Banco de dados replicados na edge, uma abordagem híbrida

Grupo 4
Ben Hur Faria Reis
Filipe Silveira Chaves
Thiago Monteles de Sousa
Felipe Aguiar Costa
João Paulo Rocha

1. Introdução

Em sistemas distribuídos, a consistência é um elemento-chave para garantir que as réplicas de dados em diferentes nós estejam sincronizadas e apresentem uma visão coerente do estado global. A replicação de bancos de dados oferece maior disponibilidade e escalabilidade, mas pode introduzir desafios relacionados à consistência dos dados. Duas abordagens principais são adotadas para lidar com esse problema: consistência forte e consistência eventual. A consistência forte assegura que todas as leituras refletem imediatamente as últimas operações de escrita, enquanto a consistência eventual permite que as réplicas se tornem consistentes em algum momento futuro. A escolha entre essas abordagens depende das necessidades específicas do sistema e dos requisitos do usuário.

Diante desse contexto, o presente projeto tem como objetivo criar um banco de dados replicado que ofereça a possibilidade de escolha entre a consistência forte e a consistência eventual, proporcionando maior flexibilidade e adaptabilidade ao sistema. Através da aplicação de princípios de sistemas distribuídos, fundamentos de arquiteturas e paradigmas de comunicação, pretende-se alcançar uma solução eficiente e confiável para a replicação de dados. Além disso, o projeto visa explorar o uso de técnicas como os *Conflict-free Replicated Data Types* (CRDT) para garantir a consistência eventual e o algoritmo Raft para assegurar a consistência forte. Dessa forma, busca-se proporcionar aos usuários a capacidade de selecionar a abordagem mais adequada às suas necessidades e ao contexto de uso do banco de dados replicado.

2. Revisão bibliográfica

Diversas técnicas de consistência são capazes de serem utilizadas aplicando conceitos de consistência forte ou eventual. Para isso, foi realizada uma busca por referências que possibilitem a fundamentação teórica desses conceitos e auxiliem na construção final do projeto.

Um dos artigos selecionados para análise é "*Conflict-free Replicated Data Types: An Overview* [1]," escrito por Nuno Preguiça em 2018. O autor explora a interseção entre sistemas distribuídos e a teoria de tipos de dados replicados livres de conflito. Neste trabalho, é introduzida a abordagem da consistência eventual, um conceito essencial em sistemas distribuídos, que permite variações temporárias nos estados entre os nós. Além disso, o artigo oferece mecanismos para conduzir esses estados divergentes a uma convergência determinística, assegurando a integridade dos dados com o tempo.

O artigo apresenta de forma abrangente os conceitos relacionados à sincronização de estados de dados por meio da consistência eventual. Ele explora várias semânticas de concorrência usando Tipos de Dados Replicados Livres de Conflito (CRDTs), desde estruturas simples como registradores até formas mais complexas, incluindo contadores, conjuntos, listas e mapas. O autor introduz essas abstrações de dados e também delineia modelos de sincronização, podendo ser baseados em estados ou operações. As ideias e conceitos explorados no artigo de Preguiça ressaltam a crescente relevância dessas técnicas em ambientes onde escalabilidade e resiliência são primordiais.

Outro artigo de destaque é "*DSON: JSON CRDT Using Delta-Mutations For Document Stores* [2]," escrito por Rik Rinberg e sua equipe (2022). Eles introduzem uma nova abordagem para resolver o problema de manter dados alinhados em sistemas de armazenamento de documentos distribuídos, como abordado no artigo anterior.

Neste contexto, o desafio está em garantir que os dados permanecem coerentes em sistemas de armazenamento de documentos distribuídos utilizando CRDTs juntamente com o formato JSON, conhecido por DSON.

O que é notável é como essa abordagem se alinha com a discussão anterior sobre a importância da coerência dos dados em sistemas distribuídos. Neste caso, os autores demonstram como os CRDTs podem ser empregados para abordar essa questão, especialmente ao empregar o formato DSON para armazenar e trocar informações. Isso fortalece a ideia de que essa metodologia é versátil e amplia nossa compreensão de como

lidar com os desafios dos sistemas distribuídos, visando assegurar a precisão e atualização constante dos dados.

Além disso, temos o trabalho "*Consistency Models of NoSQL Databases* [3]" do autor Miguel Diogo (2019), em que o autor se dedica a analisar e comparar como cinco bancos de dados NoSQL implementam o conceito de consistência, tanto no estilo eventual quanto no estilo forte. Ao fazer essas comparações, Miguel Diogo destaca os melhores métodos e abordagens para cada tipo de consistência. Ele também aponta as vantagens e desvantagens ao trabalhar com diferentes bancos de dados.

Ao olharmos desde a consistência eventual até a consistência forte, e agora incluindo a análise dos bancos de dados NoSQL, percebemos uma ligação entre esses trabalhos. Eles enriquecem a maneira como entendemos a interação complexa entre consistência de dados, sistemas distribuídos e as diferentes maneiras de enfrentar os desafios nesse cenário, mostrando mais sobre como as técnicas e modelos podem ser usados de maneira eficaz para manter os dados corretos e confiáveis em sistemas distribuídos que estão se tornando cada vez mais dinâmicos e complexos.

Outra contribuição notável é o artigo "Strongly consistent replication for a bargain [4]" por Krikellas (2022). Aqui, o foco está em analisar e comparar duas técnicas escaláveis que têm como objetivo garantir tanto segurança quanto desempenho em sistemas de banco de dados, tudo enquanto mantêm uma forma de consistência forte. Novamente, o artigo oferece insights ao apresentar diferentes métodos e abordagens para implementar essa consistência em cenários diversos.

Encerrando a revisão bibliográfica, o último artigo a ser considerado é "Consenso em Sistemas Distribuídos: RAFT vs CRDTs [5]" de Donald Hamnes (2019). Neste estudo comparativo, o autor conduz uma análise minuciosa dos dois protocolos mais proeminentes para alcançar consenso em sistemas distribuídos: o protocolo RAFT, associado à consistência forte, e os CRDTs, que trabalham com a consistência eventual.

Através dessa análise, o autor explora os fundamentos teóricos sobre esses protocolos e explora suas implementações práticas. Além disso, o estudo inclui simulações que visam avaliar a viabilidade, usabilidade e eficácia de ambas as abordagens em diferentes cenários distribuídos.

Ao abordar o tema do consenso, este artigo fecha o ciclo da revisão bibliográfica, abrangendo diferentes perspectivas sobre a manutenção da integridade dos dados em

sistemas distribuídos. Os trabalhos discutidos anteriormente trataram da consistência eventual, tipos de dados replicados, métodos para alcançar consistência forte e abordagens para enfrentar os desafios de ambientes distribuídos.

Juntos, esses estudos enriquecem nossa compreensão dos sistemas distribuídos, destacando uma variedade de soluções e abordagens que os pesquisadores têm explorado para lidar com questões críticas de sincronização e consistência. Ao examinar a trajetória percorrida por esses artigos, fica claro que o campo continua a evoluir, impulsionado pela necessidade de lidar com a crescente complexidade das aplicações distribuídas em nossa era interconectada e serve de motivação e entendimento para a ideia que buscamos implementar ao longo do trabalho.

3. Referencial Teórico

3.1 Fundamentos de Sistemas Distribuídos relacionados ao Projeto:

Nesta seção, serão descritos os princípios de sistemas distribuídos que serão aplicados no projeto, bem como os fundamentos das arquiteturas e estilos arquiteturais relevantes para a replicação do banco de dados.

Descrição dos princípios de sistemas distribuídos: Serão explorados os princípios de replicação e tolerância a falhas, fundamentais para a criação de um banco de dados replicado.

3.1.1 Descrição dos fundamentos de arquiteturas de sistemas distribuídos e dos estilos arquiteturais:

Os sistemas distribuídos são compostos por uma variedade de dispositivos interconectados que trabalham juntos para alcançar objetivos comuns. A arquitetura desses sistemas desempenha um papel fundamental na determinação de como esses dispositivos se comunicam, colaboram e gerenciam recursos compartilhados. Nesse contexto, serão explorados dois estilos arquiteturais essenciais - o modelo cliente-servidor e o modelo peer-to-peer - como estruturas fundamentais para a replicação do banco de dados proposto.

No modelo cliente-servidor, os dispositivos, conhecidos como clientes, solicitam serviços ou recursos a um servidor central. Esse servidor atua como um ponto central de autoridade, fornecendo informações ou funcionalidades conforme requisitado pelos clientes. Essa abordagem é particularmente útil quando há necessidade de controle centralizado, gerenciamento de acesso e distribuição eficiente de recursos. No contexto do projeto em questão, as chamadas de serviços externos feitas pelos nós do cluster do banco de dados seguirão o esquema cliente-servidor. Isso significa que os clientes enviaram solicitações ao servidor central, que coordena as operações e retornará as respostas apropriadas.

Por outro lado, o modelo peer-to-peer se baseia na comunicação direta entre os dispositivos. Nesse cenário, os dispositivos, conhecidos como nós, formam uma rede descentralizada, na qual cada nó pode atuar tanto como cliente quanto como servidor. Essa arquitetura promove o compartilhamento eficiente de recursos e a troca direta de informações entre os pares, eliminando a dependência de um ponto central de controle. No contexto da replicação de dados no cluster, a comunicação interna entre os nós será realizada no estilo *peer-to-peer*. Isso significa que os nós do cluster interagem diretamente entre si, permitindo a replicação e sincronização de dados de forma descentralizada, sem a necessidade de um nó central intermediário.

A escolha entre esses dois estilos arquiteturais dependerá das necessidades específicas do sistema e dos objetivos de replicação de dados. A combinação estratégica desses modelos pode resultar em uma arquitetura híbrida que aproveita as vantagens de ambos os estilos, garantindo uma replicação eficiente e confiável do banco de dados em um ambiente distribuído.

3.1.2 Descrição dos fundamentos de paradigmas de comunicação em sistemas distribuídos:

Em sistemas distribuídos, a comunicação eficaz entre dispositivos é essencial para garantir a colaboração e a troca de informações de maneira transparente. Dois paradigmas de comunicação amplamente utilizados são o *Remote Procedure Call (RPC)* e o protocolo HTTP.

O R*emote Procedure Call* (RPC) é um paradigma de comunicação que permite que processos em dispositivos diferentes se comuniquem e executem procedimentos em outros

dispositivos remotos. Esse paradigma é projetado para ocultar a complexidade da comunicação de rede, permitindo que chamadas de procedimentos sejam realizadas em dispositivos remotos da mesma maneira que seriam realizadas localmente. Isso torna a interação entre processos distribuídos mais transparente, facilitando a construção de aplicativos distribuídos sem a necessidade de lidar diretamente com detalhes de comunicação de baixo nível.

Por sua vez, o protocolo HTTP (*Hypertext Transfer Protocol*) é amplamente conhecido por facilitar a comunicação entre clientes (como navegadores da web) e servidores na Internet. Esse protocolo estabelece um modelo *request-reply*, no qual os clientes enviam solicitações (*requests*) aos servidores e recebem respostas (*replies*) contendo os dados ou recursos solicitados. No contexto do projeto, o protocolo HTTP será usado para a comunicação externa entre os clientes e os nós do cluster que hospedam o banco de dados replicado. Essa abordagem oferece uma interface padronizada para a manipulação externa dos dados armazenados, permitindo que os clientes solicitem operações de inserção, modificação, recuperação e exclusão de registros. Essas solicitações são enviadas aos nós do cluster por meio de solicitações HTTP, e as respostas contêm os resultados das operações realizadas nos dados replicados.

Em consonância com esses paradigmas, o projeto adotará uma abordagem específica para a comunicação interna e externa. A comunicação interna entre os nós do cluster, que é crucial para a replicação e sincronização de dados, será realizada utilizando o paradigma de comunicação RPC. Isso permitirá que os nós troquem informações de maneira eficiente e coordenada, garantindo a consistência dos dados replicados. Por outro lado, a comunicação externa entre os clientes e os nós do *cluster* será baseada no protocolo HTTP, com um esquema *request-reply*. Essa abordagem oferecerá aos clientes uma interface uniforme e familiar para interagir com o banco de dados replicado, independentemente de sua localização na rede distribuída.

3.2. Descrição sobre robustez em sistemas distribuídos:

Nesta parte do referencial teórico, serão apresentados conceitos relacionados à garantia de robustez no contexto de sistemas distribuídos os tipos de consistência bem como seus conceitos.

A consistência em um sistema distribuído refere-se à propriedade de que os dados armazenados em diferentes nós do sistema estejam sempre de acordo entre si.
Em uma consistência forte um nível mais rigoroso de garantia de consistência é requisitado. Isso significa que, em qualquer momento, todos os nós do sistema veem exatamente os mesmos dados, independentemente de qual nó eles estão acessando. Para alcançar a consistência forte, os sistemas geralmente utilizam mecanismos de bloqueio e sincronização rigorosos, o que pode resultar em um desempenho mais lento, especialmente em cenários de alta concorrência.

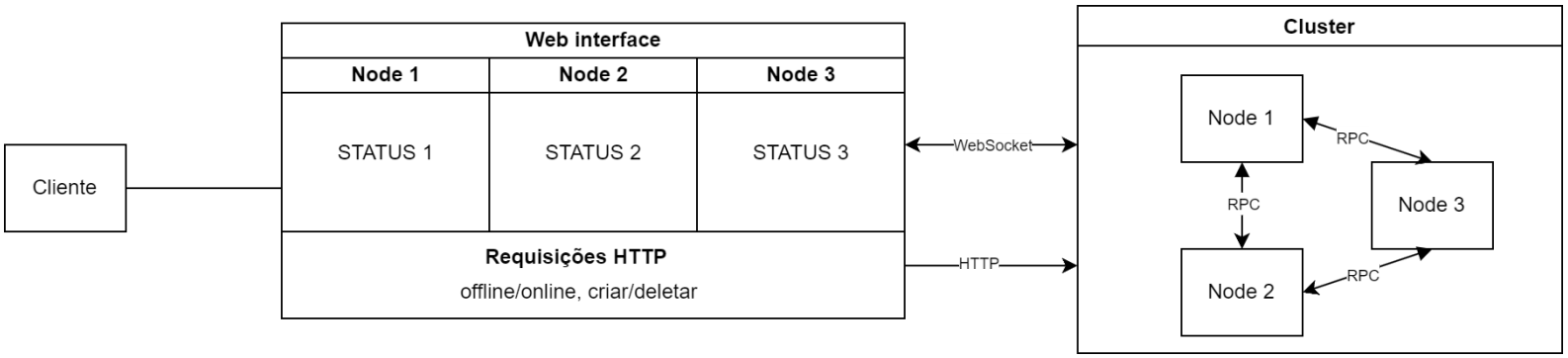
Uma das técnicas mais comuns para garantir a consistência forte é o uso de algoritmos de consenso, como o algoritmo Raft. Nesses algoritmos, os nós coordenam-se para determinar um único líder que é responsável por coordenar e aplicar todas as operações no sistema. As operações são replicadas em todos os nós e só são confirmadas após o líder garantir que a maioria dos nós as tenha processado com sucesso. Essa abordagem oferece alta consistência, mas pode ser mais suscetível a problemas de disponibilidade se o líder falhar ou se a rede entre os nós estiver instável.

Uma outra modalidade de consistência é o modelo eventual, a qual é um método menos rigoroso de consistência em sistemas distribuídos. Nesse caso, os nós podem ter visões diferentes dos dados em um determinado momento, mas, com o tempo, os dados convergem para um estado consistente em todos os nós. Isso significa que, após um período de tempo suficiente e sem novas alterações, todos os nós eventualmente terão as mesmas informações.

Para alcançar a consistência eventual, são utilizados mecanismos de replicação que permitem que os nós operam de forma assíncrona e independente. Uma abordagem comum é a utilização de *Conflict-free Replicated Data Types* (CRDTs). Os CRDTs são projetados para garantir que as operações de atualização dos dados sejam comutativas e associativas, o que permite que os nós combinem as alterações de forma independente, sem gerar conflitos. À medida que as atualizações se propagam entre os nós, eles eventualmente convergem para um estado consistente.

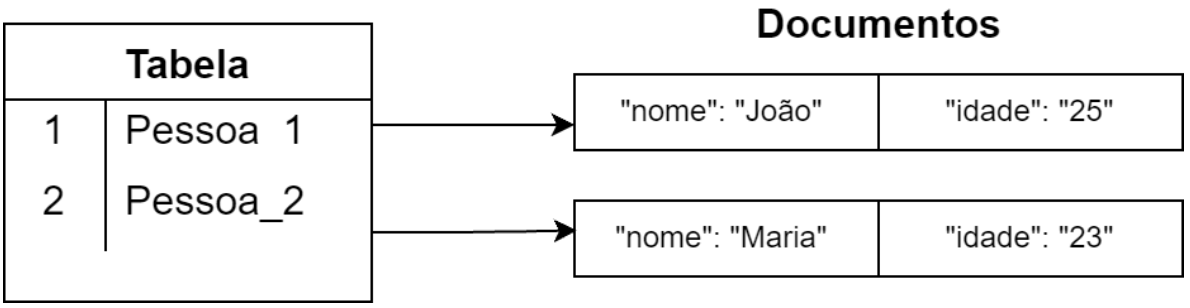
A consistência eventual é especialmente útil em sistemas que valorizam a disponibilidade e a tolerância a falhas, uma vez que os nós podem continuar a operar mesmo que a comunicação entre eles seja interrompida temporariamente. No entanto, pode haver uma janela de tempo em que os dados estão inconsistentes, o que pode ser aceitável em alguns cenários de aplicação, como em redes sociais ou sistemas de colaboração, onde pequenas discrepâncias temporárias não são críticas.

4. Metodologia



Nesta seção, será descrito como o projeto será implementado e quais etapas serão seguidas para atingir os objetivos propostos. A metodologia abrangerá a criação das tabelas no banco de dados, a especificação da consistência eventual para cada tabela, as operações de deleção, inserção, modificação e obtenção de registros nas tabelas, bem como a utilização dos princípios, arquiteturas e paradigmas de comunicação descritos no referencial teórico.

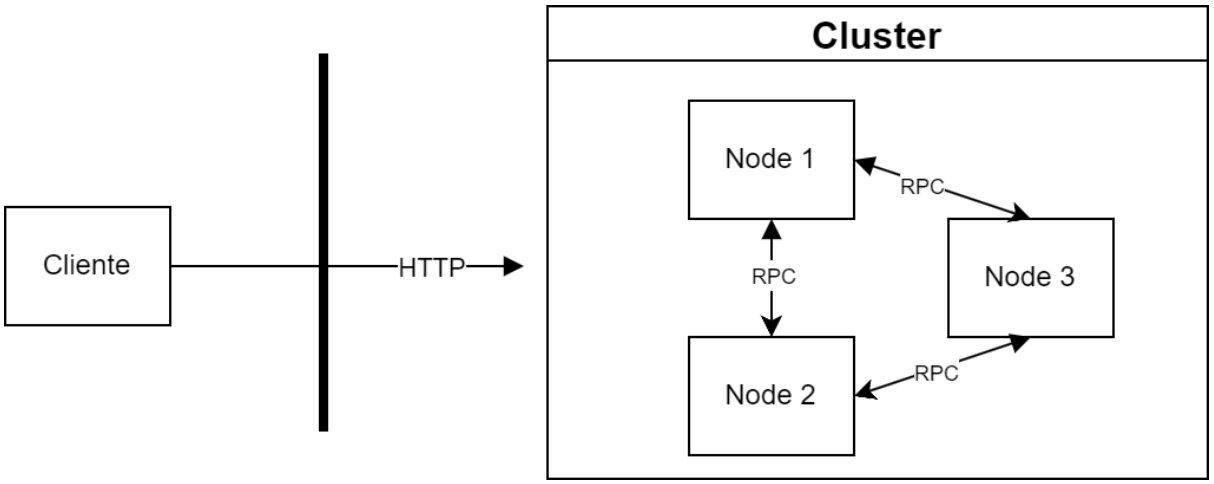
4.1. Design dos dados



Para o design dos dados será utilizado tabelas no banco de dados, onde cada tabela é uma coleção de documentos, cada documento tem um ID do tipo *string*, cada tabela pode ser

configurada para ter a consistência eventual (CRDT). Além disso, os documentos irão guarda um mapa chave-valor, as chaves e os valores *strings*, as chaves podem ser arbitrárias, não tendo um esquema fixo para os dados.

4.2. Cluster



Serão implementadas as operações de deleção, inserção, modificação e obtenção de registros nas tabelas, de acordo com a consistência eventual.
Os princípios de replicação e tolerância a falhas serão aplicados para garantir a efetividade da replicação do banco de dados.
Como demonstrado na figura acima, serão utilizados os estilos arquiteturais cliente-servidor e *peer-to-peer*, onde as chamadas de serviços externos e a comunicação interna de replicação seguirão os estilos cliente-servidor e *peer-to-peer,* respectivamente. Utilização do paradigma de comunicação RPC e protocolo HTTP, tendo a comunicação entre os nós do cluster e os clientes será feita utilizando RPC e o protocolo HTTP com esquema *request-reply*.

4.2. Mecanismo de replicação

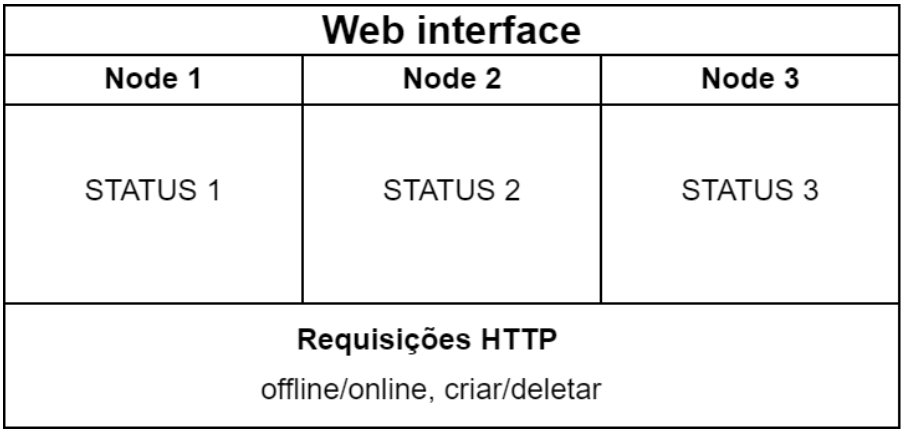
É implementada uma sincronização de CRDTs baseada em estados. Cada nó, ao receber uma operação de um cliente, vai enfileirando os novos estados dos documentos que precisam ser propagados para os outros nós. Considerando *n* nós, cada nó possui *n-1* filas de estados pendentes a serem enviados para cada outro nó. Estas filas são persistidas em disco.

Um timer é utilizado para propagar esses estados periodicamente, utilizando RPC multicast. (Essa estratégia poderia ser otimizada utilizando um protocolo gossip para tornar o multicast mais eficiente.)

Quando um nó recebe um novo estado para um documento, ele executa a operação `local_doc_state = merge(local_doc_state, new_doc_state)`. Como a operação `merge` é comutativa, todo nó sempre alcança o mesmo estado final após o recebimento dos estados remotos.

Em caso de um nó estar fora do ar e não poder receber os novos estados, os estados pendentes para este nó são enviados para as filas de todos os outros nós, para garantir maior tolerância a falhas. Assim, qualquer outro nó poderá depois enviar os estados pendentes para o nó que esteve fora do ar.

4.3. Visualizador dos nós



Foi desenvolvida uma interface web que possibilita aos usuários interagir com o sistema. Dentre as interações possíveis estão a habilidade de habilitar ou desabilitar temporizadores, sincronizar CRDTs, alternar nós entre os estados online e offline, além de criar, deletar e manipular tabelas e documentos em um nó específico.

Para monitorar o estado dos nós, é empregado um conjunto de conexões WebSocket que se comunica com os nós presentes no sistema distribuído. Essas conexões WebSocket são responsáveis por manter os estados dos CRDTs atualizados periodicamente na visualização da página web, através da respectiva conexão associada a cada nó.

O processo de realização de tais operações é viabilizado por meio de uma página web HTML contendo um código. Esse código permite as interações mencionadas anteriormente com o sistema, utilizando conexões HTTP e WebSocket.

As principais funções nessa aplicação são:

1. `sendHTTPMessage(nodeID, method, path, message)`:

   Esta função efetua requisições HTTP para interagir com o sistema DB sync. Ela constrói as requisições considerando o ID do nó, o método HTTP, o caminho e a mensagem. Dessa maneira, ela possibilita o envio de mensagens de requisição nos formatos GET, PUT, POST e DELETE.

2. `connectWebSocket(nodeID)`:

   A função é responsável por estabelecer conexões WebSocket com os nós, viabilizando a recepção de mensagens em tempo real. Adicionalmente, ela propicia a atualização imediata da interface com o estado dos nós sempre que uma mensagem é recebida.

A comunicação HTTP terá como possíveis interações:

- **`PUT /<table>`**

  Cria uma tabela

- **`DELETE /<table>`**

  Cria uma tabela

- **`GET /<table>/<doc id>`**

  Retorna o mapa de chaves e valores de um documento

- **`PUT /<table>/<doc id> (body: keys/values)`**

  Substitui o documento inteiro, apagando outras chaves não especificadas nesta chamada

- **`PATCH /<table>/<doc id> (body: keys/values)`**

  Substitui valores dentro de um documento, mantendo chaves já existentes que não foram especificadas nesta chamada

- **`DELETE /<table>/<doc id>`**

  Deleta um documento

5.Conclusão

O propósito inicial foi integrar os princípios de CRDT para assegurar a consistência eventual, ao passo que o algoritmo Raft foi adotado para garantir a consistência forte. A decisão quanto à consistência e aplicação fica sob a responsabilidade do usuário. Entretanto, a integração do algoritmo Raft enfrentou algumas complicações técnicas que não puderam ser superadas dentro do cronograma. Para implementá-lo com sucesso, seriam necessárias mais recursos temporais do que originalmente planejado para o escopo da disciplina.

Com a finalidade de cumprir o objetivo foi então, desenvolvida uma interface web para otimizar a interação do usuário com o sistema. Permitindo criar, remover e manipular tabelas e temporizadores. Além disso, essa interface possibilita a modificação do estado dos nós por meio do WebSocket, que é responsável por manter os CRDTs atualizados.

A arquitetura proposta demonstra sua adaptabilidade a diversos domínios, abrangendo conceitos essenciais de Sistemas Distribuídos, como replicação, consistência e chamadas remotas de procedimento (RPC). Essa flexibilidade torna a arquitetura viável para ambientes que requerem não apenas mecanismos de tolerância a falhas para a preservação dos dados, mas também a escalabilidade do sistema em si.

6. Referências

[1] Preguiça, N. (2018). *Conflict-free Replicated Data Types: An Overview*. http://arxiv.org/abs/1806.10254

[2] Rinberg, A., Solomon, T., Shlomo, R., Khazma, G., Lushi, G., Keidar, I., & Ta-Shma, P. (2022). DSON. *Proceedings of the VLDB Endowment*, *15*(5), 1053–1065. https://doi.org/10.14778/3510397.3510403

[3] Diogo, M., Cabral, B., & Bernardino, J. (2019). Consistency models of NoSQL databases. In *Future Internet* (Vol. 11, Issue 2). MDPI AG. https://doi.org/10.3390/fi11020043

[4] K. Krikellas, S. Elnikety, Z. Vagena and O. Hodson, "Strongly consistent replication for a bargain," 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010), Long Beach, CA, USA, 2010, pp. 52-63, doi: 10.1109/ICDE.2010.5447893.

[5] Okusanya, Oluwadamilola, "Consensus in Distributed Systems: RAFT vs CRDTs" (2019). Culminating Projects in Computer Science and Information Technology. 29. https://repository.stcloudstate.edu/csit_etds/29