Uma Abordagem Dinâmica para Configuração de Timeout do DRBD Usando Inteligência Artificial em um Cenário de Nuvem

Gláucio R. Vivian[1], **André Fiorin**[1]

[1]Curso de Tecnologia em Sistemas para Internet – Instituto Federal Farroupilha (IFFar)
Caixa Postal 169 – 98.400-000 – Frederico Westphalen – RS – Brazil

{glaucio.vivian,andre.fiorin}@iffarroupilha.edu.br

Abstract. *The predominant computational paradigm in data storage and processing is cloud computing. This in turn demands high availability and fault-tolerant computing resources. Among the numerous options for existing software, DRBD appears as an efficient solution for this purpose in the Linux operating system. The objective of this work is to improve the functioning of DRBD when used in the cloud in a dynamic scenario such as the Internet, where variations in bandwidth and latency occur. In the end, a decrease in errors and replication time was observed when using a link with high latency (800ms).*

Resumo. *O paradigma computacional predominante no armazenamento e processamento de dados é a computação em nuvem. Esta por sua vez demanda por recursos computacionais de alta disponibilidade e tolerante a falhas. Dentre as inúmeras opções de softwares existentes, o DRBD aparece como uma solução eficiente para este fim no sistema operacional Linux. O objetivo deste trabalho é melhorar o funcionamento do DRBD quando utilizado na nuvem em cenário dinâmico como é a Internet, onde ocorrem variações de banda e latência. Ao final se observou a diminuição de erros e tempo de replicação quando utilizado um link com alta latência (800ms).*

1. Introdução

De acordo com [Srivastava and Khan 2018], atualmente vivenciamos a era da computação em nuvem na industria da Tecnologia da Informação (TI). Pode-se destacar uma enorme diversidade de aplicações que utilizam este tipo de tecnologia, como por exemplo: NetFlix[1], YouTube[2], Facebook[3], Instagram[4], Spotify[5], dentre outros. Com base nessa observação, pode-se concluir que as soluções que buscam proporcionar alta disponibilidade de serviços hospedados na nuvem são muito relevantes.

A alta disponibilidade é um conceito que busca garantir que um sistema computacional esteja sempre disponível para o usuário, mesmo na presença de falhas. Uma maneira de garantir isso é utilizando redundância, que consiste em fazer o espelhamento de dados de um dispositivo de bloco entre diferentes servidores. Um exemplo de ferramenta

[1]https://www.netflix.com/
[2]https://www.youtube.com/
[3]https://www.facebook.com/
[4]https://www.instagram.com/
[5]https://www.spotify.com/br/

para alcançar a alta disponibilidade através da redundância é o DRBD[6] (*Distributed Replicated Block Device*). Trata-se de um software cuja principal finalidade é a replicação de dados de um nodo principal para um nodo redundante, que é utilizado em caso de falhas no primeiro [REISNER 2020].

Para garantir a efetividade no espelhamento dos dados, o DRBD conta com um detector de defeitos baseado em *timeout*. Isso significa que o servidor redundante periodicamente envia mensagens do tipo "*are you alive?*" para o servidor primário, que por sua vez deve responder com uma mensagem do tipo "*i'm alive*" em um determinado tempo limite de resposta (*timeout*) [CHAVES 2014].

Estimar um valor de *timeout* preciso pode ser bastante complexo, pois a eficácia do serviço baseado neste parâmetro depende bastante das características da rede na qual o sistema atua. Em outras palavras, instabilidades na rede, como por exemplo perdas e atrasos na entrega de pacotes, podem interferir na replicação correta dos dados. Neste sentido, um valor de *timeout* muito pequeno pode fazer com que falhas sejam detectadas mais rapidamente, porém, com uma probabilidade maior de falsos positivos. Em contrapartida, com um valor de *timeout* maior, falsos positivos são menos frequentes, ao preço de uma detecção de falhas mais demorada [CHAVES 2014].

Para sanar estes problemas, os detectores de falhas adaptativos são capazes de trabalhar com variações nas condições de rede, alterando dinamicamente seu *timeout*. Com base nisso, este trabalho propõe a implementação de um modelo baseado em Inteligência Artificial que considera variáveis como a latência de rede sobre a Internet e largura de banda disponível para calcular o RTT (*Round Trip Time*) e ajusta dinamicamente o parâmetro de *timeout* do DRBD.

Este artigo está organizado da seguinte forma: a Seção 2 apresenta um referencial teórico sobre Inteligencia artificial, Computação em Nuvem, Tolerância a Falhas e Alta Disponibilidade. A Seção 3 apresenta os trabalhos relacionados. A Seção 4 apresenta a metodologia do trabalho. A Seção 5 são apresentados os resultados dos experimentos e finalmente, na Seção 6 as conclusões e trabalhos futuros são apresentados.

2. Referencial Teórico

Nesta seção são apresentados os principais conceitos necessários para compreensão do trabalho proposto. Aborda-se a linha de computação em nuvem, tolerância a falhas, sistema de alta disponibilidade e, ao final, são relatados os trabalhos científicos diretamente relacionados ao tema proposto.

2.1. Inteligência Artificial

A definição de Inteligência Artificial (IA) é algo bastante discutido e não há um consenso sobre o seu real significado. Essa dificuldade origina-se no problema de definir inicialmente o que é inteligência humana, podendo ser definida basicamente sobre três acepções distintas: *i*) **teológica**: vista como um dom divino dado ao homem; *ii*) **filosófica**: fonte da intelectualidade; *iii*) **psicológica**: capacidade de resolver problemas. No âmbito da computação, normalmente refere-se a inteligência artificial como uma disciplina do conhecimento humano para sistemas que pensam e agem como humanos ou logicamente [PEREIRA 2020].

[6]https://www.linbit.com/drbd/

Um teste clássico proposto por Alan Turing consiste em colocar um humano interagindo com uma inteligência artificial sem saber desse fato. Caso o humano não consiga distingui-la, pode-se assumir que trata-se de inteligência artificial. Um sistema computacional necessita ter pelo menos as seguintes capacidades para se passar por um humano: processamento de linguagem natural, representação de conhecimento, raciocínio automatizado e aprendizado de máquina [PEREIRA 2020].

De acordo com [PEREIRA 2020], a inteligência artificial pode ser historicamente dividida em quatro momentos distintos: *i)* **(1943-1950)**: início das pesquisas voltadas para desenvolvimento de neurônios artificiais; *ii)* **(1951-1969)**: surgiram os primeiros computadores capazes de jogar xadrez, provar teoremas, imitar o raciocínio humano, linguagem natural e aprender por analogia; *iii)* **(1970-1980)**: neste período o entusiasmo diminui devido a limitações no processamento e armazenamento dos dados; *iv)* **(1981-presente)**: retorna a ser uma área de pesquisa muito ativa devido ao surgimento de novos recursos tecnológicos.

De acordo com [PEREIRA 2020] *apud* [BARRETO 2001], existem três abordagens para inteligência artificial: *i*) **conexionista**: baseia-se no fato que é suficiente para criar IA a reprodução completa do cérebro humano. Exemplo: redes neurais; *ii*) **simbólica**: aceita como suficiente um conjunto de símbolos e regras de manipulação dos mesmos. Exemplo: sistema especialista; *iii*) **evolucionária**: baseia-se na teoria da evolução proposta por Darwin, onde operações genéticas de mutação e recombinação são utilizadas. Exemplo: algoritmo genético. Pode-se ainda combinar as três abordagens existentes em uma nova abordagem denominada hibrida, onde o objetivo é encontrar uma melhor solução para um problema muito especifico.

Redes neurais são um ramo da IA que trata de modelos computacionais inspirados na compreensão biológica do sistema nervoso central de animais, com foco no cérebro, seus neurônios e suas ligações complexas. Um neurônio pode ser modelado matematicamente como um somatório de todos os seus valores de entrada, cada um multiplicado por respectivo peso. O resultado do somatório é aplicado a função de ativação (função sigmoide) com a finalidade de tornar a saída mais normalizada [Abiodun et al. 2018].

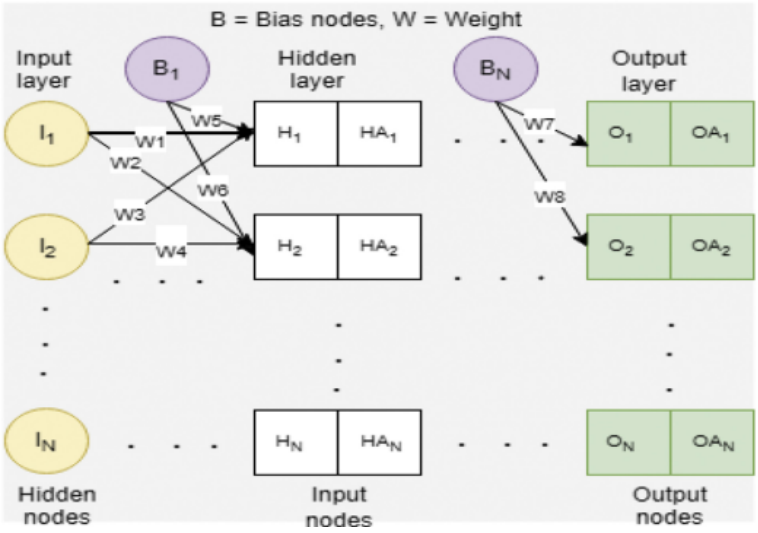

Figura 1. Topologia de uma tipica rede neural [Abiodun et al. 2018].

A Figura 1 apresenta uma rede neural construída pela camada de entrada (amarelo), a camada oculta (branco), e camada de saída (verde). Os pesos estão representados por W e os nodos de bias estão na cor violeta.

O treinamento de uma rede neural consiste em encontrar um conjunto de pesos que façam a ela se comportar aquedadamente com os dados de treinamento. O algoritmo mais empregado para a fase de treinamento é o *Back Propagation*[7], que propaga o erro da saída para entrada, ajustando os pesos com base na derivada da função de ativação.

2.2. Computação em Nuvem

O nome da computação em nuvem advém do fato de que sempre utilizou-se uma nuvem em desenhos esquemáticos para representar a Internet. Pode-se afirmar que trata-se de um paradigma de computação onde essencialmente os dados são armazenados na Internet (nuvem). A alguns anos atrás a computação em nuvem era vista com uma tendência, atualmente já pode ser considerada uma realidade. Fatores como a maior oferta de Internet banda larga e disseminação do uso de *smartphones* são sem dúvida os principais responsáveis por tornar realidade [TECMUNDO 2020].

O essencial desse paradigma é tornar disponíveis os dados do usuário em qualquer local e com o mínimo possível de softwares necessários para o funcionamento. A maioria das aplicações demanda apenas um *browser*. Alguns casos também podem utilizar aplicativos.

De acordo com [SOFTLINE 2020], essencialmente existem três modelos de computação em nuvem:

- **SASS** - *Software as a Service* (Software como Serviço): modelo que oferece o software fim já instalado, necessitando o mínimo possível de configuração para colocar a solução em funcionamento. Exemplos: Gmail[8], Dropbox[9], Google Drive[10]
- **PASS** - *Plataform as a Service* (Plataforma como Serviço): oferece a plataforma necessária já configurada. Em outras palavras, o usuário deve apenas se preocupar com o desenvolvimento do software necessário para o seu caso. Pode-se destacar como exemplo o Google *Cloud Plataform*[11]
- **IASS** - *Infrastructure as a Service* (Infraestrutura como Serviço): neste modelo, comercializa-se o hardware de um *datacenter*. Diferentes parâmetros de configuração estão disponíveis: número de CPU, quantidade de memória RAM, espaço em disco, tráfego mensal de rede. Trata-se do modelo mais flexível, contudo exige maior demanda de trabalho para instalar todos os softwares necessários. Exemplo: máquina virtual na Digital Ocean[12]

Além disso, a nuvem também pode ser classificada de acordo com o acesso às suas informações em [QINETWORK 2020]:

- **Nuvem pública:** todos os dados estão disponíveis para qualquer usuário na Internet.

[7]https://en.wikipedia.org/wiki/Backpropagation
[8]https://mail.google.com/
[9]https://www.dropbox.com/
[10]https://drive.google.com/
[11]https://cloud.google.com/
[12]https://www.digitalocean.com/

- **Nuvem privada:** os dados estão restritos a apenas quem possuir credenciais de acesso à nuvem.
- **Nuvem híbrida:** trata-se da junção das duas anteriores, onde de acordo com a necessidade os dados podem ser públicos ou privados.

2.3. Tolerância a Falhas

De acordo com [NUNES 2020], um sistema é tolerante a falhas quando o mesmo continua desempenhando o seu papel num cenário onde ocorrem falhas de software ou hardware. Para o usuário, essas falhas devem ser imperceptíveis dentro das limitações impostas ao sistema tolerante. São exemplos de técnicas de tolerância a falhas os sistemas redundantes (confiabilidade) e de alta disponibilidade.

Na literatura se encontram diversas denominações para os conceitos básicos da área. Neste trabalho optou-se por utilizar as definições apresentadas por [WEBER et al. 1990]. A falha está associada com o nível mais básico, ou seja, o hardware. Aqui se pode citar como exemplos a queima de um disco rígido e a alteração de bits na memória RAM. O erro por outro lado ocorre no universo da informação, ou seja, quando é algo de interesse do usuário. Cabe observar que uma falha pode não resultar em erro, pois pode ser que afete algo irrelevante para o usuário. Já o defeito é quando ocorre a manifestação do erro causado pela falha, visível para o usuário. Na Figura 2 podem ser visualizados os conceitos apresentados e seu universo.

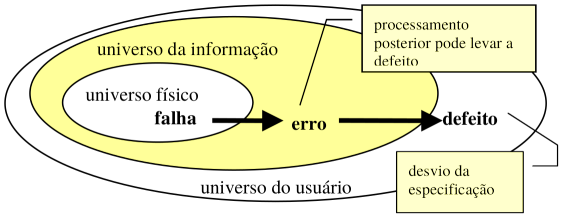

Figura 2. Universo da falha, erro e defeito [WEBER 2003].

Na tolerância a falhas o maior objetivo a ser alcançado é a dependabilidade (tradução para o termo inglês *dependability*). Ela indica a qualidade do serviço fornecido por um dado sistema e a confiança que deve ser depositada no mesmo. Na Tabela 1 se apresenta as principais medidas de dependabilidade[WEBER 2003].

Tabela 1. Medidas de dependabilidade [WEBER 2003].

Dependabilidade	*dependability*	qualidade do serviço oferecido
Confiabilidade	*reliability*	capacidade de atender a especificação, dentro das especificações
Disponibilidade	*availability*	probabilidade de um sistema estar operacional num determinado instante
Segurança	*safety*	probabilidade do sistema executar a sua função ou descontinuar para não provocar dano
Segurança	*security*	proteção contra falhas causadas por ameaças maliciosas

Dentre as acima citadas, a mais utilizada em sistemas de informação é a confiabilidade, que na prática é descrita por meio das seguintes medidas: taxas de defeitos, MTTF, MTTR e MTBF. Na tabela 2 se pode visualizar as medidas citadas [WEBER 2003].

Tabela 2. Medidas de confiabilidade[WEBER 2003].

Taxa de defeitos	*failure rate*, *hazard function*, *hazard rate*	número esperado de defeitos em um período de tempo
MTTF	*mean time to failure*	tempo até a primeira ocorrência do defeito
MTTR	*mean time to repair*	tempo médio para reparo do sistema
MTBF	*mean time between failure*	tempo médio entre falhas

As principais técnicas existentes para alcançar a dependabilidade são: *i*) prevenção de falhas, *ii*) tolerância a falhas, *iii*) validação e *iv*) previsão de falhas.

2.4. Alta Disponibilidade

Um sistema de alta disponibilidade tem como principal função estar disponível o maior tempo possível. Em outras palavras, o sistema deve estar disponível mesmo na presença de falhas [PEREIRA 2005].

Para atingir essa característica, são utilizadas soluções tanto em nível de hardware quanto em nível de software. Geralmente para que um sistema tenha alta disponibilidade, são utilizadas técnicas de hardware e de software em conjunto. No caso de alta disponibilidade em hardware, uma das alternativas é baseada na redundância de máquinas, redundância de links e espelhamento dos dados. A alta disponibilidade através de software é obtida por meio de controle de serviços e monitoramento [FILHO 2002].

A disponibilidade de um sistema pode ser calculada por meio da equação 1 [WIKIPÉDIA 2020b].

$$Disponibilidade = 100 * \frac{HorasAtivo - HorasInativo}{HorasAtivo} \quad (1)$$

Na Tabela 3 se pode visualizar o tempo *downtime* em anos ou meses de acordo com o percentual de disponibilidade pretendido ou apresentado.

Tabela 3. Tempos de disponibilidade [WIKIPÉDIA 2020b].

Disponibilidade (%)	*Downtime*/ano	*Downtime*/mês
97%	10 dias 22:48:00	0 dias 21:36:00
98%	7 dias 7:12:00	0 dias 14:24:00
99%	3 dias 15:36:00	0 dias 7:12:00
99,9%	0 dias 8:45:35.99	0 dias 0:43:11.99
99,99%	0 dias 0:52:33.60	0 dias 0:04:19.20
99,999%	0 dias 0:05:15.36	0 dias 0:00:25.92

Como se pode observar na tabela acima, quanto maior a disponibilidade menor é o tempo que um sistema pode ficar indisponível. Para alcançar uma alta disponibilidade (maior que 99,99%) maiores recursos de redundância devem ser empregados para que o tempo entre a falha e sua correção sejam diminuídos ao máximo. Essas soluções acabam por incrementar substancialmente os custos operacionais envolvidos.

Uma forma de obter a alta disponibilidade por meio de software, consiste em utilizar vários nodos, e replicar o seu sistema de arquivos. Dessa forma, caso um nodo apresente falha, os outros podem assumir o provimento do serviço. Essa abordagem também possibilita aumentar a capacidade de processamento por meio de distribuição de carga entre os nodos. Caso um nodo principal apresente falha, um nodo secundário deve assumir o seu papel.

Um software bastante utilizado para realizar a replicação do sistema de arquivo com a finalidade de alcançar a alta disponibilidade é o DRBD. A seguir são apresentados alguns aspectos sobre o DRBD que são relevantes para o desenvolvimento deste trabalho.

2.5. Estudo de Caso: DRBD

O DRBD é um software *open source* que possibilita a criação de um sistema de arquivos distribuído que pode ser comparado com um RAID nível 1, porém sobre uma rede TCP/IP. Para funcionar, o software necessita de dois dispositivos de armazenamento iguais (partição em disco): um configurado como primário e outro como secundário. Após essa configuração inicial, todo o conteúdo do disco primário é replicado no disco secundário e, posteriormente, apenas as alterações realizadas no disco primário são transferidas para o disco secundário, evitando a necessidade de copiar todo o conteúdo do disco primário novamente [WIKIPÉDIA 2020a].

A sua configuração é realizada no diretório $/etc/drdb.d/$. Nesta pasta encontram-se configurações globais, comuns e de recursos. Recomenda-se que as configurações de recursos sejam colocadas em arquivos externos com a extensão "res" e importadas no arquivo principal.

O software possui três tipos diferentes de protocolos que se diferenciam pela forma como consideram o término da sincronização. A seguir são apresentadas as características de cada um. Protocolo A (assíncrono): considera como término de escrita quando a mesma encontra-se persistida no disco local e enviada pela rede. Protocolo B (semi-assíncrono): considera como término de escrita quando a mesma encontra-se persistida no disco local e confirmado a recepção no disco remoto. Protocolo C (síncrono): considera como término de escrita quando a mesma encontra-se persistida no disco local e remoto.

As configurações mais importantes são as que se encontram na seção net, e se referem aos ajustes de configuração da interface de rede.

- *sndbuf-size*: tamanho do *buffer* de envio do *socket* TCP.
- *rcvbuf-size*: tamanho do *buffer* de recebimento do *socket* TCP.
- *timeout*: tempo aguardado para abandonar a conexão TCP após a detecção da perda de comunicação com o nodo secundário.
- *connect-int*: caso não seja possível conectar ao dispositivo DRBD remoto imediatamente, o DRBD continua tentando se conectar. Com esta opção, você pode definir o tempo entre duas tentativas.

- *ping-int*: se a conexão TCP/IP ligando um par de dispositivos DRBD ficar ociosa por mais de um segundo, o DRBD irá gerar um pacote *keep-alive* para verificar se seu parceiro ainda está ativo.
- *ping-timeout*: o tempo que um nodo tem para responder a um pacote keep-alive.
- *max-buffers*: limite de páginas de memória usada pelo nodo secundário do DRBD no recebimento.
- *ko-count*: caso o nó secundário não consiga concluir uma única solicitação de gravação dentro do *timeout*, ele é expulso do *cluster*.
- *max-epoch-size*: o maior número de blocos de dados entre duas barreiras de escrita.

As estatísticas de funcionamento do DRBD podem ser encontradas no arquivo $/proc/drbd$.

3. Trabalhos Relacionados

No trabalho de [ANTONI et al. 2014], os autores apresentam o uso de alta disponibilidade para segurança da informação em aplicações Web. Apresentou-se um novo modelo de alta disponibilidade usando o servidor Web Ngnix previamente configurado como *proxy* reverso para distribuir a carga entre os dois nodos configurados com o DRBD (replicação do sistema de arquivos do Linux). Também implementou-se a replicação de um banco de dados relacional do tipo *master/master*, mas utilizando o conceito de endereço IP compartilhado entre os modos.

[CHAVES 2014] apresenta um modelo para configurar dinamicamente o parâmetro de *timeout* do DRBD, baseado na média ARIMA. O modelo proposto utiliza com variáveis a latência da rede em uso no momento e possibilita ajustes mais precisos e reais da aplicação em questão. Neste trabalho ajusta-se o parâmetro *timeout* de acordo com atraso na rede no intuito de otimizar a detecção de falhas de nodos de replicação, resultando na diminuição de falsos positivos para nodos indisponíveis.

Em 2019, [TOMAZONI and FIORIN 2017], realizaram um estudo comparativo de ferramentas de espelhamento de dados para prover alta disponibilidade. No trabalho comparou-se os softwares livres DRBD e RSync. O comparativo foi realizado considerando os seguintes indicadores: tempo de execução, uso de CPU, uso de memória e tráfego de rede. Ao final, observou-se que tanto o DRBD como o Rsync utilizam a capacidade máxima da rede para transmitir os dados. Convêm mencionar que o Rsync não realiza a replicação dos dados em tempo real, devendo ser executado por meio de uma tarefa agendada. Desta forma, apenas o DRBD é indicado para uso em alta disponibilidade em tempo real.

Como se observou, o DRBD é uma solução bastante utilizada para pesquisas práticas na área de tolerância a falhas e alta disponibilidade, por ser um recurso bastante flexível, permitindo implantar uma diversidade de serviços sobre a sua plataforma tolerante a falhas. A detecção de falhas da replicação é fundamental pois o software trabalha com replicação em tempo real, um cenário fundamental para construir a nuvem.

4. Metodologia

Este trabalho se propõe a ajustar de forma dinâmica os parâmetros de detecção de nodos *offline* do software DRBD. Para tanto, é proposto o uso da Inteligência Artificial para monitorar a latência e banda disponível entre nodo primário e secundário.

A partir dessas informações, uma Inteligência Artificial será treinada para configurar dinamicamente o parâmetro *timeout-ping* do DRBD. Este parâmetro é utilizado pelo software como um *timeout* a fim de detectar a indisponibilidade de um nodo. Caso a resposta do nodo não retorne antes do tempo configurado, o mesmo é considerado indisponível.

Como já mencionado anteriormente, em um cenário dinâmico, como é o caso da Internet, ambiente onde acontece a computação nuvem, o mesmo não pode ser considerado como fixo. Este ajuste visa obter melhores resultados sobre a abordagem tradicional adotada pelo *software*, onde o parâmetro é constante. A Figura 3 apresenta um fluxograma da metodologia proposta.

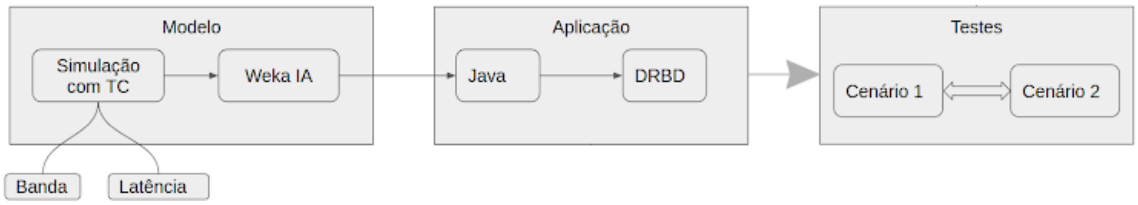

Figura 3. Metodologia do trabalho.

Os testes foram realizados em dois cenários distintos. O cenário 1 é a tradicional implementação da replicação do DRBD, onde o parâmetro de *timeout* tem um valor fixo. Este cenário foi utilizado como controle. No cenário 2, o valor é ajustado de forma dinâmica pela inteligência artificial. A Figura 4 ilustra os dois cenários propostos para realizar a comparação e avaliar a proposta deste trabalho.

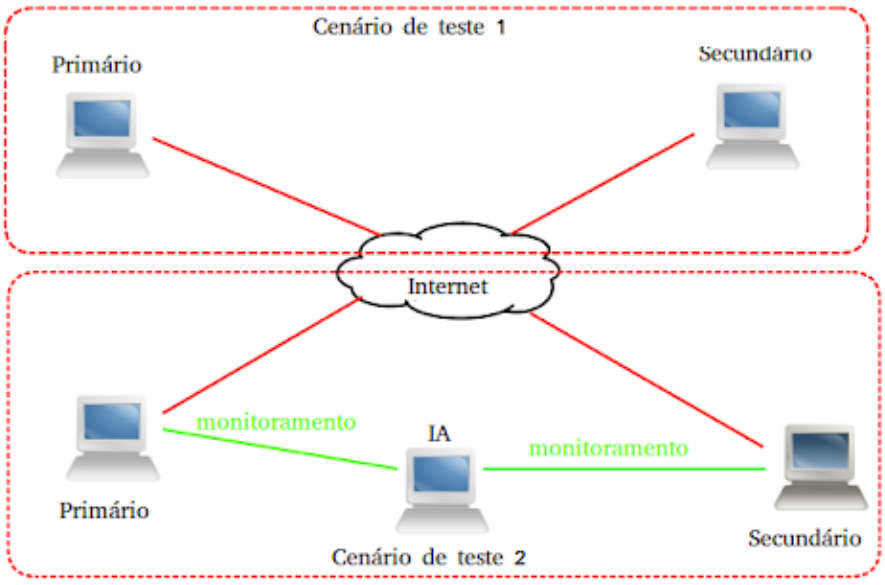

Figura 4. Cenário de testes.

Para realizar a comparação dos dois cenários propostos acima, foram utilizadas as estáticas de funcionamento do DRDB, conforme Figura 5.

```
[root@nzhost1 ~]# cat /proc/drbd
version: 8.2.6 (api:88/proto:86-88)
GIT-hash: 3e69822d3bb4920a8c1bfdf7d647169eba7d2eb4 build by root@nps22094, 2009-06-09
16:25:53
 0: cs:Connected st:Primary/Secondary ds:UpToDate/UpToDate C r---
    ns:15068 nr:1032 dw:16100 dr:3529 al:22 bm:37 lo:0 pe:0 ua:0 ap:0 oos:0
 1: cs:Connected st:Primary/Secondary ds:UpToDate/UpToDate C r---
    ns:66084648 nr:130552 dw:66215200 dr:3052965 al:23975 bm:650 lo:0 pe:0 ua:0 ap:0 oos:0
```

Figura 5. Estatísticas do DRDBs.

As estatísticas devem ser interpretadas de acordo com a seguinte nomenclatura: *i*) ns: dados enviados na rede; *ii*) nr: dados recebidos na rede; *iii*) dw: dados gravados no disco rígido; dr: dados líquidos do disco rígido; *iv*) al: número de atualizações dos metadados; *v*) bm: número de atualizações da área de *bitmap* dos metadados; *vi*) lo: solicitações abertas para o subsistema de E/S; *vii*) pe: pedidos enviados ao parceiro, mas que ainda não foram respondidos; *viii*) ua: solicitações recebidas pelo parceiro, mas que ainda não foram respondidas; *ix*) ap: solicitações de E/S de bloco encaminhadas para o DRBD, mas ainda não respondidas; *x*) ep: número de objetos de época; *xi*) wo: método de ordem de escrita usado; *xii*) oos: quantidade de armazenamento atualmente fora de sincronia.

Os experimentos foram realizados com um *dataset* composto por diferentes tipos de arquivos tradicionalmente utilizados na nuvem, como por exemplo, arquivos de imagem, arquivos de áudio, vídeos, documentos de textos e arquivos em formato PDF. Os mesmos arquivos foram copiados para uma partição replicada de 500mb. A Figura 6 apresenta algumas informações sobre os arquivos utilizados, como seus nomes, data de criação e respectivos tamanhos.

```
-rw-r--r-- 1 glaucio glaucio 220083383 dez  6 06:14 177-hd.mp4
-rw-r--r-- 1 glaucio glaucio    645700 nov 17 11:20 banner_topo_Inscrições.jpg
-rw-r--r-- 1 glaucio glaucio   5289384 jul  9 13:59 file_example_MP3_5MG.mp3
-rw-r--r-- 1 glaucio glaucio   1250515 out 27  2016 manual-latex-1.pdf
-rw-r--r-- 1 glaucio glaucio     23616 dez 10 11:44 wget-log
-rw-r--r-- 1 glaucio glaucio      1164 dez 10 11:44 wget-log.1
-rw-r--r-- 1 glaucio glaucio       966 dez 10 11:47 wget-log.2
```

Figura 6. *Dataset* de testes.

5. Resultados e discussão

A primeira etapa do desenvolvimento deste trabalho consistiu no treinamento da IA para que fosse possível apreender a determinar o RTT da rede de acordo com a largura de banda (100kbit até 1gbit) e latência (10ms até 450ms). Para isso foi desenvolvido um *script* que foi alimentando com parâmetros obtidos a partir de uma simulação realizada com o módulo NeTem[13] do Linux. O *script* desenvolvido para medir o RTT pode ser observado na Figura 7.

[13]https://wiki.linuxfoundation.org/networking/netem

```
#!/bin/bash

DEV=eth0;
IP="127.0.0.1";
CONT=50;
PACOTE=512;#MINUS 8 - HEADER SIZE
PERDA=0.5%;
DUPLICADOS=0.5%;
CORROMPIDOS=0.5%;
PERC JITTER=0.1; #Percentual de Jitter
RATES="100kbit 250kbit 500kbit 750kbit 1mbit 2.5mbit 5mbit 7.5mbit 10mbit 25mbit 50mbit 75mbit \
       100mbit 250mbit 500mbit 750mbit 1gbit";
LATENCIES="10 30 50 70 90 110 130 150 170 190 210 230 250 270 290 310 330 350 370 390 410 430 450";

for RATE in ${RATES} ; do
  for LATENCY in ${LATENCIES}; do
    JITTER="$(echo "(${LATENCY}*${PERC JITTER})" | bc)";
    tc qdisc delete dev ${DEV} root netem;
    echo "Simulando Rate:" ${RATE} " Latency:" ${LATENCY} " Jitter:" ${JITTER};

    tc qdisc add dev ${DEV} root netem loss ${PERDA} duplicate ${DUPLICADOS} corrupt ${CORROMPIDOS} \
      rate ${RATE} delay ${LATENCY}ms ${JITTER}ms distribution normal;

    PING="$(ping -q -s ${PACOTE} -c ${CONT} ${IP} | tail -1 )"; #RTT
    echo ${PING};
  done
done
```

Figura 7. *Script* para medir o RTT.

Na Figura 8 se pode visualizar a progressão do RTT de acordo com a latência no eixo x. Cada série do gráfico representa uma largura de banda diferente.

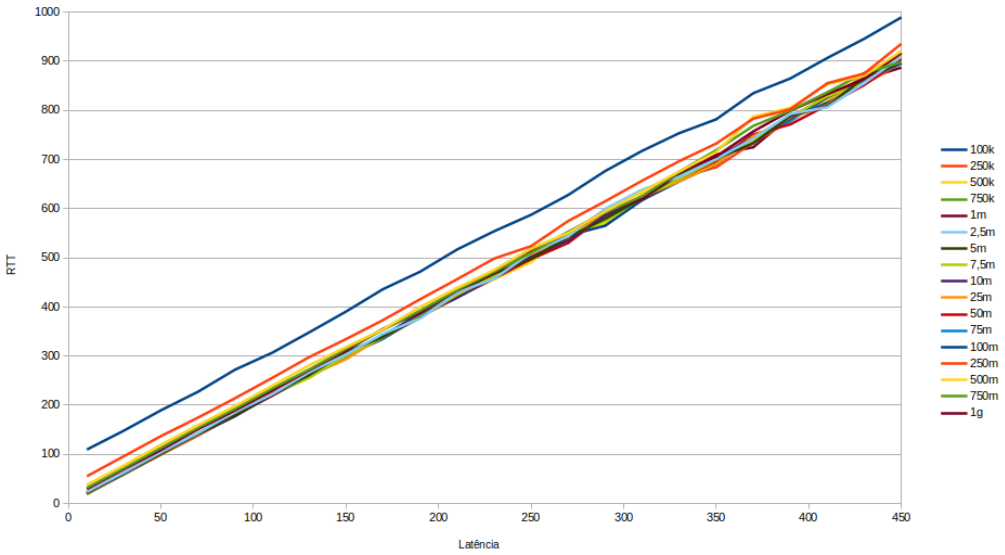

Figura 8. RTT medido para treinamento da IA.

O eixo x representa a latência da rede, enquanto o eixo y representa o RTT mesurado. Observa-se a influência no aumento do RTT nos casos onde a largura de banda do link é menor.

Após a obtenção dos dados, usou-se o Weka[14] para treinar a IA com duas diferentes topologias. A primeira consiste em usar uma rede neural com entrada de números reais e com mais de uma camada oculta, a segunda representa cada uma das combinações de entrada como classes e possui apenas uma camada oculta. As Figuras 9 e 10 representam respectivamente as topologias acima mencionadas.

[14]https://www.cs.waikato.ac.nz/ml/weka/

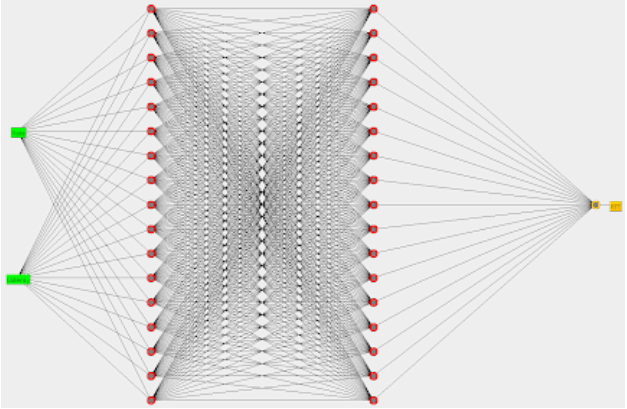

Figura 9. IA com números.

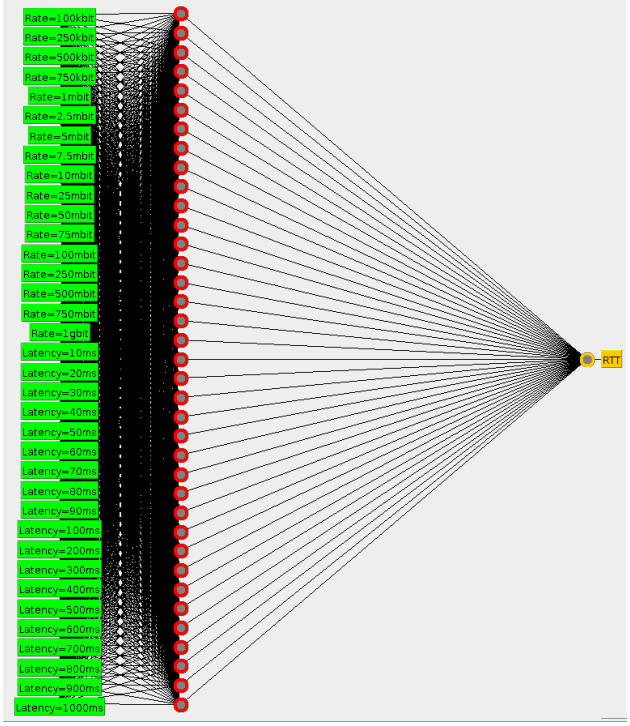

Figura 10. IA com classes.

Após o treinamento constatou-se por meio da métrica RMSE (*Root Mean Square Error*) os erros apresentados por cada uma das topologias propostas, quanto menor o seu valor mais aproximado dos valores originais. A rede com a topologia de classes apresentou melhores resultados na predição dos valores para o RTT, ficando com RMSE de 0,4; a rede a com números reais obteve RMSE de 5,68. Durante os treinamentos da IA também foi realizado um teste com Regressão Linear, ficando o RMSE em 5,17. Outro ponto levado em consideração para utilizar o modelo baseado em IA é o fato de ser uma abordagem mais flexível e adaptável, num futuro onde se poderia acrescentar novos elementos ao modelo, bastaria apenas realizar um novo treinamento. Construiu-se um programa em Java para calcular o *timeout* com base no modelo de rede neural obtido com o Weka. O mesmo é executado durante o *script* de testes da Figura 11.

```
#! /bin/bash

PASS="1234\n";
PERC_JITTER=0.1; #Percentual de Jitter
LATENCIES="10 100 300 500 800";
RATES="100mbit";

function stats { #obtem as estatisticas
  while [ true ]; do #aguardar termino sync no server
    LIST1="$(cat /proc/drbd)";
    if grep -q "bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0" <<< "${LIST1}"; then
      echo ${LIST1};
      break;
    fi
    sleep 1s;
  done

  #echo "Wait slave...";
  while [ true ]; do #aguardar termino sync no slave
    LIST2="$(echo -e ${PASS} | sshpass -p 1234 ssh root@vm2c 'cat /proc/drbd')";
    if grep -q "bm:0 lo:0 pe:0 ua:0 ap:0 ep:1 wo:f oos:0" <<< "${LIST2}"; then
      echo ${LIST2};
      break;
    fi
    sleep 1s;
  done
}

function clean { #limpa a replicação
  echo -e ${PASS} | sudo -S tc qdisc delete dev eth1 root netem; #limpa rede
  rm -fr /pasta/dataset/; #apaga o dataset
  PID=$!;
  wait ${PID};
  sleep 10s;
}

function IA {
  RTT="$(java -jar DRBD_IA.jar -r=$1 -l=$2)";
  #Define o tempo limite para respostas a pacotes keep-alive.
  #O valor padrão é 0,5 segundos, com um mínimo de 0,1 segundos e um máximo de 3 segundos. A unidade é décimos de segundo.
  RTT="$(echo "(${RTT}/100)" | bc)";
  if [ ${RTT} -gt 30 ]; then
    RTT=30;#3000ms
  elif [ ${RTT} -lt 1 ]; then
    RTT=1;#100ms
  fi
  echo "IA RTT: "${RTT}"00";
  echo -e ${PASS} | sudo -S sed -i -e '/ping-timeout/ s/ .*;/ '${RTT}';/' /etc/drbd.d/global_common.conf;
  echo -e ${PASS} | sudo -S sshpass -p 1234 scp -p /etc/drbd.d/global_common.conf root@vm2c:/etc/drbd.d/global_common$
  echo -e ${PASS} | sudo -S /etc/init.d/drbd reload;
  echo -e ${PASS} | sudo -S sshpass -p 1234 ssh -t root@vm2c '/etc/init.d/drbd reload';
  sleep 10s;
}

for RATE in ${RATES}; do
  for LATENCY in ${LATENCIES}; do
  JITTER="$(echo "(${LATENCY}*${PERC_JITTER})" | bc)";
  echo "-------------------------------------";
  clean;
  echo "Rate:"${RATE}" Delay:"${LATENCY}" Jitter:"${JITTER};

  # loss 0.3% duplicate 0.1% corrupt 0.1%
  echo -e ${PASS} | sudo -S tc qdisc add dev eth1 root netem rate ${RATE} delay ${LATENCY}ms ${JITTER}ms distribution normal;

  PING="$(ping -q -s 512 -c 3 vm2 | tail -1 )";
  echo ${PING};
  PING="$(echo ${PING} | awk '{print $4}' | cut -d '/' -f 2)";

  #IA ${RATE} ${PING};
  echo "NO-IA: 300";

  stats;
  #echo "Copying..";
  INICIO="$(date +%s%N)";
  echo -e ${PASS} | sudo -S dmesg -C; #limpa buffer de erros do kernel
  cp -r /home/glaucio/dataset.zip /pasta/; #copia o dataset para o DRBD
  #PID=$!;
  #wait ${PID};
  sleep 30s;
  echo -e ${PASS} | sudo -S dmesg; #mostra os erros
  stats;
  FIM="$(date +%s%N)";
  TEMPO="$(echo "(${FIM} - ${INICIO})" | bc)";
  echo ${TEMPO};

  done
done
```

Figura 11. *Script* de testes.

Os testes foram realizados para um link de 100mbits. Optou-se por esta taxa de transmissão por ser o valor do padrão *Fast Ethernet* (IEEE 802.3u) presente na maioria dos computadores. Os dois cenários (RTT fixo e RTT ajustado com IA) foram analisados por meio da contagem de *bits* enviados pelo nodo principal (*send*), *bits* recebidos pelo nodo secundário (*receive*) e *bits* gravados em disco no nodo secundário (*write*). Caso os três valores sejam idênticos, não ocorreram perdas de pacotes, portanto não houve erros.

A Tabela 4 apresenta os resultados para cinco diferentes testes de replicação com a variação da latência de 10ms até 800ms e *ping-timeout* com valor fixo de 300ms. O valor fixo da latência foi escolhido com base na mediana dos valores de latência testados. Nesta situação, 50% dos elementos são menores que o valor e 50% são maiores.

Tabela 4. Testes Cenário 1 (Controle).

latência	*jitter*	mín	ping	máx	RTT	*send*	*receive*	*write*	erros	seg.
10	1	10,0	10,6	11,7	300	222088	222088	222088	0	24
100	10	79,4	94,5	117,0	300	222072	222072	222072	0	25
300	30	270,9	292,7	333,2	300	222072	222072	222072	0	57
500	50	437,6	486,6	543,4	300	222136	222136	222136	0	170
800	100	720,9	817,1	959,8	300	224064	223948	223948	116	4640

Observa-se que ocorreram perdas de pacotes onde a latência foi muito maior que o *ping-timeout*, verifica-se que foram enviados mais bits do que os recebidos. Tal fato indica que houve encerramento do *socket* TCP durante a replicação, ocorrendo a perda de pacotes por meio do não recebimento em tempo do pacote *keep-alive*.

No caso onde a IA atuou medindo a latência do *link* e determinando o valor do RTT, com base no modelo construído de IA, o parâmetro *ping-timeout* foi ajustado dinamicamente. Nesta situação não ocorreram perdas de pacotes TCP durante a replicação dos dados. A Tabela 5 apresenta os resultados dos testes no cenário 2 (IA).

Tabela 5. Testes Cenário 2 (IA).

latência	*jitter*	min	ping	max	RTT	*send*	*receive*	*write*	erros	seg.
10	1	9,8	10,9	11,7	100	222092	222092	222092	0	24
100	10	79,1	96,0	112,1	100	222072	222072	222072	0	29
300	30	294,2	307,9	316,9	600	222136	222136	222136	0	116
500	50	464,4	506,1	559,0	900	222136	222136	222136	0	191
800	100	692,7	820,6	919,4	900	222096	222096	222096	0	3803

Para facilitar a compreensão dos resultados, foram elaborados gráficos para demonstrar os RTTs e erros nos dois cenários de testes. Estes gráficos são apresentados nas Figuras 12 e 13.

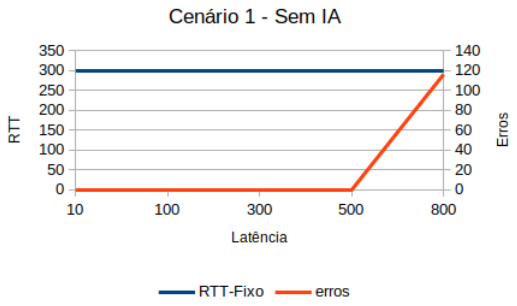

Figura 12. Erros no cenário 1.

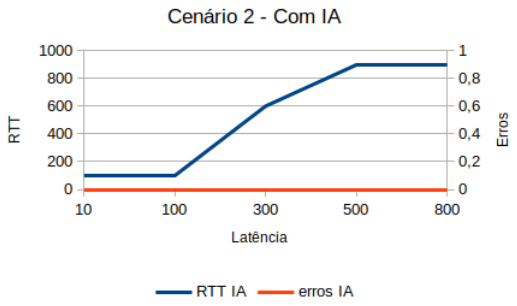

Figura 13. Erro no cenário 2.

Novamente se observa a inexistência de erros no cenário com a IA atuando. Os erros ocorridos acabam por interferir no tempo de replicação, pois demandam que os blocos com problemas sejam reenviados pelo DRBD.

A seguir são apresentadas as conclusões deste trabalho, bem como possibilidades de trabalhos futuros.

6. Conclusão e trabalhos futuros

Este trabalho apresentou o emprego da inteligência artificial para configurar de forma dinâmica o software de alta disponibilidade DRBD. A abordagem proposta possibilita o ajuste específico do parâmetro de *timeout* do *socket* TCP usado na replicação dos dados em um ambiente de computação em nuvem. A inteligencia artificial foi treinada para configurar o RTT de acordo com a latência e largura de banda do *link*.

Após os experimentos com um *link* de 100mbits, onde se comparou o parâmetro configurado de forma estática (controle) com o dinâmico ajustado pela IA, se observou que em cenários de nuvem, onde a latência do *link* de replicação é bastante grande (800ms), a abordagem proposta neste trabalho apresentou um menor número de erros e, no caso do cenário de controle, o tempo de execução foi maior, pois devido aos erros houve necessidade de retransmissão de pacotes.

Assim, conclui-se que o objetivo deste trabalho foi atingido de forma totalmente satisfatória, pois a abordagem proposta apresentou resultados esperados nos testes realizados. A partir disso, é possível afirmar que a configuração dinâmica do parâmetro de *timeout* através do uso de IA pode tornar o detector de falhas do DRBD mais eficiente na implementação de sistemas de alta disponibilidade.

Como sugestão de trabalhos futuros é possível realizar testes com link de outras larguras de banda (10mbit, 500mbit, 1gbit) e com tamanhos maiores de *dataset* e partição de replicação.

Referências

[Abiodun et al. 2018] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Mohamed, N. A., e Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11):e00938.

[ANTONI et al. 2014] ANTONI, M., VIVIAN, G. R., e PREUSS, E. (2014). Segurança da informação: Proposta de arquitetura de alta disponibilidade para aplicações web. *Anais do EATI - Encontro Anual de Tecnologia da Informação e Semana Acadêmica de Tecnologia da Informação.*

[BARRETO 2001] BARRETO, J. M. (2001). *Inteligência Artificial no Limiar do Século XXI - Abordagem Híbrida, Simbólica, Conexionista e Evolucionária*. UFSC.

[CHAVES 2014] CHAVES, A. D. (2014). Utilizando o modelo arima para configuração dinâmica dos parâmetros de timeout do drbd. Monografia. Universidade Regional Integrada do Alto Uruguai e das Missões. Santiago.

[FILHO 2002] FILHO, N. A. P. (2002). Linux, clusters e alta disponibilidade. Dissertação de Mestrado — Universidade de São Paulo.

[NUNES 2020] NUNES, C. R. (2020). Tolerância a falhas em sistemas distribuídos. Disponível em: https://www.ppgia.pucpr.br/~alcides/Teaching/SistemasDistribuidos/TOF/01_conceitosbasicos.pdf. Acessado em: 13/04/2020.

[PEREIRA 2005] PEREIRA, R. (2005). Alta disponibilidade em sistemas gnu/linux utilizando as ferramentas drdb, heartbeat e mon. *Monografia apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras. Cap*, 4:13–21.

[PEREIRA 2020] PEREIRA, S. L. (2020). Introdução à inteligência artificial. Disponível em: https://www.ime.usp.br/~slago/IA-introducao.pdf. Acessado em: 12/04/2020.

[QINETWORK 2020] QINETWORK (2020). Saas, paas e iaas – os serviços de computação em nuvem. Disponível em: https://www.qinetwork.com.br/saas-paas-iaas-os-servicos-de-computacao-em-nuvem/. Acessado em: 03/04/2020.

[REISNER 2020] REISNER, P. (2020). Drbd.conf – o arquivo de configuração dos dispositivos drbd. Disponível em: http://www.drbd.org/users-guide/re-drbdconf.html. Acessado em: 04/04/2020.

[SOFTLINE 2020] SOFTLINE (2020). Iaas, paas e saas: entenda os modelos de nuvem e suas finalidades. Disponível em: https://brasil.softlinegroup.com/sobre-a-empresa/blog/iaas-paas-saas-nuvem. Acessado em 03/04/2020.

[Srivastava and Khan 2018] Srivastava, P. e Khan, R. (2018). A review paper on cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 8(6):17–20.

[TECMUNDO 2020] TECMUNDO (2020). O que é computação em nuvens? Disponível em: https://www.tecmundo.com.br/computacao-em-nuvem/738-o-que-e-computacao-em-nuvens-.htm. Acessado em: 03/04/2020.

[TOMAZONI and FIORIN 2017] TOMAZONI, T. e FIORIN, A. (2017). Estudo comparativo de ferramentas de espelhamento de dados para alta disponibilidade com software livre. Monografia. Instituto Federal Farroupilha, Campus de Frederico Westphalen.

[WEBER et al. 1990] WEBER, T., Jansch-Pôrto, I., e WEBER, R. (1990). Fundamentos de tolerância a falhas.

[WEBER 2003] WEBER, T. S. (2003). Tolerância a falhas: conceitos e exemplos. Apostila do Programa de Pós-Graduação–Instituto de Informática.

[WIKIPÉDIA 2020a] WIKIPÉDIA (2020a). Drbd. Disponível em: https://pt.wikipedia.org/wiki/DRBD. Acessado em: 04/04/2020.

[WIKIPÉDIA 2020b] WIKIPÉDIA (2020b). Sistema de alta disponibilidade. Disponível em: https://pt.wikipedia.org/wiki/Sistema_de_alta_disponibilidade. Acessado em: 04/04/2020.