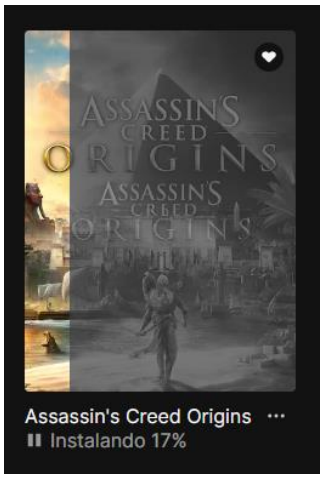
Estes números não estão distantes do consumidor final já que é comum encontrar SmartTVs com padrão HDMI 1.2 e 1.4 no varejo, e do público profissional que em estações de trabalho com placas gráficas Nvidia arquitetura Ampere (geração anterior lançada em 2020 a geração atual é a Ada Lovelace) tem DisplayPort 1.4a e suportam 4k 12-bit HDR em 240Hz ou 8k 12-bit HDR em 60Hz.

Operação e consumo distribuídos

É comum hoje o consumo de dados em redes distribuídas, mesmo que o usuário não esteja ciente disso, seguem dois exemplos de uso sutil.

Plataformas de jogo de início acelerado

É comum produtos multimídia, como jogos AAA de grandes empresas, passarem de 30GB de dados alguns chegando a 200GB no lançamento, e em plataformas de venda e distribuição desse conteúdo como a Steam e a Epic Games, existe uma estratégia que se baseia em baixar os dados referentes a texturas, funcionalidades e mapas das primeiras fases para que o usuário tenha um início rápido do jogo com o download adicional do jogo completo em segundo plano.



Assassin Creed Origins, onde é possível explorar o Egito antigo e é jogável antes dos 47GB do jogo estar baixado.

Distribuição de arquivos de atualização no Windows

As atualizações de pacotes no Windows são frequentes, a menos de uma década a Microsoft vem adotando a visão do Windows quase como uma plataforma para venda de serviços, e com updates não só de segurança, mas de funcionalidade bem mais frequentes.

A partir do Windows 10 estes updates ocorrem não somente dos servidores Microsoft, mas via P2P para computadores na rede local ou mesmo na internet se tiverem tempos de latência e taxa de transmissão melhores que dos servidores do Windows Update. Isso é possível por uma estrutura de assinaturas e hashs que garantem a integridade do arquivo transmitido.

Soluções atuais

Distributed Hash Tables

Uma Distributed Hash Table (DHT) é uma estrutura de dados descentralizada que tem seus dados organizados através de um chaveamento de cada um de seus valores. De uma forma similar às Hash Tables convencionais, seus dados são formados por pares de chave-valor: sendo que o valor seria o conteúdo de interesse nas aplicações que se aproveitam dos dados da DHT em questão, e a chave seria o parâmetro usado para localizar o valor associado à ela em uma operação de busca, no caso de uma DHT, além do endereçamento da memória em que o valor está armazenado, a chave deverá possibilitar que a operação de hashing retorne também em qual nó pertencente à rede da DHT o dado em questão está localizado.

A operação de hashing consiste em uma operação básica da Hash Table ou da DHT, que tem como entrada um valor (que pode ser usado como chave primária em

uma estrutura de banco de dados) armazenado na Hash Table, e como saída um valor que será usado para o chaveamento do dado associado. Essa operação traz a vantagem de tornar a busca por esse valor ter como complexidade igual à do algoritmo da função de hashing em casos em que é possível acessar aleatoriamente a memória. E no caso de uma DHT, o chaveamento permitirá localizar também o nó que contém esse valor.

Uma grande diferença entre uma DHT e uma Hashing Table é que a descentralização permite que as operações de consulta, inserção e remoção não interfiram no funcionamento da estrutura, os nós funcionam de forma independente, eles não dependem de uma central de controle, sendo possível ser operada com uma quantidade elevada de nós sem perder em nada na questão do desempenho

BitTorrent

Uma tecnologia que se baseia na estrutura de uma DHT é o protocolo de rede conhecido como BitTorrent, que consiste em um sistema de compartilhamento de arquivos entre diversos nós.

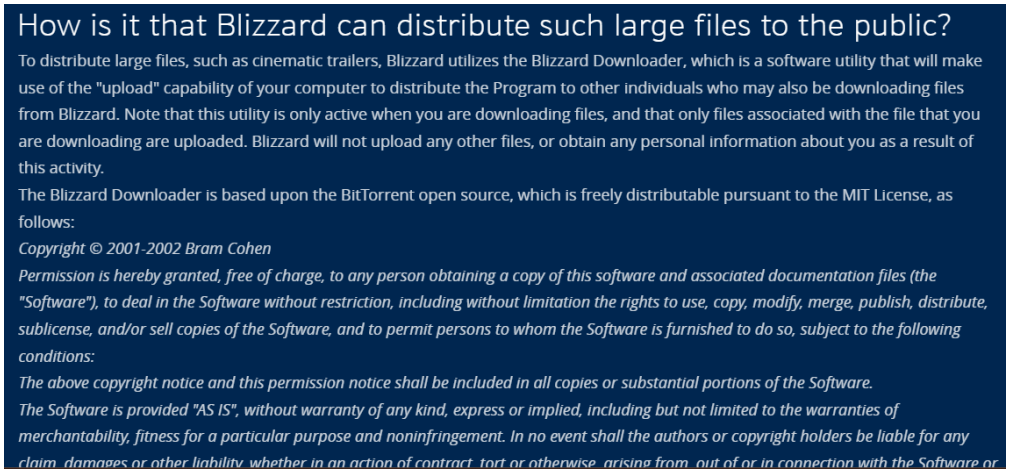
Esse sistema funciona de maneira que cada aparelho que obtém um arquivo disponível na rede, passa a participar das operações de compartilhamento desse mesmo arquivo quando estiver conectado à rede, ou seja, cada dispositivo passa a funcionar como um servidor de arquivos.

Essa característica acaba trazendo algumas vantagens no serviço, tais como a descentralização das operações, o que faz com que o sistema não dependa do funcionamento de uma central para que opere com melhor eficiência, tornando as operações menos suscetíveis à instabilidades, e faz com que a quantidade de nós crescente não sobrecarregue o servidor.

Implementações atuais

Atualizações de jogos

Como citado anteriormente, os jogos digitais são distribuídos através de downloads. Uma empresa que fornece esses jogos é a Blizzard Entertainment que utiliza um cliente BitTorrent próprio para que os clientes possam fazer o download de seus jogos (World of Warcraft , Starcraft, Diablo, Overwatch, entre outros)



Captura da tela do site oficial da Blizzard, falando sobre seu cliente de BitTorrent (disponível em: https://www.blizzard.com/pt-br/legal/28d5ebbf-c245-4408-8ba9-043dd5f056bf/legal-faq)

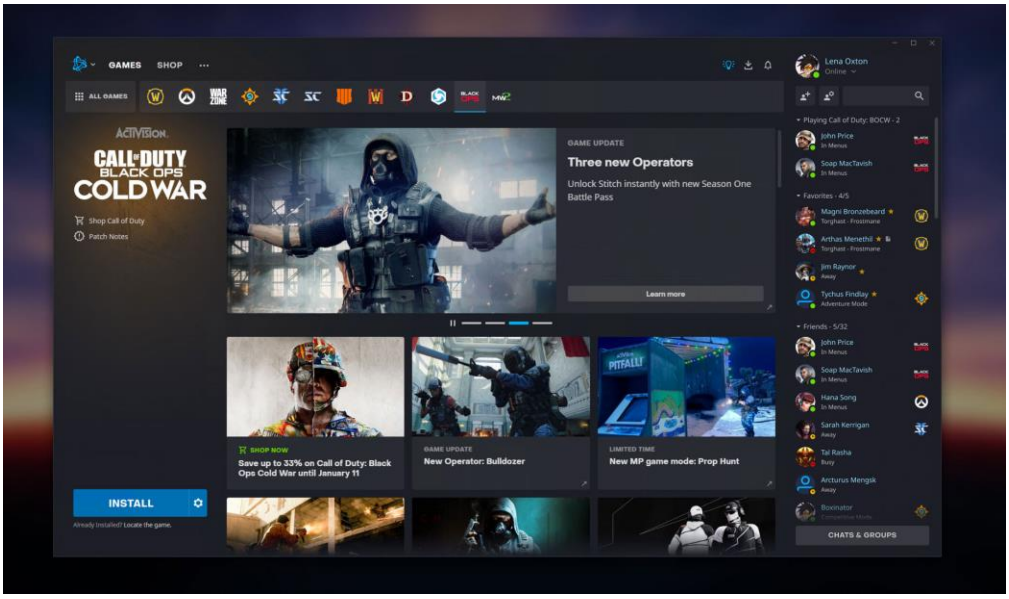


Imagem do cliente de download da Blizzard, o Battle.net (disponível em: https://www.callofduty.com/br/pt/blog/2021/02/Welcome-to-the-new-Battle-net)

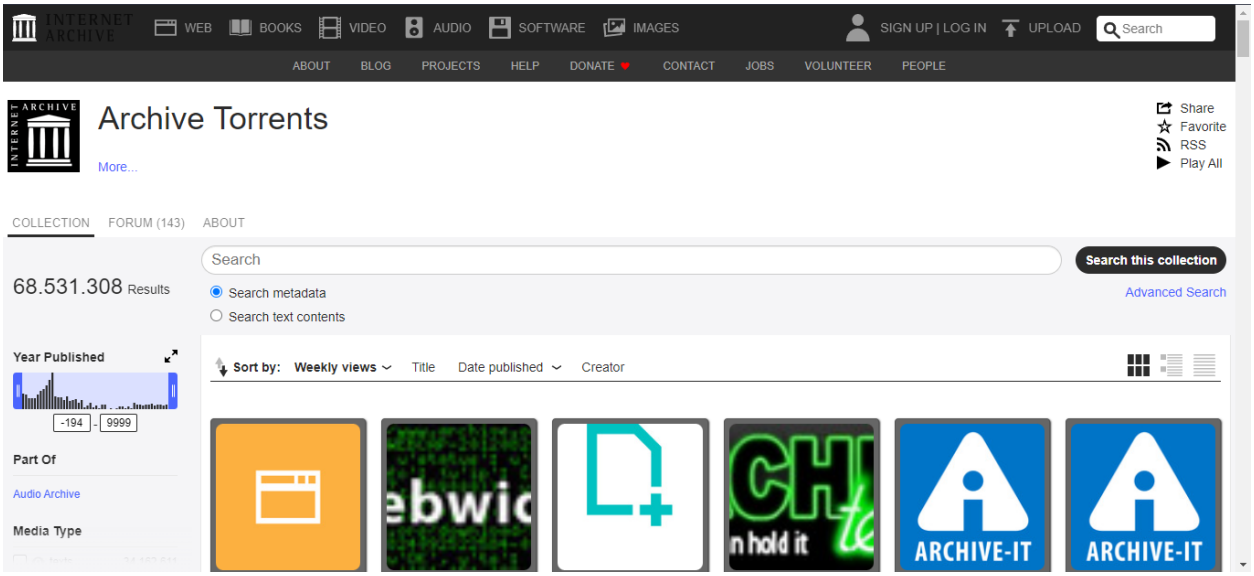
Facebook

Em 2012, o site Ars Technica fez uma reportagem sobre o facebook, mostrando detalhes do funcionamento da rede social. Em um momento, da reportagem, foi revelado que o Facebook criou um servidor de BitTorrent próprio, no qual, cada servidor do Facebook ajudará outro servidor no mesmo nó, reduzindo a latência total.

https://arstechnica.com/information-technology/2012/04/exclusive-a-behind-the-scenes-look-at-facebook-release-engineering/

Conteúdo histórico

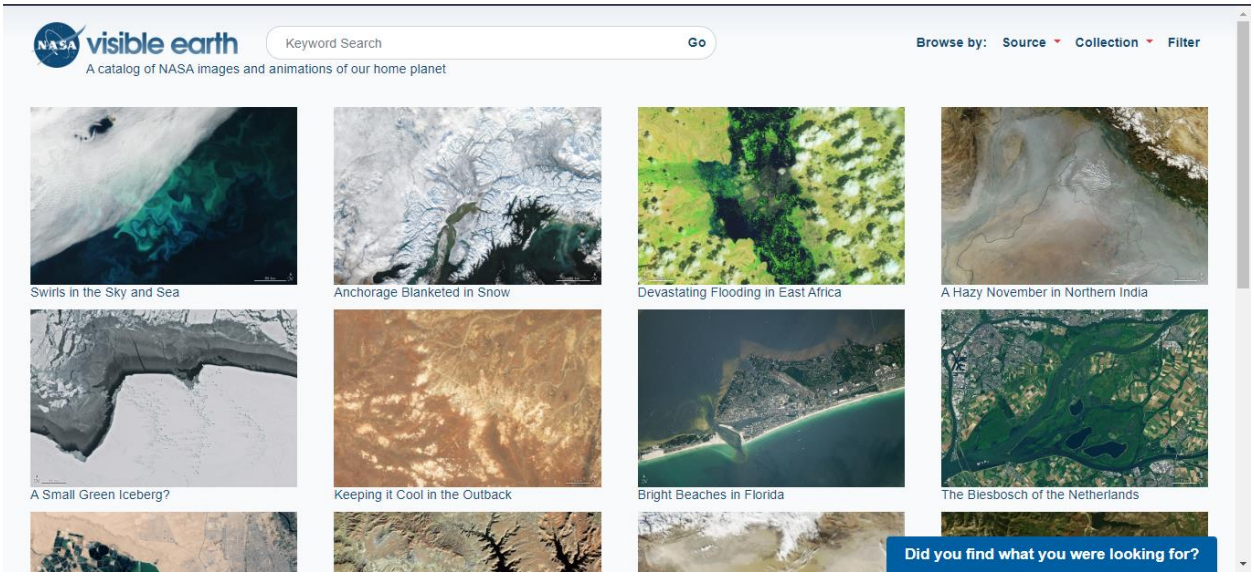
A organização Internet Archive mantém um acervo online de várias produções multimídias como páginas de internet, software, filmes, programas de TV, livros e gravações de áudio. O site possui coleções disponibilizadas em torrents e recomenda o download utilizando esse método.



Captura de tela do Archive Torrents (disponível em: https://archive.org/details/bittorrent)

NASA Visible Earth

A NASA utiliza o BitTorrent para distribuir suas imagens em alta definição do espaço.



Captura de tela do NASA Visible Earth (disponível em: https://visibleearth.nasa.gov/)

Arquitetura proposta

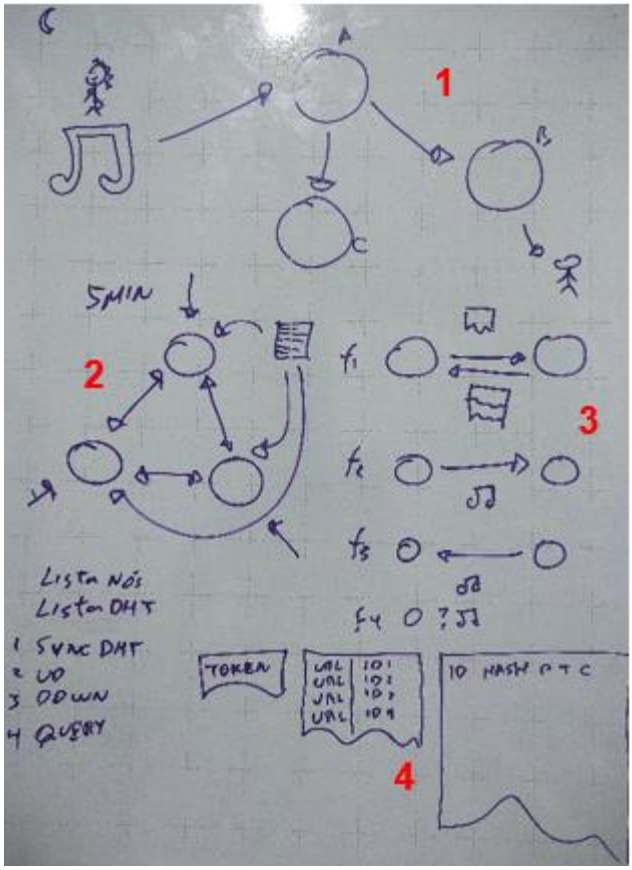
Considerações iniciais

Foram analisadas três abordagens [1][2][3] de implementação do BitTorrent e uma de DHT modificada [4], e estudadas diferentes abordagens de implementação do protocolo BitTorrent. As abordagens de implementação do protocolo BitTorrent são robustas mas com abstração alta do assunto central de DHTs, a abordagem de implementação direta de DHTs tem vantagens claras no que tange a completude técnica da solução mas também abstrai a noção mais clara de DHTs, e uma peça chave da solução é a pedagógica, uma solução abstraída ou mais formal apesar de desejável em ambientes de operação mas esconderia na apresentação da solução a estrutura de dados central no artigo, qual seja a DHT.

Optamos por uma abordagem de implementar DHTs baseadas diretamente na sua definição [7] onde é possível acompanhar as operações e deliberadamente não automatizar as alterações de nós e arquivos na rede, o que facilita sua demonstração.

Características iniciais da solução

A solução foi pensada construída em torno de DHTs que são visíveis por métodos de consulta para facilitar o entendimento de seu funcionamento, e é baseada em nós colaborativos de papel e peso idêntico na rede (ou seja, sem arquitetura master/slave) que por sua vez tem mudança fácil de topologia.



Primeiro esboço da solução

O primeiro esboço da solução foi incluído aqui para ilustrar o processo criativo da elaboração, e dá um mapa consistente das funcionalidades:

- (1) O processo de envio e consulta de arquivos na rede
- (2) O processo de sincronização colaborativa de nós e arquivos na rede
- (3) As 4 funcionalidades principais da rede
    - F1 Troca de listas de arquivos (sincronização da DHT)
    - F2 Envio de arquivo para a rede
    - F3 Recuperação de arquivo na rede
    - F4 Pesquisa na rede por um arquivo específico
- (4) As DHTs centrais da solução: nós e arquivos

Aspectos funcionais

A solução foi desenhada para permitir sincronização de dados de nós e arquivos, ser densa ou seja, permitir conexão direta entre quaisquer dois nós, ser resistente a cenários de conexão eventual entre grupos de nós (alteração de topologia), ser compatível com cenários de trechos (ou toda) na internet e usar portas padrão SSL para não ser bloqueada por firewalls de pacote.

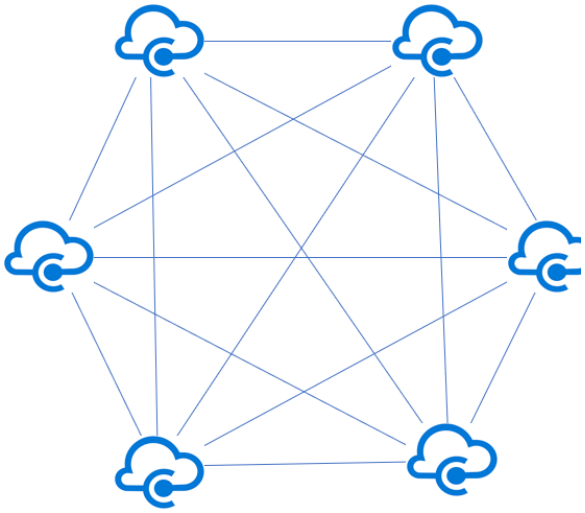


Diagrama de rede densa

Cada nó ao disparar gera sua identidade, carrega sua lista de nós conhecidos e gera sua DHT de arquivos a partir de seu diretório local.

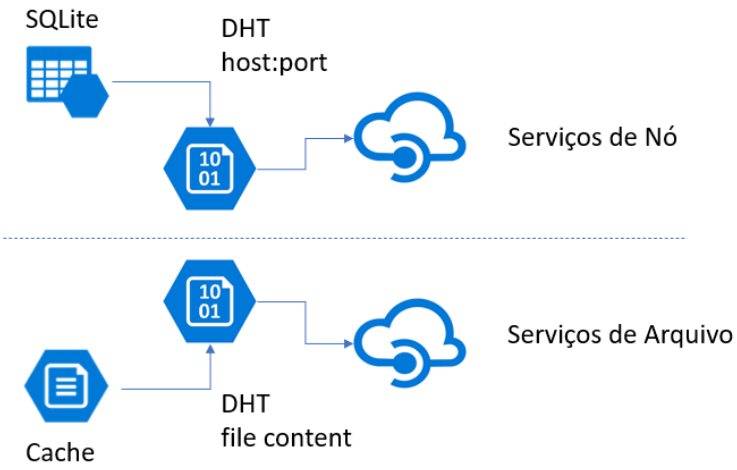


Diagrama de blocos de cada nó

A sincronização ocorre na rede entre os nós para novos nós e novos arquivos adicionados a rede, com propagação colaborativa de atualização das estruturas DHTs de nós e arquivos direta ou indiretamente. Existem duas chaves que guardam a ultima sincronização de nós e arquivos naquele nó que impede sincronização cíclica em loopback.

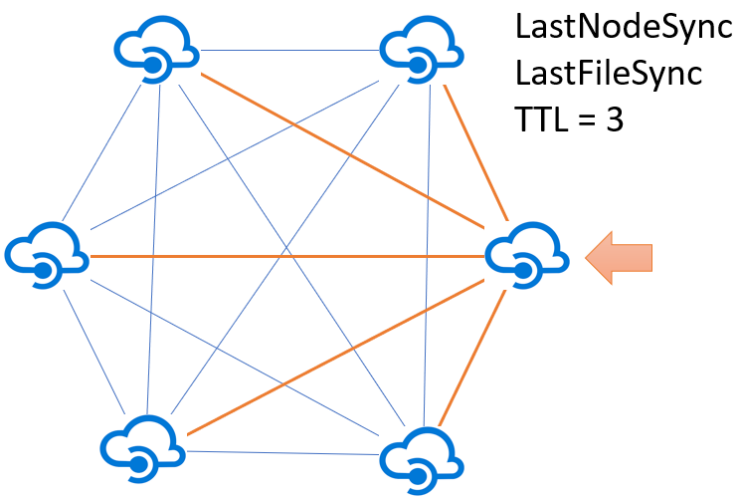


Diagrama de sincronização

A pesquisa de arquivos ocorre por difusão na rede, mas usa um cache local sincronizável da DHT de arquivos como recurso de pesquisa rápida.

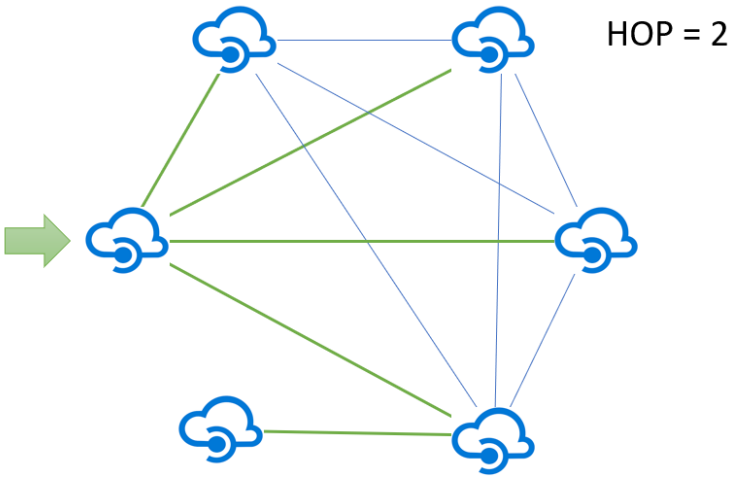


Diagrama de pesquisa na topologia variável

Implementação da solução

A implementação foi pensada para permitir múltiplas plataformas, facilidade de subida de nós, linguagem de alto nível e extensa biblioteca de apoio.

Plataformas, linguagem e framework

Foi escolhida como linguagem C# +11 no framework de código aberto .Net Core +8. Este stack de desenvolvimento permite o uso de uma ampla biblioteca pronta para uso, linguagem moderna de alto nível multiparadigma e adoção facilitada de padrões de projeto.

A solução desenvolvida suporta as mais populares distribuições Linux e Unix, MacOS e Windows e é portável também entre arquiteturas x86, x64 e ARM.

OS	Versão	Arquiteturas
Windows 10 Client	Version 1607+	x64, x86, Arm64
Windows 11	Version 22000+	x64, x86, Arm64
Windows Server	2012+	x64, x86
Alpine Linux	3.17+	x64, Arm64, Arm32
Debian	11+	x64, Arm64, Arm32
Fedora	37+	x64
openSUSE	15+	x64
Red Hat Enterprise Linux	8+	x64, Arm64
SUSE Enterprise Linux (SLES)	12 SP5+	x64
Ubuntu	20.04+	x64, Arm64, Arm32
macOS	10.15+	x64, Arm64
iOS	11.0+	Arm64
MacCatalyst	10.15+, 11.0+ no Arm64	x64, Arm64

Plataformas mais comuns do .NET Core 8 [8], para a lista completa https://github.com/dotnet/core/blob/main/release-notes/8.0/supported-os.md

Foram utilizadas fora da biblioteca padrão do .NET Core 8 as bibliotecas NewtonSoft para manipulação de estruturas JSON e RestSharp para facilitar a subida de pontos REST.

Padrões de projeto

Foram utilizados princípios SOLID no projeto como um todo, foi feita uma separação forte entre controles e serviços fazendo com que as estruturas principais da solução (as DHTs) tivessem suas operações acessíveis somente via serviços específicos e estas foram implementadas como dicionários concorrentes (thread safe) em classes Singleton que por sua vez são usadas exclusivamente via injeção de dependência.

Para a persistência de dados em banco foi usado o Entity Framework que é o MER padrão do .NET Core com o banco de dados SQLite e abordagem code-first.

Padrões de conectividade e troca de dados

Foi utilizado o padrão de APIs REST sobre HTTPS, exposto com Swagger para facilitar a apresentação e interação.

Persistência de dados

A solução é pensada para trabalhar em torno de DHTs, então deliberadamente não fizemos a persistência dos dados de arquivo em estruturas de tabela com o hash de arquivo como chave primária (ou índice separado) como seria o usual numa abordagem de persistência de dados relacionais.

Preferimos manter as estruturas em memória em formato específico de dicionários chave valor com sua chave calculada com o algoritmo de hash SHA256. Para evitar colisões, nossa estratégia, para os nós, foi gerar um hash com base no host e porta do nó e para os arquivos um hash do arquivo em si, o que em última instância pode ser usado para garantir sua integridade após a transmissão.

Não utilizamos uma tabela de persistência em banco de dados não só pelo requisito que o aplicativo operasse, gerenciasse e pesquisasse dados numa DHT, mas também pois o tempo de pesquisa dessa estrutura em memória é menor tanto para pesquisa local quanto para pesquisa na rede do que seria numa tabela persistida.

Adicionalmente a DHT de arquivos é montada no disparo da aplicação em cada nó a partir dos arquivos presentes numa pasta gerenciada pelo nó, essa recriação e o padrão de hash utilizado garante que o estado das pastas em cada nó é suficiente para remontar a DHT através da rede.

A DHT de nós também existe, mas para manter o enlace da rede entre disparos a frio do aplicativo em diferentes nós ela é persistida numa tabela em banco.

Detalhes da implementação da DHT

As tabelas DHT foram criadas utilizando a estrutura Dictionary do .NET Core 8 que utiliza hash tables internamente, possibilitando acesso direto pela chave e prevenindo colisões.

Da documentação sobre a estrutura [9]:

“A classe genérica `Dictionary<TKey,TValue>` fornece uma associação de um conjunto de chaves a um conjunto de valores. Cada adição ao dicionário consiste em um valor e sua chave associada. Recuperar um valor usando sua chave é muito rápido, próximo de O(1), porque a classe `Dictionary<TKey,TValue>` é implementada como uma tabela hash.”

Sobre limites da estrutura na adição de chaves:

“Se a contagem (`Count`) for menor que a capacidade, este método se aproxima de uma operação O(1). Se a capacidade precisar ser aumentada para acomodar o novo elemento, este método se torna uma operação O(n), onde n é a contagem (`Count`).”

Débitos técnicos

A injeção de dependência do serviço que representa o nó e, portanto, utiliza o banco de dados (diferente da que representa os arquivos) idealmente não deve ser Singleton, mas transiente. Isso aumenta significativamente a capacidade da solução de lidar com alta concorrência uma vez que este serviço é consumido em múltiplos pontos REST.

A pesquisa remota de arquivos deve suportar um número n que representa o máximo de saltos numa busca que pode ser repassada de um nó para outro, mas deve levar consigo a lista de nós onde passou e o de origem para evitar buscas cíclicas.

É necessário implementar uma autenticação que permita troca e rotação de chaves.

A rede não permite gerenciamento de redundância explícita de arquivos em nós distintos.

A pesquisa de difusão na rede pode ser paralela e processada assincronamente por nó pesquisado.

Conclusão

A implementação foi realizada utilizando DHTs e priorizado o acesso via chave valor nas estruturas em memória, estruturas estas que são sincronizadas colaborativamente através da rede de nós e utilizando os demais nós como uma espécie de memória secundária quando o atual estado da DHT em memória não tem a chave buscada.

As tabelas DHT foram criadas utilizando a estrutura Dictionary do .NET Core 8 que utiliza hash tables internamente, possibilitando acesso direto pela chave e prevenindo colisões.

Foram implementados métodos públicos de nó para sincronização, checagem de nó online, enlace de nó e visualização da estrutura e dados da DHT de nós.

Foram implementados métodos públicos de arquivo para sincronização remota e local, busca de arquivo remota e local, sincronização de DHT e visualização da estrutura e dados da DHT de arquivos.

Os métodos de determinação de chave para nó e para arquivo permitem que a tabela não tenha colisões, portanto a complexidade de tempo para busca na DHT de arquivos sincronizada é O(1), ou seja, é uma operação de tempo constante. Numa pesquisa na rede, cada nó também opera com essa mesma complexidade na sua DHT mas a pesquisa é serializada.

As operações de adição também tem complexidade O(1) e no pior caso, O(n) sendo n o numero de itens na DHT.

Referências bibliográficas

Artigos

Teo Parashkevov
[7] [Data Structures] Distributed hash table
https://medium.com/the-code-vault/data-structures-distributed-hash-table-febfd01fc0af

Luiz Monnerat, Claudio L. Amorim
[4] An effective single-hop distributed hash table with high lookup performance and low traffic overhead
https://arxiv.org/abs/1408.7070

Referências gerais

Distributed hash table
https://en.wikipedia.org/wiki/Distributed_hash_table

BitTorrent
https://en.wikipedia.org/wiki/BitTorrent

[5] DisplayPort vs HDMI: saiba qual é a melhor conexão em diferentes tipos de uso
https://tecnoblog.net/responde/qual-a-diferenca-entre-hdmi-e-displayport/

Microarquitetura Ampere
https://en.wikipedia.org/wiki/Ampere_(microarchitecture)

[8] Plataformas suportadas pelo .NET Core 8
https://github.com/dotnet/core/blob/main/release-notes/8.0/supported-os.md

[6] Comparação de especificações de GPUs voltadas ao consumidor final da Nvidia
https://www.nvidia.com/en-us/geforce/graphics-cards/compare/

[9] Dictionary<TKey,TValue> Class
https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=net-8.0&redirectedfrom=MSDN
ContainsKey
https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2.containskey?view=net-8.0&redirectedfrom=MSDN#System_Collections_Generic_Dictionary_2_ContainsKey__0_
Add
https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2.add?view=net-8.0&redirectedfrom=MSDN#System_Collections_Generic_Dictionary_2_Add__0__1_

Referências técnicas

BitTorrent Protocol Specification v1.0
https://wiki.theory.org/BitTorrentSpecification

[1] Building a BitTorrent Client
https://roadmap.sh/guides/torrent-client
https://github.com/veggiedefender/torrent-client

[2] Simple, robust, BitTorrent wire protocol implementation
https://github.com/webtorrent/bittorrent-protocol

[3] Kademlia Peer-to-Peer Distributed Hash Table Implementation in C#
https://marcclifton.wordpress.com/2017/11/10/kademlia-peer-to-peer-distributed-hash-table-implementation-in-c/
https://github.com/SyncfusionSuccinctlyE-Books/The-Kademlia-Protocol-Succinctly