Universidade Estadual do Oeste do Paraná - UNIOESTE
Centro de Engenharias e Ciências Exatas - CECE
Disciplina de Sistemas Distribuídos

Isabela Pimentel Loebel
Ítalo Gavassi
Nickolas Crema

Banco de Dados Distribuídos - Trabalho 6

Foz do Iguaçu
2024

Sumário

1. Introdução

Özsu (1991) define Banco de Dados Distribuído (BDD) como um sistema de gerenciamento de dados no qual os dados são armazenados em múltiplos computadores interconectados, em vez de um único servidor. Essa distribuição física proporciona uma série de benefícios, como escalabilidade, confiabilidade e disponibilidade aprimoradas.

Ao conceber um banco de dados, o processo compreende a elaboração do esquema global no nível conceitual, utilizando frequentemente o Modelo Entidade-Relacionamento (ME-R). Posteriormente, esse esquema é traduzido para o nível lógico, geralmente por meio da linguagem do modelo relacional, e os métodos de armazenamento físico são especificados, conforme ilustrado pelas três fases de projeto descritas por Mesquita (2009), na figura 1.

Figura 1: Fases de Projeto de um Banco de Dados. Fonte: [MESQUITA, 2009].

No contexto de um projeto de banco de dados distribuído, embora o processo seja semelhante, a distribuição ocorre no nível lógico, antes da definição das estratégias no nível físico. Essa distribuição é realizada por meio de fragmentações aplicadas ao esquema lógico do banco de dados.

Conforme a afirmação de Mesquita (2009) a fragmentação do banco de dados ocorre no nível lógico, a partir do esquema lógico original, resultando em fragmentos que compõem o novo esquema distribuído. Após a geração do esquema lógico distribuído, e principalmente após sua fragmentação, o esquema conceitual é abandonado, servindo apenas como documentação inicial da estrutura centralizada do banco de dados, nota-se que também é evidenciada a diferença entre centralização e integração. A integração de dados combina dados de diferentes fontes em um único sistema, enquanto a centralização de dados armazena todos os dados em um único local.

Autores como Ceri, Pernici e Wiederhold (1987), associaram o esquema de fragmentação no modelo relacional à dependência das consultas realizadas nos aplicativos de banco de dados. Isso sugere que a fragmentação do banco de dados é acidental e dependente das aplicações existentes.

Wildeemberg (2003) destaca que o desafio da alocação de dados consiste em encontrar a distribuição mais eficiente dos dados entre os nós da rede, visando minimizar o custo das operações envolvendo esses dados. No contexto de um SBDD, a alocação desempenha um papel crítico, uma vez que uma distribuição ineficiente dos dados pode resultar em um aumento significativo no custo de acesso ao SBDD.

Geralmente, durante um projeto de distribuição de banco de dados, a fase de alocação ocorre após a fase de fragmentação, que determina como os dados são divididos em fragmentos. Um dos principais objetivos durante a fase de alocação é aumentar a proximidade entre os fragmentos de dados e os nós que os acessam, o que ajuda a minimizar o custo de comunicação, e também afeta outros parâmetros envolvidos no escopo de um banco de dados distribuído.

2. Caracterização

Mattoso (2009) define um Sistema de Banco de Dados Distribuídos (SBDD) como a integração de dois conceitos fundamentais. Em primeiro lugar, ele compreende a noção de Base de Dados Distribuída, que constitui uma coleção de bases de dados distintas, logicamente interligadas através de uma rede de computadores. Em segundo lugar, incorpora o conceito de Sistema de Base de Dados Distribuídas, que se refere ao conjunto de software responsável pela administração da base de dados distribuída, assegurando a transparência da distribuição para o usuário. Dessa forma, a união das bases de dados com o sistema de gestão resulta em um Sistema de Banco de Dados Distribuídos.

Podemos considerar que as principais características de um banco de dados distribuído incluem a distribuição de dados, a transparência de distribuição, a autonomia local, a transparência de replicação, a transparência de concorrência, a escalabilidade horizontal, a tolerância a falhas, bem como os desafios de consistência e coerência.

Essas características tornam os bancos de dados distribuídos essenciais para ambientes onde se requer escalabilidade, alta disponibilidade, tolerância a falhas e distribuição geográfica dos dados. Entretanto, também introduzem complexidades adicionais no projeto, desenvolvimento e manutenção do sistema de banco de dados.

2.1. Arquitetura

A arquitetura de um banco de dados distribuído (BDD) refere-se à estrutura e organização dos diferentes componentes que compõem o sistema distribuído, projetada para garantir eficiência, escalabilidade, disponibilidade e tolerância a falhas. Dentre os elementos comuns em uma arquitetura de banco de dados distribuído, destacam-se:

- **Nós:** Representam os dispositivos individuais (servidores, máquinas virtuais, etc.) que compõem o sistema distribuído, cada um com sua própria capacidade de armazenamento e processamento;

- **Rede:** Consiste no meio de comunicação que conecta os diversos nós do banco de dados distribuído, sendo uma rede confiável e de alta velocidade essencial para uma comunicação eficiente entre os nós;
- **Gerenciador de Dados Distribuídos:** Responsável por coordenar as operações de acesso aos dados distribuídos, gerenciando a distribuição dos dados entre os nós, roteando consultas e atualizações para os nós apropriados e garantindo a consistência dos dados;
- **Sistema de Gerenciamento de Banco de Dados (SGBD):** Cada nó do banco de dados distribuído pode executar um SGBD local para gerenciar os dados armazenados nele, fornecendo uma interface para os aplicativos acessarem e manipularem os dados, além de executar operações de gerenciamento de dados;
- **Distribuição de Dados:** Os dados são distribuídos entre os diferentes nós do banco de dados distribuído para melhorar o desempenho e a escalabilidade, podendo ser realizada por meio de partições horizontais ou verticais;
- **Replicação de Dados:** Os dados podem ser replicados em vários nós do banco de dados distribuído para melhorar a disponibilidade e a tolerância a falhas, podendo ser síncrona ou assíncrona, com diferentes políticas de consistência aplicadas;
- **Balanceamento de Carga:** Importante para garantir a utilização eficiente dos recursos de armazenamento e processamento em todos os nós do banco de dados distribuído, envolvendo a distribuição equitativa das consultas e atualizações entre os nós, ou a movimentação dinâmica dos dados;
- **Tolerância a Falhas:** A arquitetura deve incluir mecanismos para detectar e se recuperar de falhas nos nós individuais ou na rede, envolvendo a detecção automática de falhas, replicação de dados e reconfiguração dinâmica do sistema.

Em contraste com os bancos de dados centralizados, nos quais a massa de dados fica em um único local, no BDD os dados ficam distribuídos em diversos servidores, podendo estar fisicamente próximos ou geograficamente distantes. Para os usuários que utilizam aplicações nesse ambiente distribuído, a arquitetura de BDD é transparente, apesar de os dados estarem distribuídos em vários locais fisicamente separados.

Na implementação desse ambiente distribuído, manter a consistência dos dados é um dos desafios mais importantes e complexos. Para garantir essa consistência, o BDD preserva as propriedades ACID:

- **Atomicidade:** Todas as ações de uma transação devem ser concluídas com sucesso (*Commit*) ou revertidas (*Rollback*) na totalidade;
- **Consistência:** A execução de uma transação isolada preserva a consistência do banco de dados;
- **Isolamento:** Cada transação não toma conhecimento das transações concorrentes;
- **Durabilidade:** As mudanças feitas por uma transação persistem no banco de dados após o seu término bem-sucedido (*Commit*).

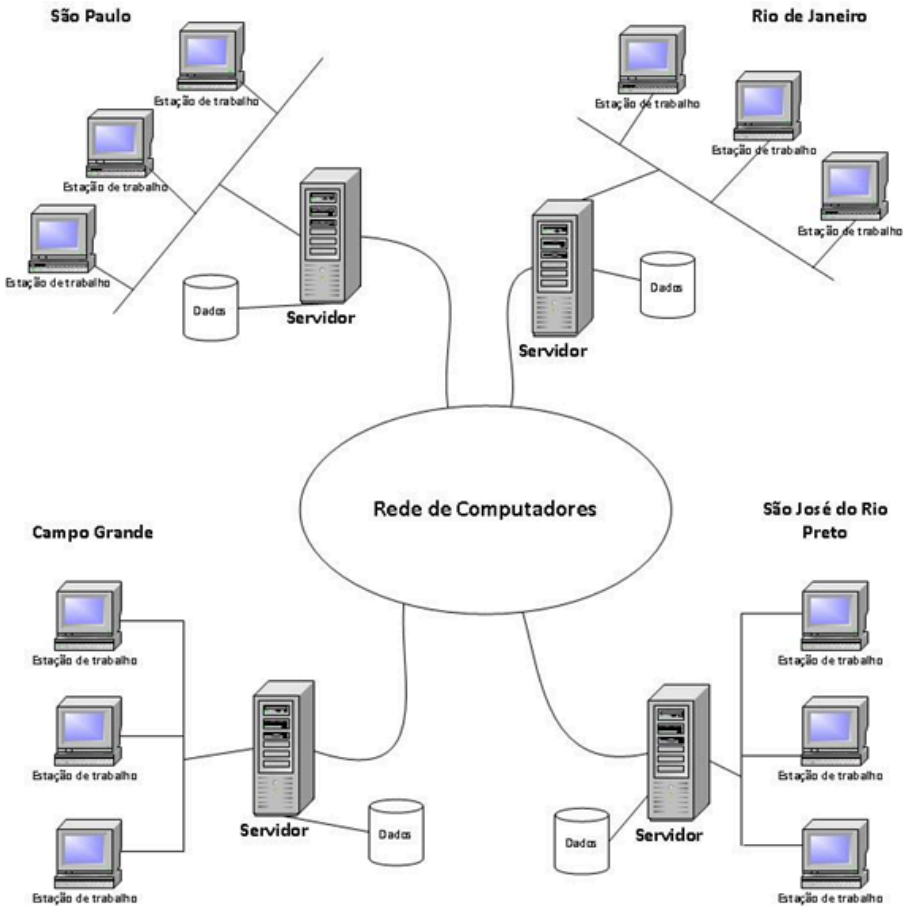

Figura 2: Arquitetura de Banco de Dados Distribuída do exemplo. Fonte: [ROSÁRIO, 2013].

No contexto do exemplo fornecido por Rosário (2013), um banco de dados distribuído pode ser imaginado em uma instituição financeira distribuída em três grandes cidades brasileiras: São Paulo, Brasília e Rio de Janeiro. Nesse cenário, transações distribuídas entre esses servidores são executadas continuamente para armazenar informações e movimentações financeiras dos clientes, tendo a arquitetura disposta na figura 2, enquanto a visão conceitual é apresentada na figura 3.

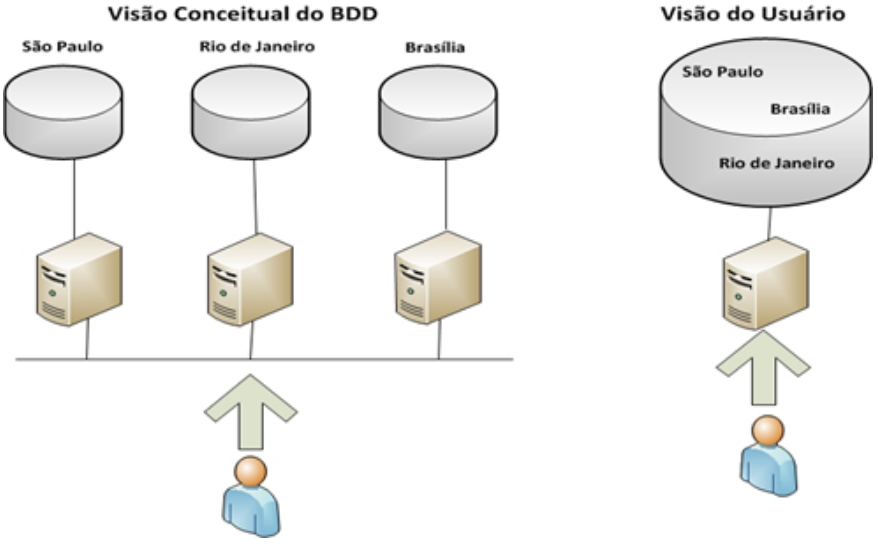

Figura 3: Visão conceitual do Banco de Dados do exemplo. Fonte: [ROSÁRIO, 2013].

Suponha que o cliente José realize um saque de R$ 1.000,00 de sua conta-corrente. Para garantir a consistência dos dados, essa operação deve ser replicada em todos os servidores envolvidos na transação. Caso contrário, o saldo de José ficará inconsistente, resultando em informações discrepantes entre os servidores.

Essa falta de consistência pode acarretar sérios problemas, uma vez que os usuários acessam os dados nos servidores com base na sua localidade, buscando reduzir a latência e obter respostas mais rápidas. Por exemplo, se José estiver acessando sua conta em São Paulo, espera-se que o sistema recupere os dados do servidor de São Paulo. No entanto, se a informação sobre o saque de R$ 1.000,00 não for replicada para o servidor de Brasília, José terá um saldo diferente em cada localidade.

Esse exemplo ilustra um dos desafios enfrentados em um ambiente de banco de dados distribuído: garantir a consistência dos dados. Outro exemplo seria o controle de estoque de um produto que foi vendido, mas a baixa não foi registrada em todos os servidores envolvidos na transação, resultando em informações imprecisas sobre a quantidade disponível desse produto. É importante ressaltar que esse cenário é simplificado e serve apenas para demonstrar a importância da consistência dos dados em um ambiente distribuído.

2.2. Vantagens

Em sua obra, Ozsu (1999) ressalta as vantagens proporcionadas pelos sistemas de banco de dados distribuídos, especialmente em relação à:

2.2.1. Transparência na Gerência dos Dados Distribuídos e Replicados

Essa transparência abrange diferentes aspectos, incluindo a localização física dos dados, onde os usuários podem acessar e manipular informações sem necessariamente conhecer sua localização exata na rede. Além disso, a replicação dos dados em vários nós do sistema aumenta a disponibilidade e a confiabilidade das informações, mesmo diante de possíveis falhas em um dos nós.

Outro aspecto relevante é a transparência da fragmentação, que permite aos usuários interagir com partes lógicas dos dados sem precisar entender a complexa divisão física, resultando em uma visão unificada e integrada da base de dados, independentemente de sua distribuição física. Essa abordagem transparente na gestão dos dados contribui para uma experiência de usuário mais consistente e eficiente em ambientes distribuídos.

2.2.2. Confiabilidade através de Transações Distribuídas

Os SBDD podem proporcionar um alto nível de confiabilidade através da implementação de transações distribuídas. Nesse cenário, as operações são realizadas coordenadamente em múltiplos nós do sistema. Essa abordagem visa reduzir os pontos únicos de falha, pois mesmo que ocorra uma falha em um nó específico, a integridade dos dados não é comprometida. Isso se deve ao fato de que outras réplicas dos dados podem ser acessadas para assegurar a continuidade das operações. Dessa forma, a utilização de transações distribuídas aumenta a resiliência do sistema, garantindo que mesmo em situações de falha, a consistência e disponibilidade dos dados sejam mantidas de maneira eficaz.

2.2.3. Aumento de Desempenho

A distribuição dos dados entre servidores distintos é uma prática comum nos SBDD e traz consigo uma série de benefícios, incluindo a redução da latência e do tempo de resposta. Ao distribuir os dados entre os servidores, torna-se possível acessá-los no nó mais próximo do usuário ou da aplicação, minimizando a necessidade de acesso remoto. Isso contribui significativamente para a melhoria do desempenho do sistema, uma vez que a latência associada ao acesso remoto é mitigada, evitando possíveis atrasos e lentidões nas operações de consulta e manipulação de dados.

Outra estratégia fundamental para otimizar o desempenho nos SBDD é o emprego do paralelismo no processamento de consultas. Por meio dessa técnica, consultas complexas são divididas em tarefas menores, as quais são executadas simultaneamente em diferentes nós do sistema. Dessa forma, os recursos computacionais disponíveis são melhor aproveitados, possibilitando um processamento mais rápido e eficiente das consultas. O paralelismo no processamento de consultas é essencial para lidar com cargas de trabalho intensivas e consultas complexas, garantindo que o sistema consiga responder de maneira ágil e eficaz às demandas dos usuários.

2.2.4. Facilidade de Expansão

A escalabilidade horizontal é uma característica fundamental dos SBDD, possibilitando a adição de novos nós de armazenamento e processamento conforme a demanda aumenta. Essa capacidade oferece uma vantagem significativa, uma vez que permite que a infraestrutura do banco de dados cresça de forma flexível, acompanhando o crescimento do volume de dados e do número de usuários. A adição de novos nós ao sistema distribuído não apenas aumenta a capacidade de armazenamento e processamento, mas também garante que o desempenho e a disponibilidade do sistema sejam mantidos.

Dessa forma, os SBDD conseguem lidar eficazmente com o aumento da carga de trabalho, sem comprometer a qualidade do serviço oferecido aos usuários. Essa flexibilidade e capacidade de expansão tornam os SBDD uma escolha ideal para ambientes dinâmicos e em constante evolução, onde a escalabilidade é essencial para garantir a eficiência operacional e a satisfação do usuário.

2.3. Projeto de Bases de Dados Distribuídas

O projeto de bases de dados distribuídas envolve uma série de atividades para garantir que os dados sejam armazenados, acessados e manipulados eficientemente em um ambiente distribuído. Duas abordagens comuns citadas por Mattoso (2009) para o projeto de bases de dados distribuídas são o Projeto Ascendente e o Projeto Descendente.

2.3.1. Projeto Ascendente

O Projeto Ascendente visa integrar bases de dados preexistentes em um ambiente distribuído, permitindo o acesso e manipulação unificada dos dados. Este projeto assume um papel crucial na construção de sistemas distribuídos eficientes e escaláveis. Com sua abordagem abrangente e detalhada, fornece um roteiro sólido para a integração eficaz de bases de dados em um ambiente distribuído. Através da implementação meticulosa das etapas

descritas, este projeto pode garantir o acesso unificado e eficiente aos dados, impulsionando o sucesso de sistemas distribuídos complexos.

Etapas Detalhadas do Projeto Ascendente:

- Identificação Detalhada de Bases de Dados:
    - Mapeamento Abrangente: Identificar todas as bases de dados relevantes, incluindo bancos de dados relacionais, NoSQL, XML e outros formatos;
    - Análise Profunda de Propriedades: Registrar informações sobre cada base de dados, como tipo, tamanho, estrutura, frequência de atualização, localização física, políticas de acesso e segurança, e proprietários;
    - Avaliação Crítica de Relevância: Analisar a relevância de cada base de dados para o sistema distribuído, considerando sua utilidade, qualidade e confiabilidade dos dados.
- Análise Aprofundada de Esquemas:
    - Engenharia Reversa Detalhada: Extrair os esquemas de dados de cada base de dados, incluindo tabelas, colunas, tipos de dados, chaves primárias e estrangeiras, relacionamentos;
    - Análise Semântica Rigorosa: Descobrir a semântica dos dados em cada base de dados, incluindo o significado de cada coluna, tabela e relacionamento;
    - Normalização Avançada: Normalizar os esquemas de dados para eliminar redundâncias e inconsistências, garantindo a qualidade e confiabilidade dos dados.
- Mapeamento Completo de Esquemas:
    - Integração Conceitual: Criar um modelo de dados conceitual unificado que represente a visão geral de todos os dados a serem integrados;
    - Mapeamento Estruturado: Mapear cada elemento dos esquemas de dados das bases existentes para o modelo de dados conceitual unificado;
    - Resolução de Conflitos Complexos: Resolver conflitos de nomenclatura, tipos de dados e semântica entre os diferentes esquemas de dados;
    - Transformação de Esquemas: Transformar os esquemas de dados das bases existentes para o formato do modelo de dados unificado.
- Desenvolvimento de Mecanismos de Integração Robustez:
    - Extração Eficiente de Dados: Desenvolver mecanismos para extrair dados das bases existentes eficientemente, considerando diferentes formatos de dados, APIs e protocolos;
    - Transformação Flexível de Dados: Implementar mecanismos para transformar os dados extraídos para o formato do modelo dedados unificado, aplicando regras de negócios e validação de dados;

    - Carga Incremental Otimizada: Carregar os dados transformados no sistema distribuído incrementalmente, minimizando o impacto no desempenho e na disponibilidade do sistema;
    - Sincronização Contínua de Dados: Estabelecer mecanismos para sincronizar continuamente os dados entre as bases existentes e o sistema distribuído, garantindo a consistência dos dados em tempo real.
- Implementação de Camadas de Abstração Versáteis:
    - Camada de Acesso a Dados Unificada: Criar uma camada de abstração que fornece uma interface unificada para acessar e manipular os dados de todas as bases de dados, independentemente de suas características específicas;
    - Camada de Tradução de Consultas: Implementar uma camada para traduzir consultas de linguagem natural para a linguagem específica de cada base de dados, permitindo que os usuários utilizem uma linguagem única para acessar os dados;
    - Camada de Gerenciamento de Transações: Desenvolver uma camada para gerenciar transações distribuídas que abrangem várias bases de dados, garantindo a consistência e a integridade dos dados.
- Considerações Adicionais:
    - Segurança Robusta: Implementar medidas de segurança robustas para proteger os dados contra acesso não autorizado, uso indevido e violações;
    - Gerenciamento de Desempenho Eficiente: Monitorar e otimizar o desempenho do sistema de integração para garantir acesso rápido e eficiente aos dados;
    - Evolução Contínua: Planejar e implementar mecanismos para a evolução contínua do sistema de integração, adaptando-se às mudanças nas bases de dados e nas necessidades do sistema distribuído.

2.3.2. Projeto Descendente

O Projeto Descendente complementa o Projeto Ascendente ao concentrar-se na distribuição estratégica das entidades globais, como tabelas ou entidades do banco de dados global, num sistema distribuído. Este permite distribuir as entidades globais em um sistema distribuído de maneira otimizada. Através da fragmentação e alocação cuidadosas, este projeto contribui para o desempenho aprimorado, escalabilidade robusta e alta disponibilidade do sistema na totalidade.

Essa abordagem visa fragmentar e alocar os recursos de forma otimizada, com os seguintes objetivos:

- Aprimorar o desempenho: Reduzir o tempo de acesso aos dados, minimizando a latência e otimizando a carga de trabalho. Isso é alcançado através de uma distribuição cuidadosamente planejada das entidades globais entre os diferentes nós do sistema distribuído, garantindo uma recuperação rápida e eficiente dos dados quando necessário;

- Aumentar a escalabilidade: Possibilitar que o sistema se expanda facilmente, adicionando mais nós conforme necessário, sem comprometer o desempenho. A distribuição estratégica das entidades globais permite uma distribuição equitativa da carga de trabalho entre os diferentes nós, garantindo que o sistema possa lidar com um aumento significativo na demanda sem perder eficiência.
- Aprimorar a disponibilidade: Garantir o acesso contínuo aos dados, mesmo em situações de falha de nó. Isso é alcançado através da replicação e distribuição redundante dos dados em vários nós do sistema distribuído, garantindo que as informações permaneçam acessíveis mesmo em caso de interrupções ou falhas em um, ou mais nós do sistema.

Etapas Detalhadas do Projeto Descendente:

- Fragmentação Minuciosa:
    - Análise Profunda de Entidades: Avaliar as características de cada entidade global, como tamanho, frequência de acesso, padrões de consulta e granularidade de dados;
    - Seleção Precisa de Técnica de Fragmentação: Escolher a técnica de fragmentação mais adequada para cada entidade, considerando as características da entidade e os objetivos de distribuição.
    - Implementação Eficaz de Fragmentação: Fragmentar as entidades globais em fragmentos menores, utilizando técnicas como:
        - Fragmentação Horizontal: Dividir as linhas de uma tabela com base em critérios específicos, como intervalos de valores de uma coluna;
        - Fragmentação Vertical: Dividir as colunas de uma tabela em fragmentos, onde cada fragmento contém um subconjunto das colunas da tabela original;
        - Fragmentação Híbrida: Combinar fragmentação horizontal e vertical para criar fragmentos mais adequados aos requisitos de consulta e acesso aos dados;
- Alocação Otimizada de Fragmentos:
    - Definição de Critérios de Alocação: Estabelecer critérios para determinar a melhor localização para cada fragmento, como:
        - Localização Geográfica: Distribuir fragmentos com base na localização física dos dados para minimizar a latência de acesso;
        - Carga de Trabalho: Distribuir fragmentos para equilibrar a carga de trabalho entre os nós do sistema distribuído;
        - Replicação: Decidir se os fragmentos serão replicados em vários nós para melhorar a disponibilidade e a tolerância a falhas.

    - Utilização de Algoritmos de Alocação: Implementar algoritmos eficientes para alocar os fragmentos nos nós do sistema, considerando os critérios de alocação definidos;
    - Monitoramento e Ajustamento Contínuo: Monitorar a performance do sistema após a alocação e realizar ajustes quando necessário, otimizando a distribuição dos fragmentos.
- Considerações Adicionais:
    - Seleção de Ferramentas Apropriadas: Utilizar ferramentas e frameworks disponíveis para auxiliar na fragmentação e alocação de entidades, como Apache Hadoop, Apache Hive, Apache HBase, Cassandra, etc;
    - Gerenciamento de Metadados Eficaz: Armazenar e gerenciar metadados sobre os fragmentos, como localização, replicação e propriedades de acesso, para facilitar o gerenciamento do sistema distribuído;
    - Manutenção da Consistência: Implementar mecanismos para garantir a consistência dos dados fragmentados, mesmo em caso de atualizações simultâneas.

2.4. Processamento Distribuído de Consultas

O processamento de consultas em um SBDD apresenta desafios únicos em comparação com os sistemas centralizados. As consultas em um SBDD precisam ser particionadas, executadas em paralelo em diferentes nós e os resultados precisam ser combinados eficientemente.

Abordagens para Processamento de Consultas Distribuídas:

- Fragmentação de Consultas
    - A consulta é dividida em subconsultas menores que podem ser executadas em paralelo em diferentes nós;
    - Algoritmo de Fragmentação:
        - Fragmentação por Hash: Distribui os dados uniformemente nos nós com base em um valor de hash;
        - Fragmentação por Intervalo: Divide os dados em intervalos de valores e aloca cada intervalo em um nó específico;
        - Fragmentação por Lista: Divide os dados em listas e aloca cada lista em um nó específico.
- Execução de Consultas Distribuídas:
    - Planejamento de Execução:
        - Seleção de Operadores: Escolher os operadores de consulta mais eficientes para cada subconsulta;
        - Estimativa de Custo: Estimar o custo de execução de cada subconsulta em diferentes nós;

        - Geração de Plano: Criar um plano de execução que especifica a ordem de execução das subconsultas e como os resultados serão combinados.
    - Execução Paralela:
        - Execução Assíncrona: As subconsultas são executadas em paralelo em diferentes nós sem esperar que uma termine antes da outra;
        - Execução Síncrona: As subconsultas são executadas em paralelo em diferentes nós, mas a execução de uma subconsulta pode esperar pela finalização de outra.
    - Combinação de Resultados:
        - Combinação Merge-Join: Combina os resultados ordenados de duas subconsultas;
        - Combinação Hash Join: Combina os resultados de duas subconsultas usando uma tabela hash;
        - Combinação Nested Loop Join: Combina os resultados de duas subconsultas iterando sobre cada linha de uma subconsulta e comparando-a com cada linha da outra subconsulta.
- Otimização de Consultas Distribuídas:
    - Técnicas de Otimização:
        - Replicação de Dados: Replicar os dados em vários nós pode reduzir o tempo de acesso aos dados e melhorar a disponibilidade;
        - Cache de Dados: Armazenar em cache os dados frequentemente acessados pode reduzir o tempo de resposta às consultas;
        - Balanceamento de Carga: Distribuir a carga de trabalho uniformemente entre os nós pode evitar gargalos e melhorar o desempenho;
        - Materialização de Vistas: Materializar vistas que são frequentemente consultadas pode reduzir o tempo de resposta às consultas;
        - Reescrita de Consultas: Reescrever consultas para torná-las mais eficientes para execução em um SBDD.

Fases do Processamento de Consultas Distribuídas:

- Análise de Consultas:
    - Análise da sintaxe e semântica da consulta;
    - Identificação das partes da consulta que podem ser executadas em paralelo;
    - Determinação dos requisitos de acesso aos dados da consulta.
- Planejamento de Execução:
    - Seleção dos algoritmos de fragmentação e replicação de dados;
    - Geração do plano de execução otimizado.

- Execução Distribuída:
    - Execução das subconsultas em paralelo nos nós relevantes;
    - Transferência de dados entre os nós.
- Combinação de Resultados:
    - Combinação dos resultados intermediários para gerar o resultado da consulta;
    - Retorno do resultado ao cliente.
- Desafios no Processamento de Consultas Distribuídas:
    - Comunicação e Transmissão de Dados: A comunicação entre os nós pode ser um gargalo, especialmente em redes de baixa velocidade;
    - Falhas de Nó: O sistema precisa poder lidar com falhas de nó e continuar a executar as consultas eficientemente;
    - Consistência dos Dados: O sistema precisa garantir a consistência dos dados, mesmo quando as consultas são executadas em paralelo em diferentes nós;
    - Segurança: O sistema precisa ser protegido contra acesso não autorizado, uso indevido e violações de dados.
- Técnicas para Otimização de Consultas Distribuídas:
    - Replicação de Dados: Replicar os dados em vários nós pode reduzir o tempo de acesso aos dados e melhorar a disponibilidade.

2.5. Controle de Concorrência

Nasseri (2017) expõe que o controle de concorrência assume um papel fundamental na facilitação do acesso concorrente de transações, com o propósito de assegurar a integridade das informações num ambiente de Sistema de Banco de Dados Distribuído (SBDD). Neste contexto, a integridade refere-se à condição na qual o banco de dados mantém um estado consistente tanto no início quanto após uma transação.

A administração da concorrência em um cenário de banco de dados distribuído é desafiadora devido à dispersão dos dados em múltiplos locais, à capacidade dos usuários de efetuar transações em pontos diversos e à possibilidade de o mecanismo de controle não estar imediatamente ciente das atividades ocorridas em outras instâncias.

Em um ambiente de sistema de banco de dados distribuído, uma transação pode demandar acesso a dados armazenados em diversos nós da rede. A grande parte dos algoritmos utilizados para o controle de concorrência distribuída se originam de três categorias principais de princípios preexistentes: algoritmos de bloqueio, algoritmos baseados em marcação temporal e algoritmos de natureza otimista.

2.5.1. Protocolo de bloqueio de duas fases

O algoritmo de bloqueio de duas fases, adotando a premissa “pode ler qualquer informação, mas deve escrever em todas”, opera mediante o bloqueio da leitura das informações acessadas durante uma transação, convertendo esses bloqueios de leitura em bloqueios de escrita para as informações que requerem atualização. Consequentemente, a

versão local dos dados é temporariamente bloqueada, enquanto os bloqueios de escrita são aplicados em todas as versões pertinentes desses dados nos nós correspondentes do sistema distribuído.

Tais bloqueios de escrita permanecem em vigor até que todas as versões relevantes dos dados sejam atualizadas. A liberação dos bloqueios é realizada após a conclusão bem-sucedida da transação ou no caso de ocorrência de um erro, quando então uma mensagem de erro correspondente é emitida.

É pertinente observar que este tipo de controle de concorrência não oferece salvaguardas contra a ocorrência de *deadlocks*. Em situações onde transações se veem impedidas de progredir devido a um impasse, a transação em questão é abortada e reiniciada, acompanhada do envio de uma notificação de aborto (*ABORT*) para registrar tal evento.

2.5.2. Algoritmo de bloqueio *would-wait*

O segundo algoritmo de bloqueio segue a política de "leia tudo, escreva tudo", que diverge do método anterior, especialmente no que diz respeito ao tratamento de impasses, como o *deadlock*. Neste algoritmo, cada transação é atribuída um número com base em sua chegada ao controle de concorrência, garantindo que transações mais antigas não sejam bloqueadas por transações mais recentes.

Se uma transação mais antiga solicitar um bloqueio e isso levar a uma espera por parte de uma transação mais nova, a transação mais recente será prejudicada, resultando em inconsistência e necessidade de reinício. Essa abordagem possibilita que as transações mais recentes aguardem as mais antigas, prevenindo a ocorrência de *deadlocks*.

2.5.3. Sequência de ordenação por data/hora

Ao contrário dos algoritmos anteriores, que empregam carimbos de data/hora para iniciar as transações, este algoritmo adota uma abordagem distinta. Em vez de utilizar um sistema de bloqueio convencional, ele compartilha carimbos de data/hora com todos os itens de dados disponíveis recentemente, e o acesso aos dados é regido pela ordem desses carimbos. Transações que buscam acessar os dados de forma não sequencial são reiniciadas. Quando uma transação solicita a leitura de um item, o carimbo de data/hora de leitura pode diferir do carimbo de data/hora de escrita. Se o carimbo de data/hora da transação solicitante preceder o carimbo de data/hora de escrita do item em questão, a operação de atualização é simplesmente descartada.

No caso de dados em que se aplica a política de "leia qualquer um e escreva todos", a abordagem consiste em enviar uma solicitação de leitura para todas as versões possíveis até que uma solicitação de escrita seja submetida e aprovada por todas as versões pertinentes. A coesão do algoritmo é assegurada pelo emprego de um protocolo de *COMMIT*, no qual os escritores retêm suas atualizações em um espaço de trabalho designado até o momento de confirmar as mudanças.

2.5.4. Protocolo Otimista

Este algoritmo fundamenta-se no uso de carimbos de data/hora. O algoritmo de controle de concorrência otimista opera com base na modificação de informações certificadas. Para cada item de dados, são mantidos carimbos de data/hora de leitura e de

escrita. As transações podem ler ou atualizar itens de dados livremente, armazenando cada atualização em um espaço local até o momento do *COMMIT*.

Em cada leitura, a transação deve revisar a versão do identificador (carimbo de data/hora de escrita) associada ao item lido. Ao finalizar todas as transações e relatar ao controle de concorrência, a transação recebe um carimbo de data/hora exclusivo. Com esse carimbo de data/hora exclusivo, a transação pode enviar uma mensagem pronta para COMMIT para cada transação no grupo.

As transações aprovam cada leitura e escrita localmente. Uma solicitação de leitura é aprovada se a versão lida ainda for a versão atual do item e se nenhuma solicitação de escrita com um carimbo de data/hora mais recente tiver sido aprovada localmente recentemente. Uma solicitação de escrita é aprovada se as próximas leituras não forem aprovadas e confirmadas, e se as próximas leituras não tiverem sido aprovadas localmente recentemente.

2.6. Problemas

O autor Rana (2018) enumera sete tópicos de problemas comuns em bancos de dados distribuídos:

1. **Controle de Concorrência Distribuída:** Para manter a consistência dos dados em um sistema distribuído, é necessário coordenar o acesso concorrente de múltiplas transações aos dados. Isso é alcançado por meio de técnicas e algoritmos de controle de concorrência, os quais garantem que apenas uma transação modifique os dados por vez, prevenindo a ocorrência de deadlocks e assegurando a integridade do banco de dados;
2. **Controle de Replicação:** O controle de replicação de dados envolve a criação de cópias de um banco de dados, total ou parcialmente, e o armazenamento dessas cópias em diferentes locais. Manter a consistência entre essas cópias durante as transações é desafiador e requer mecanismos que garantam a sincronização entre elas;
3. **Tratamento de Deadlock:** O deadlock ocorre quando múltiplos usuários tentam acessar os mesmos recursos do banco de dados simultaneamente, resultando em bloqueios mútuos. Tratar e prevenir deadlocks é essencial para garantir a eficiência do sistema distribuído;
4. **Ambiente do Sistema Operacional:** A seleção adequada de um sistema operacional é crucial na implementação de um ambiente de banco de dados distribuído. O sistema operacional desempenha um papel fundamental no gerenciamento do banco de dados e na compatibilidade com as necessidades da organização;
5. **Gerenciamento Transparente:** A dispersão dos dados em diferentes nós e o acesso simultâneo por diversos usuários são desafios centrais em bancos de dados distribuídos. Portanto, é fundamental garantir o gerenciamento transparente dos dados para manter a integridade do banco de dados;
6. **Segurança e Privacidade:** A aplicação de políticas de segurança em ambientes distribuídos é desafiadora devido à natureza interconectada desses sistemas. É necessário implementar mecanismos robustos de segurança e criptografia para proteger dados sensíveis contra potenciais ameaças ao sistema;

7. **Gerenciamento de Recursos:** Em sistemas distribuídos, os recursos estão distribuídos em diferentes localizações, representando desafios no roteamento e na cooperação eficiente com esses recursos. Um gerenciamento eficaz desses recursos é essencial para o funcionamento adequado do sistema distribuído.

3. Demonstração

Para a realização deste experimento, foi optado pela utilização do MongoDB, um sistema de gerenciamento de banco de dados NoSQL que disponibiliza uma ampla gama de recursos. Esses recursos viabilizam aos desenvolvedores de aplicativos e administradores de bancos de dados a personalização do comportamento de um cluster fragmentado ou de uma implantação de conjunto de réplicas. Dessa forma, o MongoDB oferece a capacidade de aumentar a "sensibilidade ao centro de dados", permitindo a separação operacional e baseada em localização.

O MongoDB possui uma ferramenta nomeada ReplicaSet, traduzida como conjunto de réplicas, é um grupo de processos *mongod* que mantêm o mesmo conjunto de dados, sendo *mongod* o processo *daemon* principal para o sistema MongoDB. Ele lida com solicitações de dados, gerencia o acesso aos dados e executa operações de gerenciamento em segundo plano. Conjuntos de réplicas fornecem redundância e alta disponibilidade, e são a base para todas as implantações em produção. Ele é essencial para garantir redundância e alta disponibilidade dos dados, sendo a base para todas as implantações em produção. Aqui estão algumas informações principais sobre ReplicaSets no MongoDB:

1. **Composição do ReplicaSet**: Um ReplicaSet consiste em vários nós de armazenamento de dados (*data bearing nodes*), que mantêm os dados do conjunto de réplicas, e opcionalmente um nó árbitro (*arbiter node*), que participa do processo de eleição, mas não armazena dados. Um dos nós de armazenamento de dados é designado como primário, enquanto os outros são secundários;
2. **Redundância e Disponibilidade de Dados**: O ReplicaSet proporciona redundância, garantindo que os dados permaneçam acessíveis mesmo na ocorrência de falhas em um dos nós. Isso é possível graças à replicação dos dados entre os membros do ReplicaSet;
3. **Escrita e Leitura de Dados**: O nó primário recebe todas as operações de escrita, enquanto os nós secundários replicam o log de operações (*oplog*) do primário e aplicam as operações em seus próprios conjuntos de dados. Isso permite que os clientes possam enviar operações de leitura para qualquer nó do ReplicaSet;
4. **Eleição de um Novo Primário**: Se o nó primário falhar, os nós secundários elegíveis realizam uma eleição para escolher um novo primário. Esse processo garante a continuidade das operações mesmo em caso de falha de um nó;
5. **Adição de Árbitros**: Em certas situações, como restrições de custo, pode-se adicionar um nó árbitro ao ReplicaSet. O árbitro participa das eleições, mas não armazena dados, ajudando a evitar impasses no processo de eleição;

6. **Operações de Manutenção**: O ReplicaSet suporta operações de manutenção, como adição e remoção de nós, e permite o rebaixamento ou promoção de nós conforme necessário.

Em resumo, o ReplicaSet do MongoDB é uma poderosa ferramenta para garantir a disponibilidade e a integridade dos dados em ambientes distribuídos, oferecendo redundância e alta disponibilidade mesmo em face de falhas de hardware ou rede.

3.1. Implementação

Para implementar a proposta de demonstração, foi criado um cluster, sendo um conjunto de servidores interconectados, que atuam como se fossem um único sistema e trabalham juntos para realizar tarefas de forma mais eficiente e escalável, utilizando ReplicaSet com três nós, de endpoints *localhost:27017, localhost:27018* e *localhost:27019.* A figura 4 representa a topologia do ReplicaSet e a figura 5 apresenta os dados contidos no mesmo.

A partir do ReplicaSet é possível visualizar a replicação dos dados sincronizadamente em todos os nós (servidores) que fazem parte do mesmo. Garantindo alta disponibilidade a partir da redundância. A visualização da eficiência da demonstração é possível ser verificada ao incluir ou alterar dados em um dos servidores.

O padrão acesso aos dados do ReplicaSet é feito a partir do endpoint `"mongodb://localhost:27017,localhost:27018,localhost:27019/?replicaSet=cluster_tb6"`, Essa padronização contém o endpoint de todos os nós do ReplicaSet para que seja possível: Distribuir leituras e gravações entre os nós, melhorando o desempenho e a escalabilidade; Tolerância a falhas, como todos os nós são conhecidos, é possível continuar operando sem interrupção caso um nó venha a falhar, usando os outros nós para atender às solicitações.

Caso uma operação seja enviada para um endpoint de um dos nós isoladamente, como por exemplo o nó de endpoint `"mongodb://localhost:27017"`, caso este nó esteja inalcançável no momento, a operação será perdida e nunca será realizada.

Para manipulação dos dados foi desenvolvido um simples programa em Python, que permite inserir e consultar dados diretamente do endpoint do ReplicaSet bem como aos endpoints dos nós separadamente, sendo eles: `"mongodb://localhost:27017"` (nó primário), `"mongodb://localhost:27018"` (nó secundário) e `"mongodb://localhost:27019"` (nó secundário).

O programa apresenta uma TUI simples, apresentada na figura 6. Como esperado ao realizar uma operação de inserção ou remoção de dados do endpoint do ReplicaSet, a operação realizada é replicada para todos os nós presentes no ReplicaSet, o mesmo ocorre caso a operação seja realizada ao endpoint de um nó separadamente.

Utilizaremos de exemplo o nó de endpoint `"mongodb://localhost:27019"`, vamos realizar uma operação de inserção neste nó e observar o estado do nó de endpoint `"mongodb://localhost:27018"` para verificar a mudança, ambos se encontram com os mesmo dados inseridos, descritos pela figura 5.

Ao realizar a operação citada, representada na figura 7 obtemos a saída do programa, apresentada na figura 8, e podemos observar a replicação do dado inserido no nó de endpoint "mongodb://localhost:27018", evidenciado na figura 9.

```
members: [
  {
    _id: 0,
    name: '127.0.0.1:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 1880,
    optime: { ts: Timestamp({ t: 1711459818, i: 1 }), t: Long('3') },
    optimeDate: ISODate('2024-03-26T13:30:18.000Z'),
    lastAppliedWallTime: ISODate('2024-03-26T13:30:18.528Z'),
    lastDurableWallTime: ISODate('2024-03-26T13:30:18.528Z'),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1711457972, i: 1 }),
    electionDate: ISODate('2024-03-26T12:59:32.000Z'),
    configVersion: 5,
    configTerm: 3,
    self: true,
    lastHeartbeatMessage: ''
  },
  {
    _id: 1,
    name: 'localhost:27018',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 1859,
    optime: { ts: Timestamp({ t: 1711459818, i: 1 }), t: Long('3') },
    optimeDurable: { ts: Timestamp({ t: 1711459818, i: 1 }), t: Long('3') },
    optimeDate: ISODate('2024-03-26T13:30:18.000Z'),
    optimeDurableDate: ISODate('2024-03-26T13:30:18.000Z'),
    lastAppliedWallTime: ISODate('2024-03-26T13:30:18.528Z'),
    lastDurableWallTime: ISODate('2024-03-26T13:30:18.528Z'),
    lastHeartbeat: ISODate('2024-03-26T13:30:25.300Z'),
    lastHeartbeatRecv: ISODate('2024-03-26T13:30:24.346Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: '127.0.0.1:27017',
    syncSourceId: 0,
    infoMessage: '',
    configVersion: 5,
    configTerm: 3
  },
  {
    _id: 2,
    name: 'localhost:27019',
    health: 1,
    state: 2,
    stateStr: 'SECONDARY',
    uptime: 1844,
    optime: { ts: Timestamp({ t: 1711459818, i: 1 }), t: Long('3') },
    optimeDurable: { ts: Timestamp({ t: 1711459818, i: 1 }), t: Long('3') },
    optimeDate: ISODate('2024-03-26T13:30:18.000Z'),
    optimeDurableDate: ISODate('2024-03-26T13:30:18.000Z'),
    lastAppliedWallTime: ISODate('2024-03-26T13:30:18.528Z'),
    lastDurableWallTime: ISODate('2024-03-26T13:30:18.528Z'),
    lastHeartbeat: ISODate('2024-03-26T13:30:25.491Z'),
    lastHeartbeatRecv: ISODate('2024-03-26T13:30:24.728Z'),
    pingMs: Long('0'),
    lastHeartbeatMessage: '',
    syncSourceHost: 'localhost:27018',
    syncSourceId: 1,
    infoMessage: '',
    configVersion: 5,
    configTerm: 3
  }
],
```


Figura 4: Estrutura dos nós do ReplicaSet.

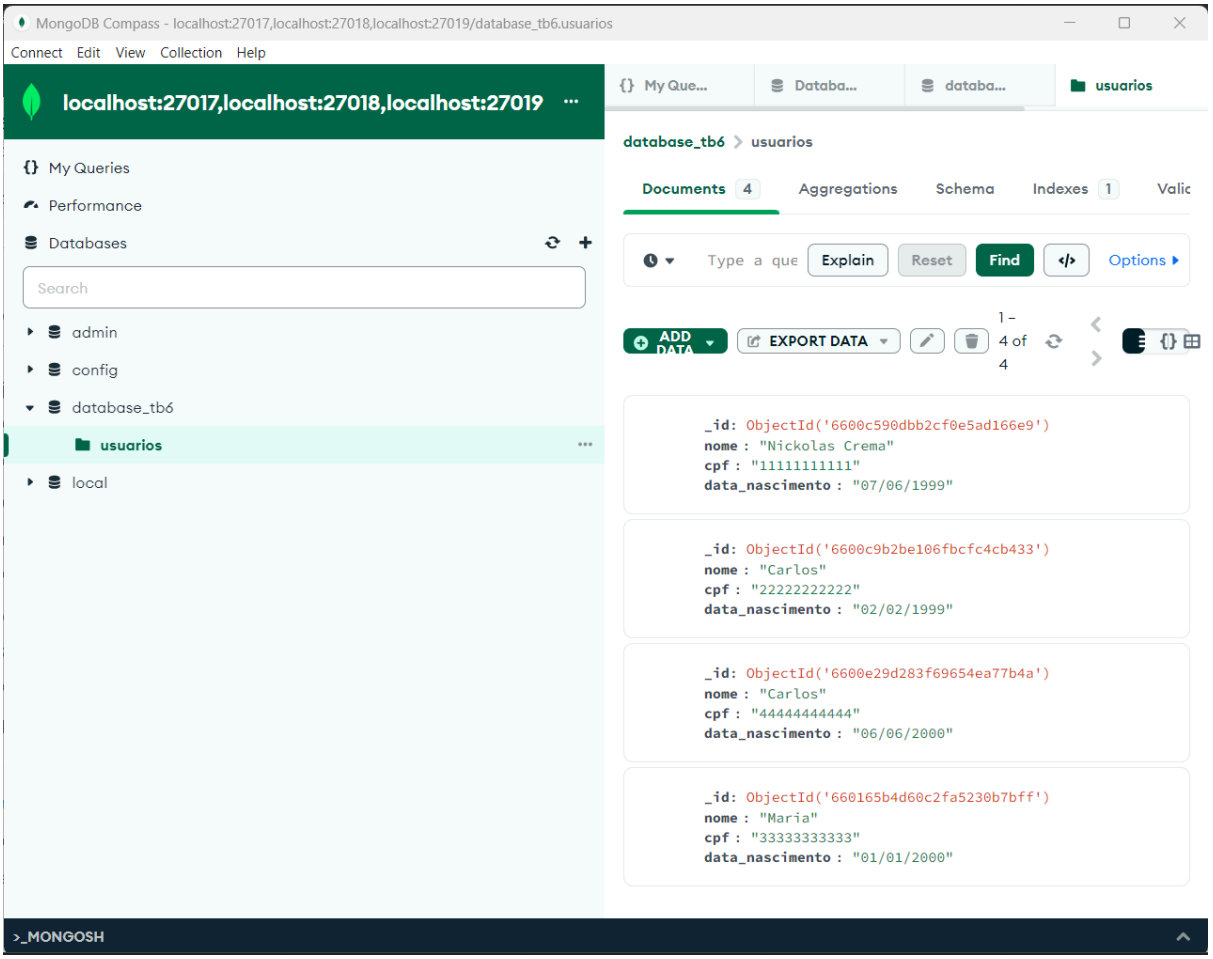

Figura 5: Banco presente no endpoint do ReplicaSet.

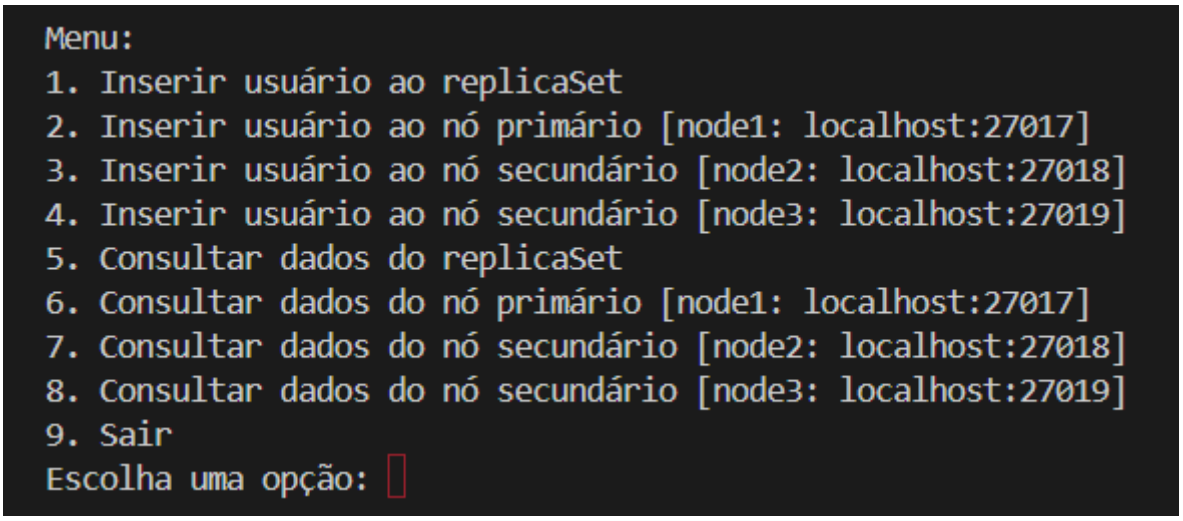

Figura 6: TUI do programa para manipulação dos dados em Python.

```
Menu:
1. Inserir usuário ao replicaSet
2. Inserir usuário ao nó primário [node1: localhost:27017]
3. Inserir usuário ao nó secundário [node2: localhost:27018]
4. Inserir usuário ao nó secundário [node3: localhost:27019]
5. Consultar dados do replicaSet
6. Consultar dados do nó primário [node1: localhost:27017]
7. Consultar dados do nó secundário [node2: localhost:27018]
8. Consultar dados do nó secundário [node3: localhost:27019]
9. Sair
Escolha uma opção: 4
Digite o nome do usuário: Italo Gavassi
Digite o cpf do usuário: 99999999999
Digite a data de nascimento do usuário (DD/MM/YYYY): 05/02/2000
```

Figura 7: Realizando inserção de dados no nó de endpoint “mongodb://localhost:27019”.

```
Dados inseridos no nó secundário [node3]
Endereço: localhost:27019
ID inserido: 6602d4fb9d38bdf85996fefb

Pressione ENTER para continuar...
```

Figura 8: Saída apresentada pelo programa ao inserir um dado no nó de endpoint “mongodb://localhost:27019”.

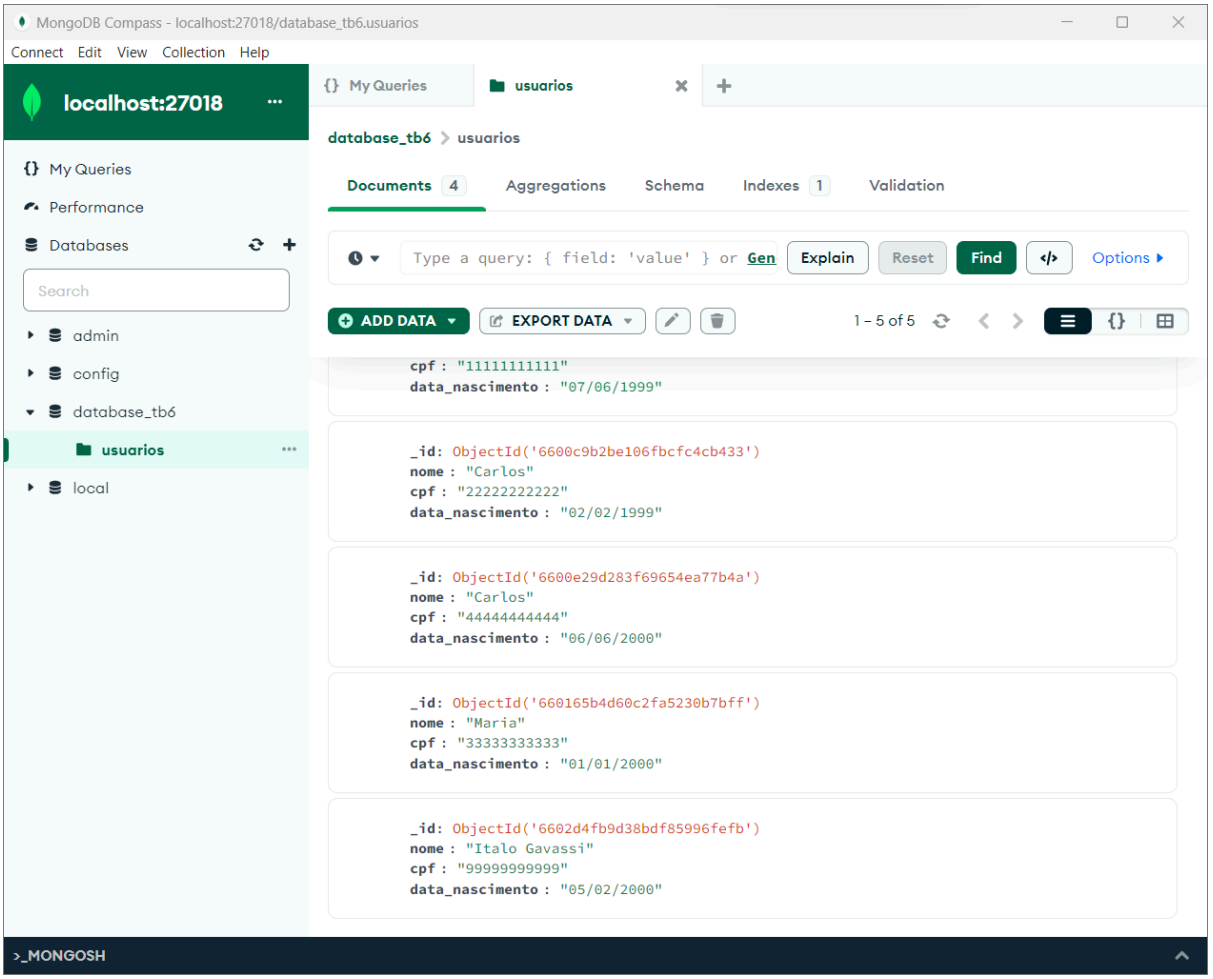

Figura 9: Dados do nó de endpoint “mongodb://localhost:27018” após inserção do dado no nó de endpoint “mongodb://localhost:27019”.

4. Considerações Finais

Compreender os Banco de Dados Distribuídos revela um campo complexo e repleto de desafios e oportunidades. Durante esse estudo, foi identificado a importância da arquitetura distribuída, destacando a distribuição de dados e processamento em uma rede. As vantagens desses sistemas são notáveis, desde a transparência na gestão de dados até o aumento de desempenho e a facilidade de expansão.

O projeto de bases de dados distribuídas também é crucial, exigindo uma abordagem cuidadosa, seja ascendente ou descendente. Além disso, técnicas avançadas de processamento de consultas e controle de concorrência são essenciais para garantir a eficiência e a consistência dos dados em ambientes distribuídos.

É importante reconhecer os desafios enfrentados na implementação e operação desses sistemas, incluindo escalabilidade, consistência de dados, segurança e desempenho. Enfrentar esses desafios requer soluções robustas e uma compreensão profunda das complexidades envolvidas.

Em última análise, os Banco de Dados Distribuídos representam uma área em constante evolução, com oportunidades emocionantes para melhorar a gestão e o processamento de dados em larga escala. Ao compreendermos profundamente esses sistemas,

estamos preparados para enfrentar os desafios e aproveitar ao máximo as oportunidades que eles oferecem. Em resumo, os Banco de Dados Distribuídos representam uma área emocionante e desafiadora da computação, com vastas oportunidades para melhorar a eficiência, escalabilidade e confiabilidade dos sistemas de gerenciamento de dados.

5. Referencias Bibliográficas

MESQUITA, E. J. S.; FINGER, M. **Projeto de Dados em Bancos de dados Distribuídos**. 1998. Tese de Doutorado. Instituto de Matemática e Estatística da Univerisade de São Paulo, 27 de mar de 1998.

CERI, S.; PERNICI, B.; WIEDERHOLD, G. **Distributed database design methodologies**. Proceedings of the IEEE, v. 75, n. 5, p. 533-546, 1987.

WILDEMBERG, M. et al. **Alocação de Dados em Bancos de Dados Distribuídos.** In: SBBD. 2003. p. 215-228.

ÖZSU, M. Tamer et al. **Principles of distributed database systems**. Englewood Cliffs: Prentice Hall, 1999.

MATTOSO, M. **Sistemas de Bancos de Dados**. Universidade Federal do Rio de Janeiro (UFRJ). 2009. Disponível em: <https://www.cos.ufrj.br/~marta/BdDistribuido.pdf>. Acesso em 20 de mar de 2024.

VIOTTI, P.; KAUFMANN, M. **Modern Distributed Database Systems: The Definitive Guide.** 2015. Disponível em: <https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/distributed-data-management.html>. Acesso em 20 de mar de 2024.

CELKO, J.; KAUFMANN, M. **Distributed SQL Databases**. 2017. Disponível em: <https://en.wikipedia.org/wiki/Distributed_SQL>. Acesso em 20 de mar de 2024.

KLEPPMANN, M.; O'REILLY, M. **Designing Data-Intensive Applications.** 2017.

ÖZSU, M. T.; VALDURIEZ, P. **Distributed database systems: where are we now?**. Computer. 24(8), 68–78. doi:10.1109/2.84879. 1991.

ROSÁRIO, L. G. **O que é Banco de Dados Distribuído?**. 2013. Disponível em: <https://imasters.com.br/banco-de-dados/o-que-e-banco-de-dados-distribuido>. Acesso em 21 de mar de 2024.

NASSERI, M et al. **Concurrency Control Methods in Distributed Database: A Review and Comparison**. 2017. Disponível em: <https://www.researchgate.net/publication/319367720_Concurrency_control_methods_in_distributed_database_A_review_and_comparison>. Acesso em 21 de mar de 2024.

RANA, S et al. **Distributed Database Problems, Approaches and Solutions - A Study.** 2018. Disponivel em: <https://www.researchgate.net/publication/328280121_Distributed_Database_Problems_Approaches_and_Solutions_-_A_Study>. Acesso em 21 de mar de 2024.