**Giovani Nascimento Pereira - RA: 168609**

**Matheus Campanha Ferreira - RA: 174435**

**Vitor Kaoru Aoki - RA: 178474**

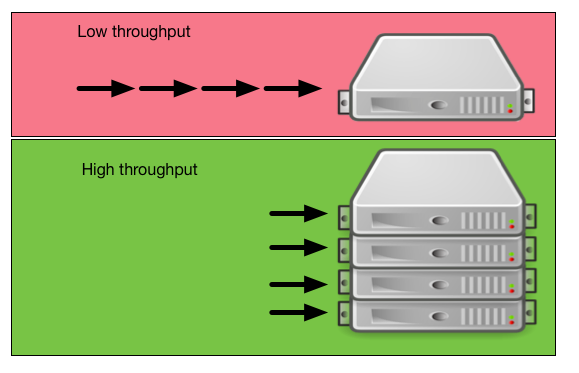
**neo4j**

**Descrição:**

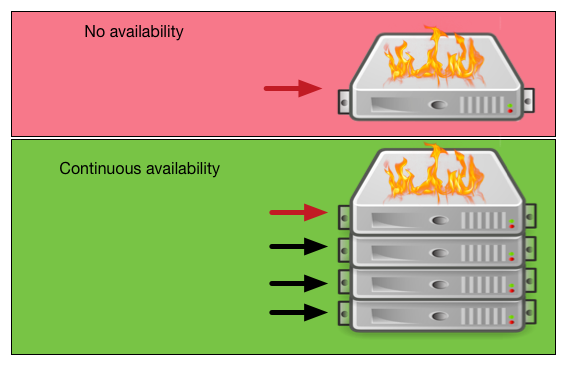
**-** **Arquiteturas Possíveis:**

O neo4j trabalha com uma arquitetura distribuída baseada em Clusters. Com essa abordagem, diversos problemas podem ser contornados, como por exemplo, uma taxa de transferência maior de dados, já que, com um sistema distribuído, os dados não vêm de uma única fonte, o que faz com que o congestionamento de dados seja menor e também a quantidade de fontes para se obter os dados seja maior, fazendo com que eles sejam obtidos com uma taxa de transferência maior.

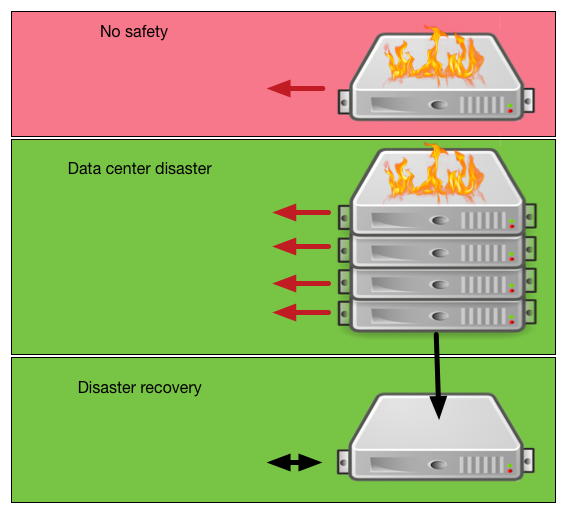
Além disso, por trabalhar com uma arquitetura distribuída e com redundância de dados, em caso de haver algum acidente com um dos clusters que ocasione o seu não funcionamento, o sistema distribuído permite que o banco de dados continue funcionando. Isso ocorre, pois, ao ser detectado o erro, o sistema se encarrega de buscar em outro cluster os dados requeridos, continuando assim, normalmente o funcionamento do banco. Ainda sim, o neo4j possui um sistema de recuperação dos dados do cluster perdido. Esse sistema de segurança se encarrega de recuperar os dados do cluster original, para que a manutenção possa ser realizada e ele não perca seus dados. Essas ferramentas trazem uma maior segurança para o banco, pois os dados estarão seguros e o usuário não perderá acesso ao banco em caso de falhas. Em contrapartida, para que esse sistema funcione bem, é preciso uma quantidade maior de memória para armazenamento dos dados, acarretando em um maior gasto com memória física e também um ótimo controle de sincronização dos clusters, para que funcionem corretamente como uma única máquina, o que demanda um bom sistema de controle e profissionais sempre atentos para qualquer eventualidade.



Comparação entre um sistema com um único servidor e um sistema distribuído baseado em clusters, onde é indicado a diferença na taxa de transferência de dados



Comparação entre um sistema com um único servidor e um sistema distribuído baseado em clusters, onde é mostrado que em caso de alguma falha, o sistema distribuído continua seu funcionamento

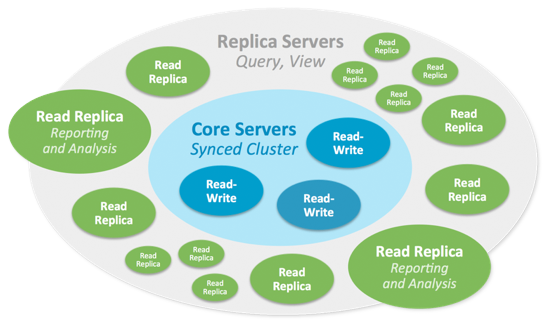


Comparação entre um sistema com um único servidor e um sistema distribuído baseado em clusters, onde é mostrado o sistema de recuperação de dados em caso de falha nos clusters

O neo4j trabalha com dois tipos diferentes de abordagens diferentes para a mesma arquitetura distribuída baseada em clusters:

**- Casual Clusters:**

Os Casual Clusters são baseados em um core, onde se encontram servidores com os dados armazenados e servidores periféricos, chamados de *Read Replicas* que funcionam basicamente como caches dos servidores do core.



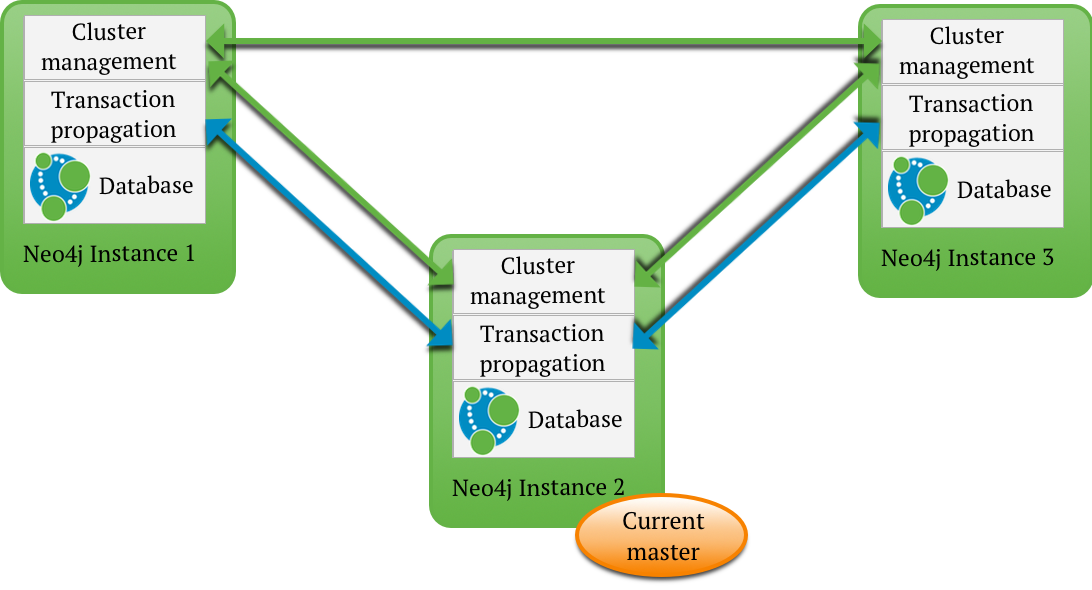
Uma representação de como funciona o Casual Cluster

Os servidores do core trabalham em um protocolo Raft, que garante a segurança de que as transações serão armazenadas em um número suficiente de servidores, a fim de que ao ser realizada uma nova transação, os dados estejam consistentes. Esses servidores do core armazenam os dados do banco e, como apresentam duplicata de dados, permitem uma maior segurança quanto a perda e inconsistência de dados, já que os dados possuem cópias. Porém, como mencionado acima, para isso, é necessário uma maior quantidade de memória física.

Os *Read Replicas* funcionam como caches dos servidores do core. Elas são também parte do banco de dados, mas podem realizar somente ações de leitura. Elas, ao receberem a transação, enviam-na para os servidores do core, para que o dado seja buscado, e retornam-o para o cliente que fez a requisição. Armazenando-o ao final, para caso seja feita uma nova busca por ele. Caso o dado buscado já esteja armazenado, ele é retornado diretamente sem ter que ser acessado no servidores do core. Isso facilita a busca por dados no grafo, de forma a aumentar a velocidade das transações e não sobrecarregarem os servidores do core, o que poderia causar problemas de mau funcionamento e lentidão.

**- Highly Available Clusters:**

Essa outra abordagem é representada por um sistema de mestre-escravo.



Representação do sistema mestre-escravo utilizado nos clusters

Nesse sistema, todos as instâncias do cluster possuem uma cópia dos dados. Quando uma transação de escrita é realizada na instância mestre e *commitada* com sucesso, as instâncias escravas são atualizadas, a fim de ficarem sincronizadas com a mestre. Quando uma transação de escrita é requerida em uma instância escrava, ela é sincronizada com a mestre e, caso seja *commitada* com sucesso, as outras instâncias escravas são atualizadas.

Esse sistema, de mestre-escravo com redundância de dados em todas as instâncias traz uma maior segurança dos dados. Com ele, os dados não correm o risco de serem perdidos e também, caso alguma falha ocorra como interrupção na conexão ou falha de hardware, o banco continua conectado, já que outras instâncias podem suprir a que deu defeito. Caso a instância defeituosa seja a mestre, o sistema possui um sistema inteligente de escolha de qual instância escrava se tornará a nova mestre, fazendo assim, com que o sistema não perca consistência e seja tolerante a falhas, já que a instância principal, que controla as demais, continuará operando para realizar as transações.

Esses mecanismos de segurança e otimização das consultas apresentam um contraponto, que assim com a outra abordagem (Casual Clusters) ele necessita de um espaço de armazenamento físico maior, para que todas as instâncias possuam uma cópia dos dados. Além disso, um bom controle entre as instâncias têm que ser implementado a fim de garantir a perfeita sincronia do cluster e fazer com o sistema continue consistente, além de necessitar para isso, um contínuo trabalho de monitoramento do banco.

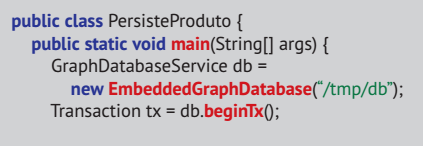
**- Estruturas de Armazenamento:**

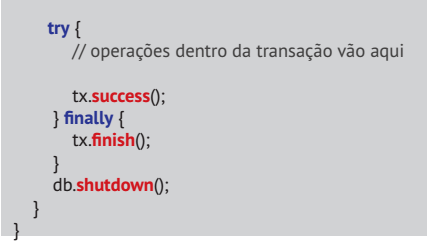
**- Indexação:**

**- Linguagem de Consulta e mecanismos de Processamento e Otimização de Consultas:**

**- Mecanismos de Processamento de Transações e Controle de Concorrência:**

O SGBD neo4j possui o controle de concorrência baseado nas propriedades ACID. Com isso, ele permite a atomicidade do banco, a consistência, o isolamento e a durabilidade do mesmo. Para o controle de transações, é verificado dentro do Cypher se uma transação já existe, ou se uma nova deve ser criada. Caso uma transação já exista, a nova query (seja ela uma busca, ou uma escrita) será rodada dentro do dessa transação já existente e não será interrompida caso a transação não tenha sido *commitada*. Caso uma transação não exista, ela é criada, e a query roda dentro dessa transação, que é *commitada* apenas quando todas as queries forem finalizadas. Esse sistema permite que várias queries sejam *commitadas* como uma única transação.





Exemplo de criação de uma transação e de sua execução

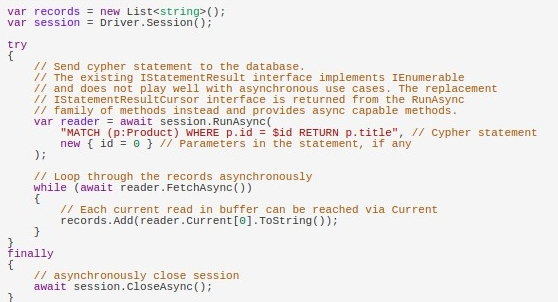
Por seguir as propriedades ACID, o neo4j apresenta uma maior confiabilidade. Por conta delas é possível realizar transações de forma que seja possível manter a consistência dos dados, sem que uma interfira nas ações da outra. A sua ferramenta que permite que várias queries sejam executadas em uma mesma transação, permite uma execução mais rápida das queries, já que elas executam em uma mesma transação, sem que sejam criadas diversas transações, uma para cada query. Isso faz com que não seja perdido tempo por conta da atomicidade de transações. Dentro das transações, é permitido que várias queries sejam executadas, pois o sistema Cypher funciona de forma a realizar as queries em partes. Na execução das queries as ações são realizadas de forma a ou buscar um elemento, ou executar uma escrita no banco.

Com isso, temos um ganho em relação ao tempo de execução de transações. Porém, para que se tenha uma segurança em relação a modificação do banco, para cada transação, as queries executadas, antes de serem *commitadas*, são armazenadas em memória e todas as modificações ficam armazenadas. Isso necessita de um espaço maior de memória, o que pode ser um problema para sistemas com pouca capacidade de memória.

Além disso, o neo4j apresenta outras 3 formas de transação, sendo elas:

**- Auto-commit Transactions:**

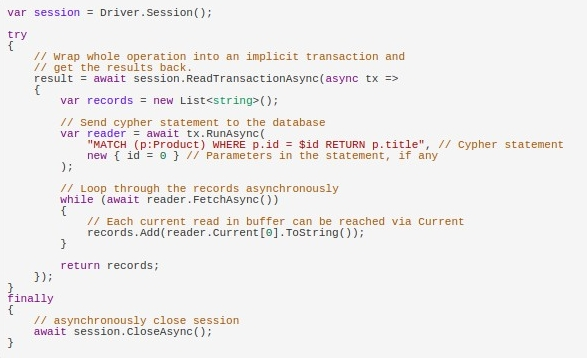
Essa forma de transação é a mais simples apresentada pelo neo4j e consiste apenas em uma transição que será *commitada* automaticamente assim que for finalizada com sucesso. Porém, caso ocorra um erro, ela não reinicia automaticamente, o que se apresenta como um problema, já que uma transação com alto grau de necessidade pode não ser realizada.



Exemplo de Auto-Commit Transaction

**- Transaction Functions:**

Diferentemente das Auto-commit Transactions, as Transactions Functions, possuem o mecanismo de reinicialização da transação caso algum erro seja detectado. Esse mecanismo dá mais segurança ao sistema no sentido de que, caso alguma transação com uma alta importância tenha algum erro durante sua execução, o sistema tentará refazê-la.



Exemplo de Transaction Functions

**- Explicit Transactions:**

As Explicits Transactions são as funções de transações básicas existentes. São elas os métodos *Begin*, *Commit*, *Rollback*, que devem ser utilizados pelo programador a fim de controlar a atomicidade e controle de concorrência das transações. Elas dão ao programador o poder de controle das transações, porém, por outro lado, devem ser utilizadas com cuidado, pois um erro em sua utilização pode fazer com que os programas não funcionem corretamente, ocasionando erros como *Deadlocks*, por exemplo.



Exemplo de Explicit Transactions

**Bibliografia:**

- Mecanismos de Processamento de Transações e Controle de Concorrência:

<https://neo4j.com/docs/developer-manual/current/drivers/sessions-transactions/#driver-transactions-transaction-functions>

<https://neo4j.com/docs/developer-manual/current/cypher/introduction/transactions/>

<http://www.univale.com.br/unisite/mundo-j/artigos/51Neo4j.pdf>

- Arquiteturas Possíveis:

<http://neo4j.com/docs/operations-manual/current/clustering/>

<http://neo4j.com/docs/operations-manual/current/clustering/causal-clustering/introduction/>

<http://neo4j.com/docs/operations-manual/current/clustering/high-availability/architecture/>