ANÁLISE DE DESEMPENHO DE BANCO DE DADOS: POSTGRESQL PADRÃO E UM *CLUSTER* UTILIZANDO O POSTGRES-BDR.

PERFORMANCE ANALYSIS OF DATABASE: DEFAULT POSTGRESQL DATABASE AND A CLUSTER USING POSTGRES-BDR.

Hudson Francis Ventura de Souza[1]
Roberto Benedito de Oliveira Pereira[2]

Resumo

No referido estudo, analisou-se o desempenho de possíveis cenários de um *cluster* de banco de dados, utilizando o Postgres-BDR e comparando-o a uma instalação padrão do PostgreSQL. Em um dos cenários, é simulado um nó *off-line* e é medido o tempo de duração para que o *cluster* faça a sincronia dos dados assim que o nó retorna ao seu estado de funcionamento. Para isso, foi utilizado 100.000 tuplas de uma tabela do banco de dados público do site do IMDB. Em todos os testes foram realizadas as operações de consulta, inserção, atualização, remoção de dados. Foram feitas as operações de CRUD por 100 vezes sobre as 100.000 tuplas em cada cenário proposto. Por fim, foram gerados gráficos e tabelas com estatística básica para cada operação. Conclui-se que o *cluster* do Postgres-BDR muito próximo, em operações de escrita de dados. Porém, na leitura, quando se trata de comparações de testes sequenciais sem a concorrência de recursos, a duração da operação chega a ser 149% superior em relação a instalação padrão do PostgreSQL. Contudo, a perda de desempenho verificada tem um impacto inferior se comparado a indisponibilidade do acesso aos dados que um SGBD centralizado pode causar.

Palavras-chave: replicação de dados; alta disponibilidade; *cluster* de banco de dados

[1] Especialista em Banco de Dados pelo Instituto de Computação da UFMT. Cuiabá - MT, Brasil. E-mail: hudsonventura@outlook.com

[2] Professor Dr. no Instituto de Computação da UFMT. Cuiabá - MT, Brasil. E-mail: roberto@ic.ufmt.br

Abstract

In this study, the performance of a possible scenarios of a database cluster has been analyzed, using Postgres-BDR and comparing it to a standard PostgreSQL installation. In one of the scenarios, an off-line node is simulated, and it is measured the time duration taken in order to the cluster synchronizes the data as soon as the node return to its pattern working state. For this scenario 100,000 tuple of a spread sheet table from the public database from IMDB website were employed. In all the tests were performed the operations of consultation, insertion, updating, removal of data. CRUD operations were performed 100 times over the 100,000 tuples in each proposed scenario. Finally, graphics and tables with basic statistics were generated for each operation. It is concluded that the Postgres-BDR cluster is awfully close in data writing operations. However, in reading, when it comes to sequential test comparisons without the competition of resources, the duration of the operation is up to 149% higher compared to the standard installation of PostgreSQL. However, the loss in performance has a lower impact compared to the unavailability of data access that a centralized DBMS could cause.

Keywords: Data Replication; High Disponibility; Database Cluster

1 INTRODUÇÃO

Uma empresa quando inicia duas atividades no mercado, normalmente utiliza sistemas únicos e centralizados para suprir uma pequena demanda. Com o passar do tempo, essa empresa pode crescer e criar filiais, porém, mesmo com a expansão, ela continua utilizando o mesmo modelo de sistema. Este crescimento e a distribuição geográfica das filiais, gera um aumento na demanda e disponibilidade dos dados. Com isso, é necessário que o sistema corporativo trabalhe de forma distribuída, pois eventualmente um sistema único pode ficar inoperante, e toda a empresa suspende suas operações. Para contornar os problemas de banco de dados centralizados, será abordado aqui uma alternativa, utilizando uma solução de banco de dados com replicação assíncrona em um *cluster* multimestre, que é um conjunto de equipamentos que trabalham para cumprir um único serviço, e multimestre se trata de um tipo de *cluster* que permite a escrita e leitura em todos os equipamentos pertencentes a este *cluster*.

Se o serviço de banco de dados fora disponibilizado por um *cluster*, caso um nó fique inoperante, os outros assumem a demanda e o serviço não é interrompido. Além disso, com vários nós, a carga final suportada pelo *cluster* é superior a um sistema gerenciador de banco de dados (SGBD) trabalhando de forma isolada. Segundo Tanenbaum e Steen (2007), a replicação de dados sob os sistemas distribuídos aumenta a confiabilidade e o desempenho do serviço como um todo.

Diante deste contexto o desenvolvimento dessa pesquisa é considerado por duas situações plausíveis. A primeira é quando uma determinada corporação, que tenha um sistema com um banco de dados centralizado, vivencie uma situação de falha. As informações nele contidas poderão ser perdidas e os serviços oferecidos poderão ser interrompidos, até que o único banco de dados seja normalizado, impactando diretamente no fluxo dos negócios. A segunda situação possível é quando uma filial qualquer da mesma empresa danifique seu *link* de dados. Essa filial ficará inoperante até que o *link* e o acesso ao banco de dados sejam reestabelecidos. Assim como Canedo, Teixeira e Bruschi (2013) citam, a indisponibilidade de dados, tempo inativo ou paradas não planejadas resultam em perda de produtividade e receita, desempenho financeiro fraco e danos à reputação. Para contornar essas situações é preciso uma estratégia, a fim de minimizar os efeitos de um mau funcionamento e indisponibilidade de acesso aos dados. Por isso, é necessário a utilização de métodos de replicação de dados.

A finalidade desse estudo é realizar testes de desempenho em uma tecnologia capaz de minimizar o problema de falha de um banco de dados centralizado. Sendo assim, poderá promover a replicação de dados entre equipamentos, obtendo uma maior disponibilidade (caso um equipamento falhe, o tráfego de dados é direcionado para outro de forma

transparente à aplicação) e tolerância a falhas. Será apresentada uma solução da replicação de dados para o sistema gerenciador de banco de dados (SGBD) PostgreSQL, que atenda o critério de multimestre (vários nós no *cluster* que permitem escrita e leitura de dados). Além de avaliar essa solução por meio de testes de desempenho, comparando o banco de dados PostgreSQL tradicional com um *cluster* banco de dados com replicação utilizando o Postgres-BDR.

Este trabalho trata-se de uma pesquisa exploratória, usando testes práticos, e está estruturado da seguinte forma: inicialmente é apresentado uma revisão dos conceitos necessários para o completo entendimento sobre um *cluster* de banco de dados, seguido da explanação do *hardware* e *softwares* utilizados nos testes. Após, são apresentados os cenários montados, e os métodos aplicados nos testes. Por fim, nos resultados, são expostos gráficos exibindo o comportamento dos testes e uma conclusão sobre a experiencia do uso do serviço do banco de dados com e sem o *cluster*.

2 REVISÃO DA LITERATURA

Nesta seção são abordados os conceitos necessários para o entendimento do funcionamento de um *cluster*.

2.1. *Cluster* de Banco de Dados

Segundo Tanenbaum e Steen (2007) um *cluster* consiste em um conjunto de estações de trabalho, ou computadores semelhantes, conectados por meio de uma rede local de alta velocidade. O *cluster* pode ser formado por dois ou mais equipamentos autônomos chamados de nós, interligados entre si, com a finalidade de oferecer serviços.

Focado em um *cluster* de banco de dados, segundo Almeida (2016), é definido como uma coleção de bancos de dados conectados por uma rede de computadores, e um sistema de gestão distribuída e transparente aos usuários. Cada nó tem seu próprio sistema operacional (SO) e seu SGBD único, mas todos trabalham para prover um mesmo serviço. Os principais tipos de *cluster* possuem objetivos definidos:

- *Cluster* de alto desempenho: desempenha um trabalho no menor tempo possível;
- *Cluster* de balanceamento de carga: equilibra a carga de trabalho do *cluster*, evitando que os recursos de um nó cheguem ao limite;

- *Cluster* de alta disponibilidade: garantir a disponibilidade dos serviços oferecidos pelo maior tempo possível.

Nas subseções de 2.1 à 2.4 é discorrido sobre as características necessárias para o funcionamento de um *cluster*.

2.2. Alta Disponibilidade

A alta disponibilidade é uma técnica que permite que o serviço oferecido no *cluster* permaneça o maior tempo possível acessível para servir aos clientes. Um *cluster* precisa ser tolerante a falhas de *hardware* e *software*. Segundo Tanenbaum e Steen (2007), um sistema de alta disponibilidade é aquele que mais provavelmente está funcionando em dado instante no tempo. Já Diesel e Ramão (2015) concluem que a alta disponibilidade tem como função permanecer o sistema ativo por um longo período e em plena condição de uso.

Um dos pré-requisitos para se chegar à alta disponibilidade de um *cluster* de banco de dados é a replicação de dados por meio de uma ou mais cópias distribuídas em nós diferentes. Supõem-se um sistema distribuído com dois nós: caso um nó deixe de funcionar, o outro assume, instantaneamente, de forma transparente. Assim, os usuários e aplicações clientes deste *cluster* não sentem diferença alguma quanto houver uma falha.

2.3. Replicação de Dados

A replicação é a manutenção de cópias dos dados em vários nós. Para Tanenbaum e Steen (2007) dados são replicados para aumentar a confiabilidade de um sistema e para aumentar o desempenho total do *cluster* por meio do balanceamento de carga. Na replicação, várias cópias dos dados são distribuídas entre os nós do sistema. Cada nó tem uma cópia completa dos dados, ou uma cópia com os dados inerentes ao nó. Almeida (2016) afirma que bancos de dados distribuídos são replicados com o propósito de garantir disponibilidade do sistema, já que os itens de dados estarão acessíveis em múltiplos nós. A replicação pode ser:

- Síncrona: ao atualizar um dado no nó principal, o gerenciador de replicação distribui a alteração para todos os outros nós do *cluster* e aguarda as respectivas respostas. Assim que todos os nós respondem com uma confirmação, o nó principal informa ao gerenciador que a alteração foi finalizada com sucesso. Somente neste momento o gerenciador libera recurso para atender uma nova demanda.
- Assíncrona: ao atualizar um dado no nó principal, o gerenciador de replicação tenta distribuir a alteração para os outros nós, mas não aguarda as confirmações, guarda o dado

em cache e libera recursos para atender uma nova demanda. Em um segundo momento, o gerenciador descartará o dado após realizar a confirmação das alterações em todos os nós do *cluster*.

As possíveis estruturas do *cluster* que influencia o funcionamento da replicação divide-se em:

- Mestre - escravo: nessa organização, um nó do *cluster* funciona como um sistema completo, fornecendo escrita e leitura de dados; já o outro nó funciona como uma cópia, permitindo que os clientes que o consultem tenham um acesso somente leitura.
- Multimestre: nessa organização, todos os nós do *cluster* funcionam tanto para leitura como para escrita. Todos os nós realizam uma mescla das informações com os dados dos outros nós.

2.4. Tolerância a Falhas

Tanenbaum e Steen (2007) afirmam que a tolerância a falhas é a característica de um sistema que pode prover seus serviços mesmo na presença de irregularidades de *hardware* ou *software*. Por mais que um disco pare ou a comunicação de um nó seja interrompida, o serviço oferecido pelo *cluster* não cessa. A tolerância a falhas está diretamente ligada à alta disponibilidade. Canedo, Teixeira e Bruschi (2013) declaram que para atenuar um ponto único de falha, sistemas são projetados com redundância, sendo que o serviço oferecido só irá falhar caso todos os componentes do sistema falhem ao mesmo tempo.

Um sistema tolerante a falhas não indica que defeitos não acontecerão, mas sim, que o *cluster* permite que as falhas estejam presentes, e, assim mesmo, não afetará o funcionamento geral dos serviços. As falhas devem ser transparentes aos usuários bem como suas correções, ou seja, nada deve ser perceptível aos usuários.

2.5. Ferramentas de Apoio

Nesta seção são abordados os *softwares* responsáveis pelo o funcionamento do *cluster*.

2.5.1. PostgreSQL

O PostgreSQL é um SGBD objeto-relacional de código-fonte aberto lançado em 1989. Segundo a sua empresa desenvolvedora PostgreSQL Global Development Group (2019), o PostgreSQL usa e estende a linguagem SQL, cumpre os princípios do ACID (atomicidade, consistência, isolamento e durabilidade), possui linguagem procedural e suporte a chaves estrangeiras, controle de concorrência multi-versionado, tablespaces e registrador de

transações sequencial WAL (do inglês *write ahead logging*) para tolerância a falhas. Segundo Almeida (2016), utiliza a licença Berkeley Software Distribution (BSD) que permite o uso para fins comerciais, sem a necessidade de pagamentos de taxas.

2.5.2. Postgres-BDR

Conforme a empresa desenvolvedora 2ndQuadrant (2019), o Postgres-BDR é uma ferramenta de replicação multimestre para bancos de dados PostgreSQL criada em 2014 e disponível apenas para sistemas baseados em Linux. É a primeira solução multimestre de código aberto para PostgreSQL. O termo BDR provém de replicação bidirecional (do inglês *Bi-Directional Replication*), e que, permite que as modificações dos nós do *cluster* sejam realizadas de forma bidirecional. O PostgreSQL possui eficiência e precisão, garantindo alta disponibilidade de todos os nós geograficamente distribuído.

3 MATERIAIS E MÉTODOS

Nesta seção apresenta-se os *softwares* e *hardwares* utilizados bem como suas configurações. Outrossim, é abordado os cenários em que os testes foram realizados.

Das escolhas dos *softwares*, optou-se pelo PostgreSQL, pois é um *software* livre e de uso gratuito (inclusive comercialmente), multiplataforma, robusto, sendo limitado pelos recursos de *hardware*. Em conjunto com o PostgreSQL, escolheu-se o Postgres-BDR que, além de seguir a mesma linha de código aberto, tem uma simples instalação e possui a replicação assíncrona multimestre. Ademais, os *hardwares* foram utilizados os disponíveis naquele momento, considerando-se que possuíam desempenho aceitável para os testes.

3.1. *Hardware* e *Software*

Nos testes foi utilizado um computador HP Elitedesk 705, com processador AMD A8 PRO-7600, com quatro núcleos de 3.1GHz, 12GB de RAM 1600MHz DDR3, SSD Kingston A400 de 120GB e rodando um Windows Server 2016 Standard build 1607. Neste equipamento foram instaladas duas máquinas virtuais (nó 1 e nó 2) para comporem o *cluster*.

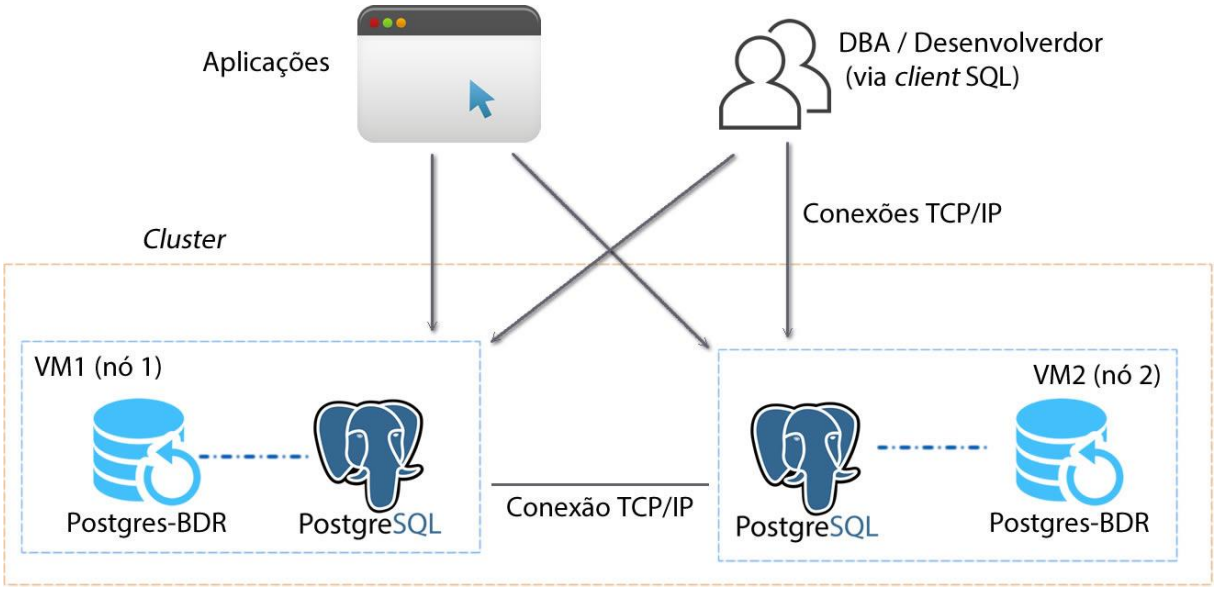
As VMs são gerenciadas pelo Microsoft Hyper-V v10. Cada uma tem como *hardwares* virtuais dois núcleos de CPU, 2GB de RAM, 30GB de armazenamento, e *softwares* GNU/Linux Ubuntu 18.04 LTS Bionic Beaver, PostgresSQL v9.4 e PostgreSQL-BDR v1.0.2-2019-02-28.

3.2. Cenário

Para esta pesquisa, foi produzido o cenário da 1 para realização de testes de desempenho e prova de funcionamento de um *cluster* utilizando o PostgreSQL e o Postgres-BDR. Os nós comunicam-se entre si através de uma rede IP (Protocolo de Internet, do inglês *Internet Protocol*). Cada PostgreSQL comunica um com o outro, por meio do Postgres-BDR que é o responsável pela organização de toda a replicação.

No nó 1 foram instalados e configurados o PostgreSQL e o Postgres-BDR. Assim que toda a comunicação fora efetivada e validada, a mesma configuração fora replicada o nó 2. Após os nós estarem devidamente funcionais, foram referenciados os nós do *cluster*. No nó 1 referenciou-se o nó 2 e no nó 2 referenciou-se o nó 1. Para este processo foi utilizado o manual de Thanh (2017).

Figura 1 - Cenário de teste para alta disponibilidade de um *cluster* de banco de dados com PostgreSQL



Fonte: Próprio autor (2019)

Os cenários propostos por esse estudo são derivados do contexto da 1. Em um momento será utilizado o *cluster* com os dois nós e, em outro momento, em um dos nós será simulado uma situação de falha temporária (que há um retorno ao funcionamento normal, em um momento posterior) e uma falha permanente (sem o retorno ao funcionamento normal). Também será utilizada uma instalação comum do PostgreSQL sem o uso do Postgres-BDR para medidas de referências de desempenho. Deste modo, os testes realizados utilizam as

quatro operações básicas nas tuplas de um SGBD que são: criar, ler, atualizar e deletar (CRUD, do inglês *create, read, update and delete*) e o cenário apresenta as variações:

A. Uma instalação padrão do PostgreSQL em uma VM, fora do *cluster*, usada como referência;
B. Um nó único no *cluster* (PostgreSQL + Postgres-BDR);
C. Dois nós *on-line* no *cluster* (PostgreSQL + Postgres-BDR em ambos os nós);
D. Um nó *on-line* e um nó *off-line* representando um defeito permanente;
E. Um nó *on-line* e um nó *off-line*, representando um defeito temporário. Posteriormente o nó *off-line* é restaurado, simulando sua recuperação;

Para manter o equilíbrio entre todas as variantes dos cenários, os arquivos de configuração dos cenários foram mantidos no seu padrão.

3.3. Método de Realização dos Testes

Para os testes, foi utilizado o banco de dados público do *Internet Movie Database* (IMDB) com informações de filmes, séries, músicas, *games* e seus principais participantes como atores, dubladores, diretores e produtores. O banco de dados foi baixado em 20/03/2018, contendo mais de 47 milhões de tuplas, divididas entre cinco tabelas e com 5,53 GB, em formato de texto, separado por tabulações (TVS). É possível visualizar o modelo lógico do banco do IMDB na Figura 2.

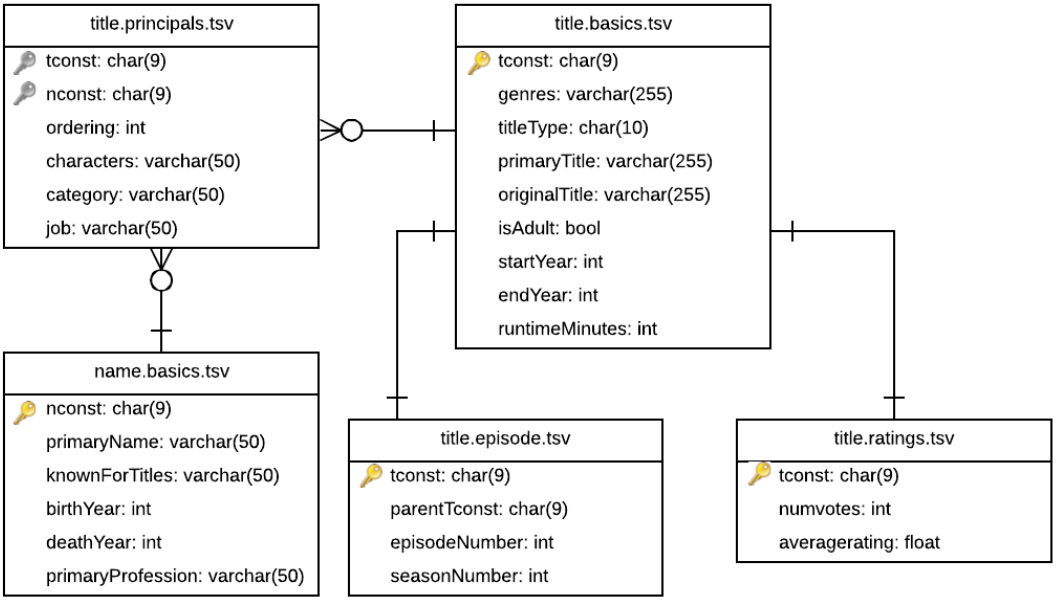
Foi criado um algoritmo escrito em C# para realizar os testes de desempenho de maneira automatizada e consecutiva, utilizando o *Entity Framework*[3] para abstração da comunicação com o SGBD. O algoritmo realiza as operações CRUD apenas sobre a tabela "name.basics.tsv" do modelo da Figura 2. Essa tabela foi eleita para a realização dos testes de CRUD por conter mais tuplas em comparação com as demais, mas limitou-se a 100.000 tuplas por conta de limitações de *hardware*. O fluxograma do funcionamento macro do algoritmo pode ser observado na Figura 3 e o código completo está disponível em https://github.com/hudsonventura/AutomatizadorArtigo. Os testes registraram os tempos iniciais e finais de cada conjunto de operações sobre as 100.000 tuplas, com o intuito de permitir um comparativo para cada diferente configuração do cenário inicial.

---

[3] *Entity Framework* é uma ferramenta de persistência de dados que permite a abstração das classes da aplicação e o SGBD. Documentação disponível em: https://docs.microsoft.com/pt-br/ef/

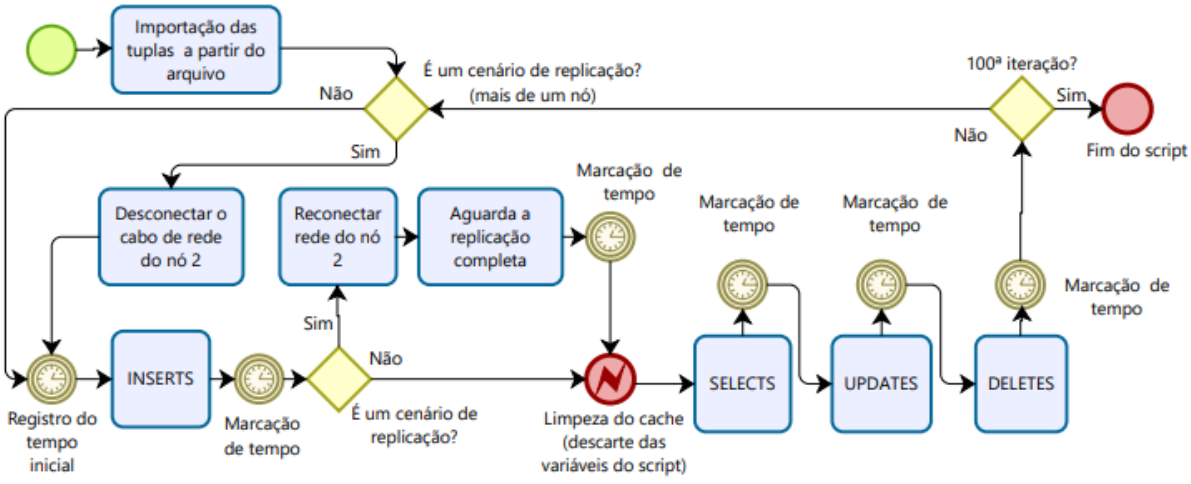
Para as configurações de testes das variações do cenário de A, B e D, foram feitos os testes de CRUD. Já as variações C e E, logo após as inserções das tuplas, foi feito o registro do tempo de replicação dos dados, e na sequência, as operações de leitura, atualização e remoção de tuplas, assim como expõem a Figura 3. Não foram utilizados índices nos bancos do *cluster*, a fim de potencializar a sobrecarga dos SGBDs. No cenário D o *cluster* tem 2 nós, mas um deles fica permanentemente *off-line* e não há replicação de dados.

Figura 2 - Modelo lógico do banco de dados do IMDB



Fonte: Próprio autor (2019)

Figura 3 - Script de Testes Automatizados



Fonte: Próprio autor (2020)

No cenário E, logo antes da operação de inserção de dados, o cabo de rede é desconectado da máquina virtual 2 (nó 2), simulando a dessincronia de dados. Então as tuplas são inseridas no nó 1. Após as inserções, o cabo de rede é reconectado ao nó 2 e é aguardado até o término de sincronia de dados para registro do tempo.

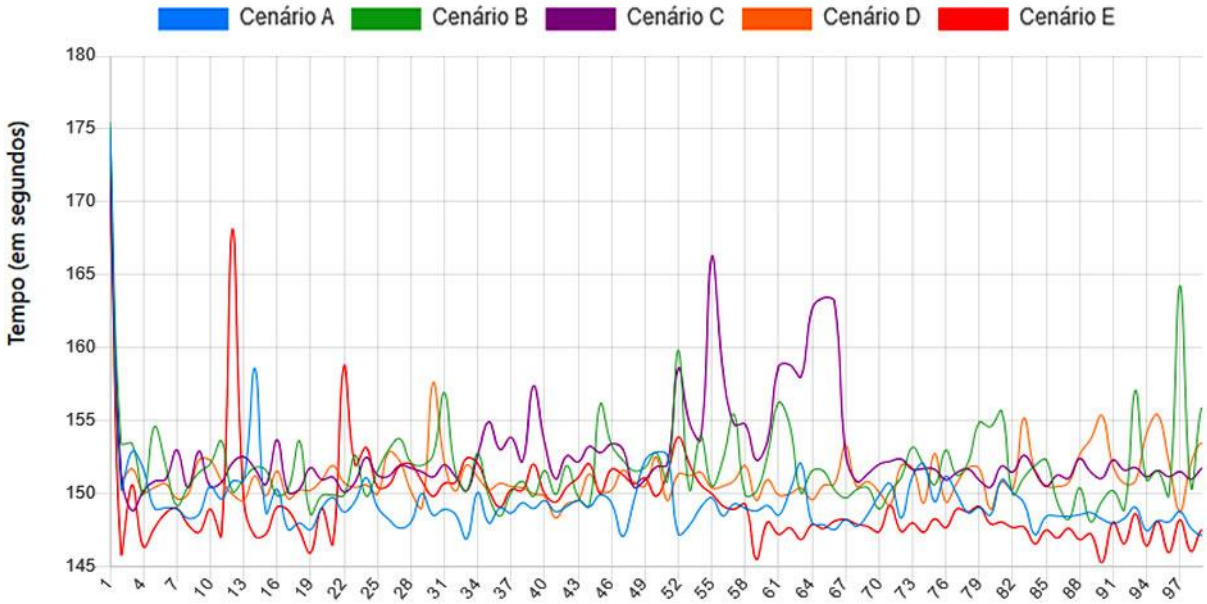
4 RESULTADOS E DISCUSSÕES

Nesta seção são apresentados os resultados obtidos nos testes. As subseções estão em ordem das operações realizadas pelo *script* de automação. Para cada teste de operação, 100.000 tuplas foram atingidas por 100 vezes. Isso permitiu que fossem gerados gráficos de tempo x iteração, apresentando o tempo de duração em segundos de cada iteração. Em conjunto, uma tabela com as estatísticas básicas foi gerada para cada operação.

4.1. *Insert*

A primeira operação feita pelo *script* de automação dos testes é o *select*. Na Figura 4 e na Tabela 1, é possível visualizar o comportamento e as estatísticas dessa operação. Neste teste foi registrado o momento inicial, após foram inseridas 100.000 tuplas e por fim foi registrado o momento final. Os tempos obtidos são dados pela diferença entre o momento inicial e momento final de cada iteração. A Tabela 1 resume os valores das 100 iterações.

Figura 4 - Comportamento dos testes de inserção de 100.000 tuplas



Fonte: Próprio autor (2019)

Tabela 1 - Estatística sobre os tempos dos testes de inserção de 100.000 tuplas

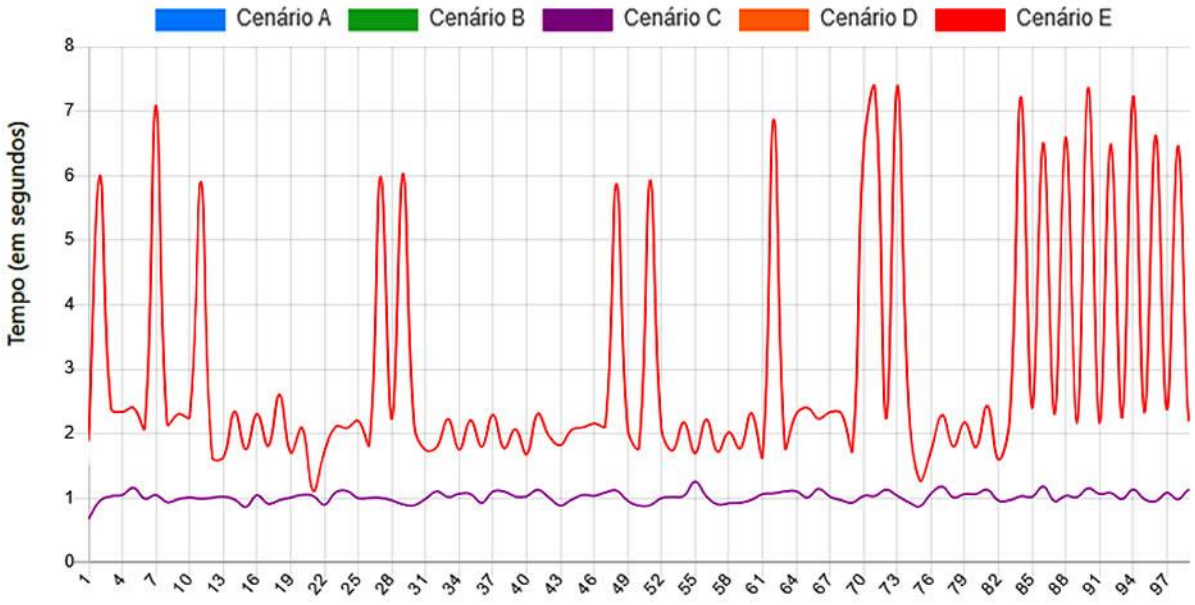
	Variação A	Variação B	Variação C	Variação D	Variação E
Mínima	146,931	148,091	148,846	148,372	145,32
Média	149,564	152,021	154,902	151,262	149,409
Máxima	175,051	175,446	171,142	170,285	174,001
Amplitude	28,119	27,355	22,297	21,913	28,261
Variância	9,309	11,751	12,342	5,942	14,373
Desvio Padrão	3,051	3,428	3,513	2,438	3,791

Fonte: Próprio autor (2019)

4.2. Replicação

A Figura 5 exemplifica o comportamento dos tempos de replicação de dados do *cluster*. Logo após as inserções de tuplas, o teste de replicação foi realizado. Este teste só foi possível para os cenários C e E, pois, são os únicos que possuem dois nós que realizam a sincronização de dados. Os cenários A e B possuem apenas um nó, e o cenário D possui o nó 2 permanentemente *off-line*, portanto, não estão presentes tanto na Figura 5 quanto Tabela 2.

Figura 5 - Comportamento dos testes de replicação de 100.000 tuplas



Fonte: Próprio autor (2019)

Tabela 2 - Estatística sobre os tempos dos testes de inserção de 100.000 tuplas

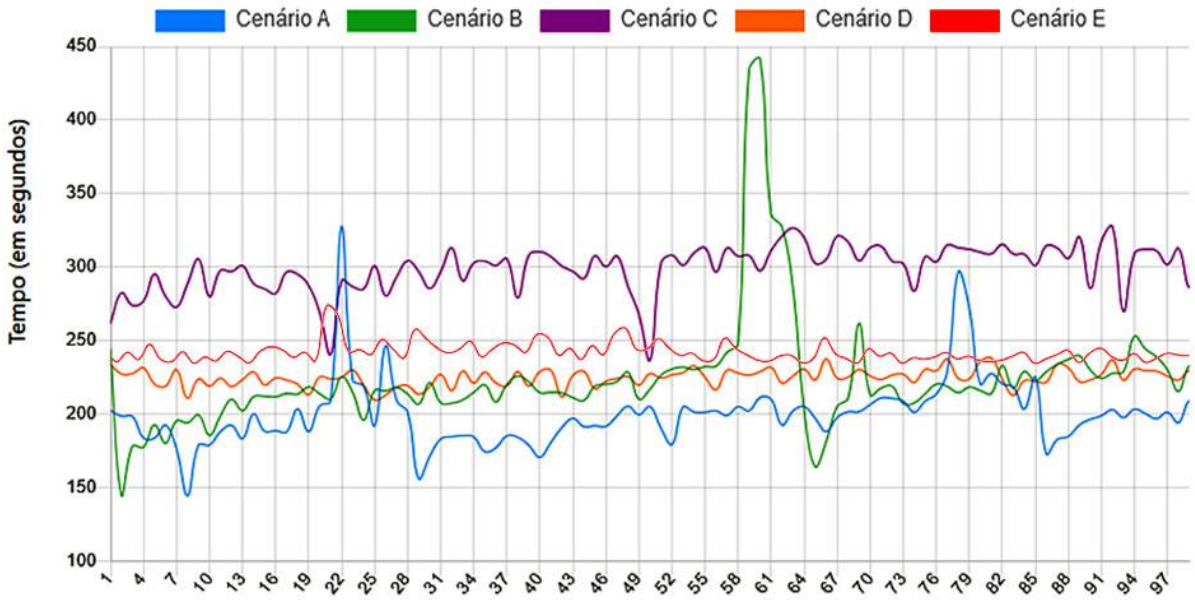
	Variação A	Variação B	Variação C	Variação D	Variação E
Mínima	-	-	0,676	-	1,108
Média	-	-	1,017	-	2,958
Máxima	-	-	1,257	-	7,398
Amplitude	-	-	0,581	-	6,290
Variância	-	-	0,007	-	3,511
Desvio Padrão	-	-	0,085	-	1,874

Fonte: Próprio autor (2019)

4.3. *Select*

No gráfico da Figura 6, é possível ver o comportamento das iterações em ordem crescente para cada estado do cenário versus o tempo de duração em segundos. Na Tabela 3, é exibida uma estatística básica dos tempos da consulta da Figura 6 para todas as variações do cenário de teste possíveis.

Figura 6 - Comportamento dos testes de consulta de 100.000 tuplas



Fonte: Próprio autor (2019)

Tabela 3 - Estatística sobre os tempos dos testes de consulta de 100.000 tuplas

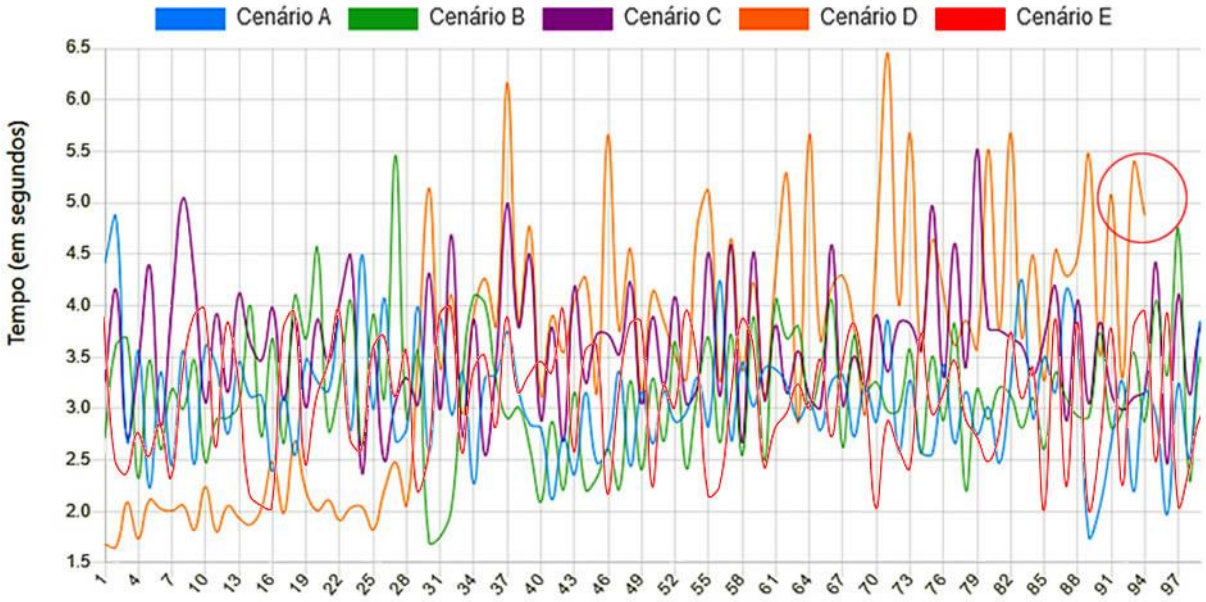
	Variação A	Variação B	Variação C	Variação D	Variação E
Mínima	144,859	145,312	236,957	209,792	237,390
Média	200,163	226,839	299,185	225,102	245,021
Máxima	327,718	441,715	327,705	238,494	276,359
Amplitude	182,859	296,403	90,748	28,702	38,969
Variância	548,031	1556,952	264,757	35,362	44,933
Desvio Padrão	23,410	39,458	16,271	5,947	6,703

Fonte: Próprio autor (2019)

4.4. *Update*

Na Figura 7 e na Tabela 4 são exibidos o comportamento e estatística dos tempos de atualização todos os atributo de todas as 100.000 tuplas. Mesmo atualizando todas as tuplas por iteração, o tempo é baixo se comparado à operação anterior. Em todos os cenários os tempos médios são próximos.

Figura 7 - Comportamento dos testes de atualização de um atributo em 100.000 tuplas



Fonte: Próprio autor (2019)

Tabela 4 - Estatística sobre os tempos dos testes de atualização de um atributo em 100.000 tuplas

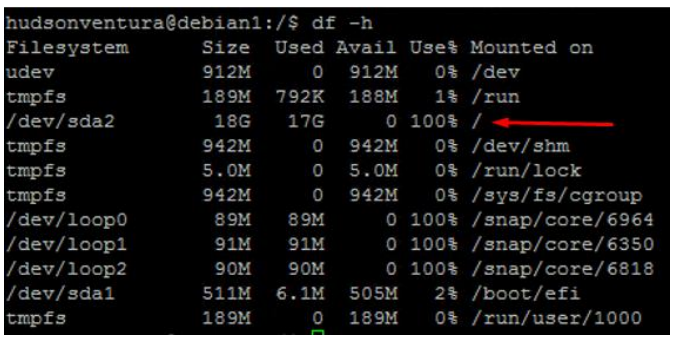
	Variação A	Variação B	Variação C	Variação D	Variação E
Mínima	1,752	1,696	2,372	1,651	2,020
Média	3,073	3,138	3,612	3,550	3,089
Máxima	4,856	5,458	5,517	6,455	3,990
Amplitude	3,104	3,761	3,145	4,804	1,970
Variância	0,339	0,416	0,405	1,502	0,378
Desvio Padrão	0,582	0,645	0,636	1,226	0,615

Fonte: Próprio autor (2019)

Foi enfrentado um problema de consumo de disco no destaque da Figura 7, onde é possível ver uma quebra na linha que ilustra os testes da variação D do cenário inicial na iteração 96. Neste caso, houve o consumo de 100% do disco de armazenamento.

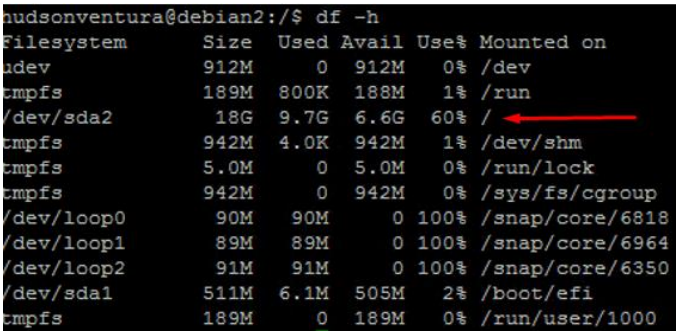
Para analisar tal situação, incialmente, foi utilizado o comando “df -h” no terminal *bash* no sistema operacional de cada nó para avaliar uma situação global dos discos. O nó um, estava ativo e recebeu todas as 100.000 tuplas por 96 iterações, enquanto o nó dois não recebeu alteração alguma pois estava com o cabo de rede virtual desconectado. No destaque da Figura 8, o diretório “/dev/sda2” está com seu consumo em 100%, diferente do nó que ficou *off-line*, como na Figura 9. O diretório “/dev/sda2” consiste no local onde está montado o diretório “/” (raiz do sistema) e, nele, estão localizados os arquivos de dados dos SGBDs.

Figura 8 - Consumo de disco do nó 1 *on-line*



Fonte: Próprio autor (2019)

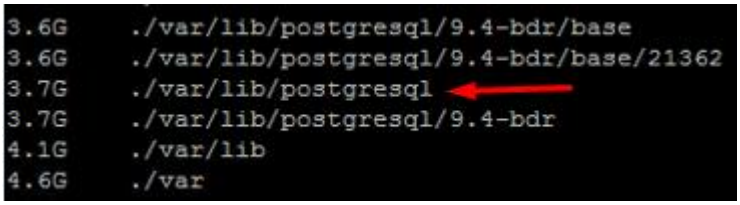
Figura 9 - Consumo de disco do nó 2 cujo cabo de rede fora desconectado (*off-line*)



Fonte: Próprio autor (2019)

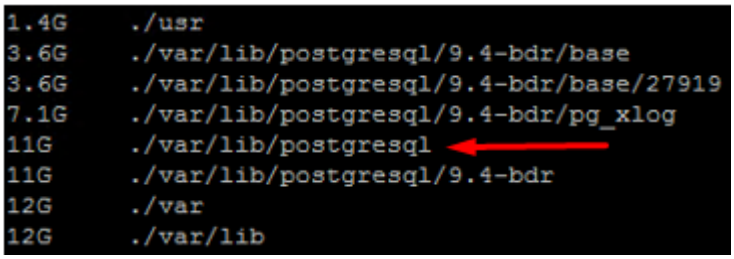
Após uma melhor análise, chegou-se aos diretórios com maior consumo de espaço em disco. Finalmente, foi avaliado o diretório "/var/lib/postgresql" onde normalmente, nos testes, deveria estar com 3.7GB, como pode ser visto na Figura 10. Entretanto, tal diretório chegou a 11GB na Figura 11, consumindo todo o espaço disponível da partição "/dev/sda2".

Figura 10 - Consumo de disco dos diretórios de dados e logs do SGBD do nó cujo cabo de rede fora desconectado (*off-line*)



Fonte: Próprio autor (2019)

Figura 11 - Consumo de disco dos diretórios de dados e logs do SGBD do nó *on-line*



Fonte: Próprio autor (2019)

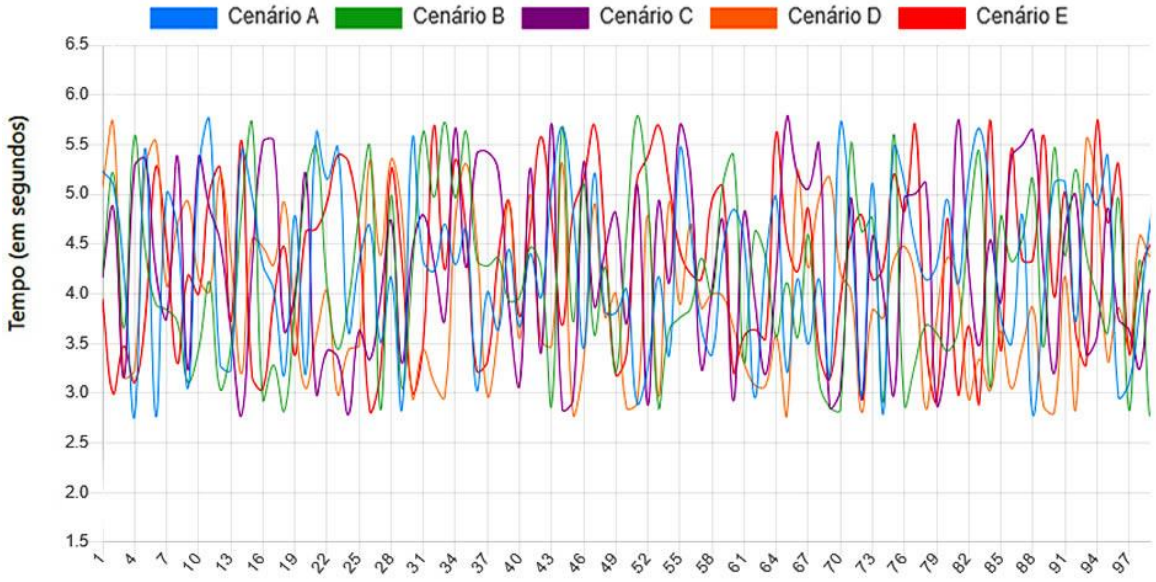
Concluiu-se que o Postgres-BDR grava todas as alterações a serem realizadas sob o banco não sincronizado (nó 2), não em cache ou em arquivo, mas no próprio SGBD *on-line* (nó 1). Nas 96 iterações realizadas, o Postgres-BDR grava todos as 100.000 tuplas por 96 iterações, não considerando que os *deletes* realizados neste teste devem sobrepor os *inserts* e *updates* anteriores. Se há uma remoção das tuplas, o Postgres-BDR poderia descartar todos as

tuplas desse *log* feito até então, todavia, isso não ocorre. Sendo assim, o espaço disponível em disco foi consumido até seu limite.

4.5. *Delete*

No gráfico da Figura 12, é possível ver o comportamento das iterações em ordem crescente para cada estado do cenário versus o tempo de duração em segundos. Tanto na Figura 12 quanto na Tabela 5 percebe-se um comportamento de todos os cenários que rodam dentro de 3 e 6 segundos, e seus tempos médios são semelhantes.

Figura 12 - Comportamento dos testes de remoção de um atributo em 100.000 tuplas



Fonte: Próprio autor (2019)

Tabela 5 - Estatística sobre os tempos dos testes de remoção de 100.000 tuplas

	Variação A	**Variação B**	**Variação C**	**Variação D**	**Variação E**
Mínima	2,765	2,780	2,775	2,779	2,821
Média	4,290	4,195	4,253	3,995	4,319
Máxima	5,746	5,786	5,774	5,719	5,741
Amplitude	2,981	3,005	2,998	2,940	2,920
Variância	0,745	0,771	0,829	0,682	0,720
Desvio Padrão	0,863	0,878	0,910	0,826	0,849

Fonte: Próprio autor (2019)

4.6. Considerações

Usando os tempos médios para conclusão das 100 iterações das operações de CRUD, criou-se a Tabela 6, contendo o tempo médio gasto nas iterações de cada cenário medido segundos, e o valor em percentual de todas as variantes do cenário proposto.

Para calcular cada valor em percentual da Tabela 6 , dividiu-se a duração de cada uma das operações pela duração do tempo do cenário A, ou seja, utilizou-se como base o valor de uma instalação padrão do PostgreSQL como 100% do tempo. Cada valor acima de 100% representa um desempenho inferior, e, um valor inferior a 100% representa um desempenho superior se comparado com o tempo de duração do cenário A. Sendo assim quanto menor o percentual apresentado, melhor o desempenho obtido no cenário.

Tabela 6 - Variações dos tempos médios dos testes (quanto menor, melhor)

Variações do cenário	***Select***	**Replicação**	***Insert***	***Update***	***Delete***
Cenário A	200s - 100%	-	149s - 100%	3,1s - 100 %	4,2s - 100 %
Cenário B	226s - 113%	-	152s - 102%	3,2s - 102 %	4,1s - 97 %
Cenário C	299s - 149%	1,01s	154s - 103%	3,6s - 117 %	4,2s - 99 %
Cenário D	222s - 111%	-	151s - 101%	3,5s - 115 %	4,0s - 93 %
Cenário E	245s - 122%	2,96s	149s - 100%	3,1s - 100%	4,3s - 101 %

Fonte: Próprio autor (2019)

Nota-se que nas operações de *insert* e *update*, as variantes têm um tempo de duração igual ou ligeiramente superior, ou seja, seus desempenhos são iguais ou pouco inferiores ao cenário A. O *delete* foi a única operação que apresentou cenários em que houve desempenhos superiores ao cenário A. A única operação que apresentou um desempenho consideravelmente menor, ou seja, um tempo de duração maior, foi a operação de *select*.

Quanto a replicação, não foi possível obter os valores em percentual pois o canário A não contempla este tipo de operação, então, apenas apresentou-se os valores obtidos, assim como explanado na seção 4.2. O cenário C, que possuem ambos os nós *on-line*, o tempo de replicação é reduzido se comparado ao cenário E, que o nó 2 fica *off-line* logo antes da operação de *insert* e *on-line* logo após todas as 100.000 inserções por iteração. O desempenho inferior da replicação do cenário E pode ser influenciado não só pelo do tempo de acesso ao disco, mas também pela latência de *hardware* pelo reestabelecimento da comunicação ao *cluster*.

5 CONCLUSÕES

Na prática, observou-se que o *cluster* do Postgres-BDR possui pequenas perdas no desempenho, quando comparado a uma instalação padrão do PostgreSQL. As operações de *insert*, *update* e *delete* possuem uma diferença no desempenho que variam em geral entre 93% a 117% da duração do cenário A. Porém, o *select* possui de 111% até 149% da duração da referência, sendo um atraso considerável. Vale salientar que os testes foram realizados de maneira serializada e não houve concorrência de recursos do *cluster*. Em contrapartida, temos os dados replicados e que podem estar distribuídos geograficamente, garantindo assim uma maior disponibilidade dos dados e, em caso de disputa de recursos, o *cluster* pode se sobressair em relação a instalação padrão do PostgreSQL.

Além da perda no desempenho, houve problemas quando um nó permanece muito tempo desativado. O nó que recebe as alterações guarda as modificações no próprio SGBD para posterior atualização. Isso fez com que o disco de armazenamento fosse completamente consumido nos testes, impedindo que se chegasse até à última iteração para a operação de *update*. Isso representa perda de dados, quebrando a alta disponibilidade proposta.

Foi identificado que o tempo de recuperação *on-line* do nó defeituoso, a ressincronia, é muito menor que uma operação de inserção comum, pois o Postgres-BDR realiza apenas a escrita de disco, não necessitando das validações inerentes a operação de *insert* de SQL. Caso um nó fique desativado por um tempo e retorne depois, a recuperação deste nó, além de ser automática, é mais rápida do que uma importação a partir de um arquivo de texto.

Contudo, a perda de desempenho verificada tem um impacto muito inferior se comparado à perda de dados e/ou indisponibilidade do acesso aos dados. A replicação de dados é um passo para a alta disponibilidade e, esse tipo de segurança, é imprescindível em ambientes produtivos, onde a integridade dos dados são essenciais.

Foi encontrado uma dificuldade nos testes de *update* na seção 4.4 onde houve o consumo de 100% do disco de armazenamento. Foram realizadas várias sessões de testes para que fosse identificado a causa do problema e obter a conclusão.

Outros cenários, não testados neste trabalho e que se deseja testar, é a inclusão de mais um nó e utilizando máquinas físicas idênticas para todos os nós, uma vez que com máquinas virtuais em um *hardware* único, pode existir concorrência de recursos da máquina física, podendo trazer resultados diferentes aos vistos aqui.

Como trabalhos futuros, apontar-se um passo-a-passo com os processos de compilação, instalação, montagem e operação de *cluster* real com o Postgres-BDR, exatamente como feito nos testes.

6 REFERÊNCIAS BIBLIOGRÁFICAS

2NDQUADRANT PostgreSQL-BDR; **AlwaysOn Multi-master Replication for Distributed PostgreSQL Database**. Disponível em: https://www.2ndquadrant.com/en/resources/postgres-bdr-2ndquadrant/. Acesso em: 07 mai. 2019

ALMEIDA, A. R. **Um Estudo Sobre Aplicação de Benchmark em Sistema de Banco de Dados Distribuído Homogêneo Baseado em PostgreSQL**. 2016. 77 p. Trabalho de Conclusão de Curso (Tecnólogo de Banco de Dados) - Faculdade de Tecnologia de Lins. São Paulo, 2016. Disponível em: http://www.fateclins.edu.br/v4.0/trabalhoGraduacao/RzgySPkpDNg7DLOEz8WEHtxEjpFuus8EvBan6.pdf. Acesso em: 09 mai. 2019

CANEDO, F.; TEIXEIRA, V.; BRUSCHI, G. Gerenciamento e Alta Disponibilidade em Armazenamento de Banco de Dados. **e-F@tec**, São Paulo, v3, n. 1, p. 121-132, mar. 2018. Disponível em: http://revista.fatecgarca.edu.br/index.php/efatec/article/view/55. Acesso em: 09 mai. 2019

DIESEL, F. L.; RAMÃO, F. P. **Cluster de Alta Disponibilidade com Ferramentas Open Source**. 2015. 14 p. Trabalho de Conclusão de Curso (Especialização em Redes de Computadores) - Faculdade Alfa Brasil, Cascavel. Disponível em: https://docplayer.com.br/18268332-Cluster-de-alta-disponibilidade-com-ferramentas-open-source-high-availability-cluster-with-open-source-tools.html. Acesso em: 03 out. 2020

NAVATHE, S.; ELMASRI, R. **Sistemas de banco de dados**. Tradução: Daniel Vieira. 6. ed. São Paulo: Pearson, 2011. cap. 25, p. 589-619

PostgreSQL Global Development Group; **What is PostgreSQL**. Disponível em: https://www.postgresql.org/about/. Acesso em: 07 mai. 2019

SILBERSCHATZ, A.; KORTH, H. F. SUDARSHAN, S. **Sistema de Banco de Dados.** Tradução: Daniel Vieira. 5. ed. São Paulo: Elsevier Editora Ltda, 2006. cap. 22, p. 561 - 585

TANENBAUM, A. S.; STEEN, M. V. **Sistemas Distribuídos, Princípios e Paradigmas**. Tradução: Arlete S. Marques. 2. ed. São Paulo: Pearson, 2007. cap. 1, 7 e 8, p. 1, 165, 194 e 195

THANH, L. Y. **Multi-Master Replication for PostgreSQL Database With PostgreSQL-BDR**. Disponível em: https://medium.com/@yenthanh/multi-master-replication-for-postgresql-databases-with-postgres-bdr-eb6d8b1bc189. Acesso em: 07 mai. 2019