

UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

KAROLINE KIMIKO FIGUEIREDO SETOUE

**APLICAÇÃO DE REDES NEURAIS ARTIFICIAIS NO
BALANCEAMENTO DE CARGA EM SERVIÇOS DE NUVEM**

BAURU
2017

KAROLINE KIMIKO FIGUEIREDO SETOUE

**APLICAÇÃO DE REDES NEURAIS ARTIFICIAIS NO
BALANCEAMENTO DE CARGA EM SERVIÇOS DE NUVEM**

Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação apresentado ao Departamento de Computação da Faculdade de Ciências da Universidade Estadual Paulista “Júlio de Mesquita Filho” – UNESP, Câmpus de Bauru.

Orientador: Prof. Dr. Kelton Augusto Pontara da Costa

BAURU
2017

Setoue, Karoline Kimiko Figueiredo.

Aplicação de redes neurais artificiais no
balanceamento de carga em serviços de nuvem /
Karoline Kimiko Figueiredo Setoue, 2017
45 p. : il.

Orientador: Kelton Augusto Pontara da Costa

Monografia (Graduação)-Universidade Estadual
Paulista. Faculdade de Ciências, Bauru, 2017

1. Balanceamento de carga. 2. Redes neurais
artificiais. 3. Computação em nuvem I. Universidade
Estadual Paulista. Faculdade de Ciências. II.
Título.

Karoline Kimiko Figueiredo Setoue

Aplicação de Redes Neurais Artificiais no Balanceamento de Carga em Serviços de Nuvem

Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação apresentado ao Departamento de Computação da Faculdade de Ciências da Universidade Estadual Paulista "Júlio de Mesquita Filho" – UNESP, Câmpus de Bauru.

Banca Examinadora

Prof. Dr. Kelton Augusto Pontara da Costa

Orientador

Universidade Estadual Paulista "Júlio de Mesquita Filho"

Departamento de computação

Faculdade de Ciências

Profa. Dra. Simone das Graças Domingues Prado

Universidade Estadual Paulista "Júlio de Mesquita Filho"

Departamento de computação

Faculdade de Ciências

Profa. Dra. Roberta Spolon

Universidade Estadual Paulista "Júlio de Mesquita Filho"

Departamento de computação

Faculdade de Ciências

Bauru, 07 de Fevereiro de 2017.

AGRADECIMENTOS

Agradeço a todos os familiares por todo o incentivo e apoio. Especialmente às minhas irmãs, por estarem ao meu lado ao longo desta jornada.

Aos meus amigos, por todo apoio, incentivo, companheirismo e por me mostrarem que não é preciso estar perto para estar presente. Sou grata por todos os momentos que compartilhamos.

Aos meus professores da Unesp, por todo o conhecimento compartilhado, especialmente ao Prof. Dr. Kelton, pela orientação ao longo deste projeto.

Ao LTIA e ao Prof. Dr. Eduardo Morgado por me proporcionar oportunidades tão boas em um ambiente com tantas pessoas excepcionais que levarei comigo ao longo da vida.

"Mundo mundo vasto mundo..."
(Carlos Drummond de Andrade)

RESUMO

O conceito de computação em nuvem tem se tornado popular nos últimos anos. Seu objetivo é prover serviços de acesso às aplicações e arquivos por meio da internet de maneira flexível e simples. Neste contexto, explorar maneiras de garantir este acesso de maneira eficiente torna-se uma tarefa importante na área de computação. Tendo isto em vista, o presente trabalho apresenta um protótipo de aplicação de balanceamento de cargas em uma arquitetura de computação em nuvem utilizando Redes Neurais Artificiais como técnica para distribuição das cargas de requisições. A construção da aplicação consiste na incorporação da rede neural no monitoramento e ajuste dos pesos em cada servidor do sistema em nuvem para os quais as requisições serão direcionadas.

Palavras-chave: Redes Neurais Artificiais. Computação em Nuvem. Balanceamento de Carga.

ABSTRACT

The concept of cloud computing has become popular in recent years. Its goal is to provide services that allows access to applications and files through the Internet in a flexible and simple way. In this context, exploring ways to secure this access efficiently becomes an important task in Computer Science. Therefore, the present work presents a load balancing application prototype in a cloud computing architecture using Artificial Neural Networks as a technique for load distribution of requests. The application consists in a neural network applied to monitoring and adjustment of the weights for each server to which requests are directed.

Keywords: Artificial Neural Networks. Cloud Computing. Load Balancing.

LISTA DE FIGURAS

Figura 1 – Modelos de Computação em Nuvem	16
Figura 2 – Estrutura Simplificada de Balanceamento de Carga	18
Figura 3 – Estrutura de uma ANN	22
Figura 4 – Fluxo de aprendizado de uma rede neural	22
Figura 5 – Fluxo de sinal em um Perceptron	23
Figura 6 – Estrutura de um Perceptron Multicamada	24
Figura 7 – Estrutura de uma Rede Neural com função de Base Radial	25
Figura 8 – EC2	27
Figura 9 – Código exemplo de servidor em nodejs	28
Figura 10 – Exemplo de execução de código em nodejs	28
Figura 11 – Arquitetura simplificada do projeto	29
Figura 12 – Conexão ao servidor de Requisições	31
Figura 13 – Round Robin	32
Figura 14 – Camadas implementadas na rede neural	35
Figura 15 – <i>Screenshot</i> de teste realizado com a Rede Neural	38
Figura 16 – <i>Screenshot</i> Dados do Balanceador - Servidores Ativos	38
Figura 17 – <i>Screenshot</i> Dados do Balanceador - Pesos mais recentes (RNA)	38
Figura 18 – <i>Screenshot</i> Aplicação para testar o balanceador	39
Figura 19 – <i>Screenshot</i> Teste com RNA	40
Figura 20 – <i>Screenshot</i> Teste com RNA já treinada	40
Figura 21 – <i>Screenshot</i> Teste com Round Robin	41

LISTA DE TABELAS

Tabela 1 – Servidores	27
Tabela 2 – Características	34
Tabela 3 – Comparação entre Distribuição de requisições entre servidores	40

LISTA DE ABREVIATURAS E SIGLAS

ANN *Artificial Neural Networks*

AWS *Amazon Web Services*

RNA *Rede Neural Artificial*

SUMÁRIO

1	INTRODUÇÃO	12
1.1	Objetivos	14
1.1.1	Objetivos Gerais	14
1.1.2	Objetivos Específicos	14
1.2	Organização da Monografia	14
2	FUNDAMENTAÇÃO TEÓRICA	15
2.1	Computação em Nuvem	15
2.1.1	Modelos de Computação em Nuvem e Tipos	16
2.2	Balanceamento de Carga	18
2.3	Redes Nerais Artificiais	21
2.3.1	Perceptron	23
2.3.2	ANNs com Múltiplas Camadas	24
3	METODOLOGIA	26
3.1	Tecnologias e Ferramentas Utilizadas	26
3.1.1	AWS - Amazon Web Services	26
3.1.2	Node.js e npm	27
3.2	Métodos e Etapas	29
4	ARQUITETURA DO PROJETO	31
4.1	Arquitetura da Rede Neural	33
4.2	Implementação e Testes	36
5	RESULTADOS	39
6	CONCLUSÃO	42
6.1	Trabalhos futuros	42
	REFERÊNCIAS	43

1 INTRODUÇÃO

O conceito de computação em nuvem ou *Cloud Computing* tem se tornado popular nos últimos anos. Seu objetivo é prover serviços de acesso a aplicações e arquivos por meio da internet de maneira flexível e simples (NUAIMI N. MOHAMED, 2012). A Computação em Nuvem propõe uma arquitetura diferente da cliente-servidor, na qual os recursos são diversificados e encontram-se dispersos em *clusters* de sistemas distribuídos e servidores ao redor do mundo.

A disponibilidade e o acesso a estes recursos é transparente ao usuário final e adaptável de acordo com a demanda. Neste contexto, grande parte da preocupação se concentra em atribuir as tarefas para os nós da nuvem de forma eficiente. Isto é feito para que o esforço e processamento de requisições seja feito da melhor maneira possível (TALEB-BENDIAB, 2010).

Para isto, a qualidade dos serviços oferecidos de acordo com a demanda dependem, também, da arquitetura do serviço e de como é feita a virtualização dos servidores e sistemas distribuídos. Geralmente, há três diferentes tipos de serviços de computação em nuvem: armazenamento virtual, chamado *Infrastructure as a Service - IaaS*, tal como Amazon Web Services (SERVICES, 2016b); *Software as a Service - SaaS*, caracterizados por provedores de serviços na internet; e, por fim, ferramentas de desenvolvimento de *software* e produtos *Platform as a Service - PaaS* (SALLAMI; ALOUSI, 2013), como por exemplo a plataforma *SoftLayer* (IBM Cloud, 2016).

Além dessa, há ainda a caracterização pelo tipo de nuvem, que podem ser públicas, privadas ou híbridas. A primeira tem como característica maior eficiência e distribuição de recursos, como por exemplo *Amazon Web Services*. A segunda caracteriza-se por ser mais fechada, geralmente pertencente a uma organização específica que é responsável pelo serviço (ZHANG; CHENG; BOUTABA, 2010) e a última consiste em uma combinação entre as duas anteriores.

Como todas as abordagens e estruturas de serviços em nuvem precisam garantir a disponibilidade do serviço nela hospedado, há a necessidade de distribuir as requisições que chegam para os serviços de maneira eficiente, visto que a distribuição de tarefas e requisições feita de forma eficiente leva a melhora no tempo de resposta, economia de energia e diminui o risco de falhas.

Com base nesse requisito, o balanceamento de carga tem por objetivo receber as requisições e distribuí-las para os nós do serviço de nuvem. Isto pode ser feito de diversas maneiras e a classificação dos algoritmos que realizam o balanceamento de carga é dividida, geralmente, em dois tipos: algoritmos estáticos e algoritmos dinâmicos

(NUAIMI N. MOHAMED, 2012). Nas abordagens estáticas, as requisições são distribuídas de acordo com a capacidade do nó da nuvem de processá-la. Neste caso, é necessário ter conhecimento prévio da infraestrutura da nuvem e seus componentes. As abordagens dinâmicas, por sua vez, levam em consideração diferentes atributos dos nós, tais quais a largura de banda. Nesse caso, a distribuição é feita de forma dinâmica e requer constante monitoramento dos nós, o que geralmente torna esse tipo de balanceamento mais complexo de se implementar.

Dentre as técnicas empregadas neste contexto é possível citar o Round-Robin, considerado um algoritmo de simples implementação, *Least Connections* ou Menor Número de Conexões (NGINX, 2017a), *Ant Colony* e até mesmo abordagens que utilizam *Particle Swarm Optimization - PSO* (PANDEY et al., 2010). No entanto, sempre há maneiras de melhorar técnicas já existentes, assim como propor novas soluções.

Com isto em mente, técnicas de reconhecimento de padrões também podem ser viáveis neste contexto. Seu objetivo é aprender uma função que melhor separe amostras de classes diferentes no espaço de características (DUDA; HART; STORK, 2000). Esse processo de aprendizado e reconhecimento pode ser feito por meio de Redes Neurais Artificiais (RNA) - *Artificial Neural Networks (ANN)* (HAYKIN, 2007), cuja arquitetura baseia-se na estrutura de neurônios, que por meio de redes, são capazes de realizar as mais complexas atividades.

Tais redes podem ser representadas como grafos orientados, onde cada nó do grafo é um neurônio e o fluxo de informação é representado por suas arestas. Os dados de entrada geralmente constituem em um vetor cujos elementos são pesos para cada uma das características da amostra. A classificação das entradas com base no vetor de características é feito por meio de uma função de ativação sendo que cada tipo de RNA apresenta diferentes métodos de acordo com sua arquitetura.

Dentre os vários tipos de redes neurais existentes, pode-se citar as Redes Neurais de Base Radial (*Radial Basis Function - RBF*), Redes Neurais do tipo Perceptron Multicamadas (*Multilayer Perceptron - MLP*) Redes Neurais Probabilísticas (*Probabilistic Neural Networks - PNNs*) (SPECHT, 1990a) (SPECHT, 1992). Por possuírem uma arquitetura mais simplista, as RBFs tem sido objeto de estudo de diversos trabalhos, bem como as PNNs e suas versões melhoradas.

A aplicação de RNAs no contexto de balanceamento de cargas é ainda pouco explorado e pode apresentar bons resultados. Como o balanceamento de carga em sistemas de nuvem dependem, de certa forma, das características do serviço e/ou da requisição, o presente projeto propõe uma abordagem no contexto apresentado utilizando RNAs. Tendo estes conceitos em vista, o presente projeto tem por objetivo um protótipo de aplicação de RNAs no balanceamento de carga em sistemas em nuvem.

1.1 Objetivos

1.1.1 Objetivos Gerais

O objetivo geral deste projeto é planejar e desenvolver um módulo de aplicação que utiliza uma rede neural artificial no balanceamento de carga em serviços de nuvem, bem como analisar seu uso em ambientes reais.

1.1.2 Objetivos Específicos

- a) Estabelecer o estado da arte acerca de aplicação de redes neurais artificiais no contexto de balanceamento de carga em sistemas de nuvem;
- b) Estudar o funcionamento de serviços em nuvem;
- c) Identificar características e requisitos considerados no balanceamento de carga;
- d) Definir os elementos necessários para desenvolvimento da aplicação que fará o balanceamento;
- e) Definir tipo de RNA a ser utilizada;
- f) Planejar a estrutura da aplicação;
- g) Implementar a ferramenta inteligente;
- h) Analisar os resultados no contexto de balanceamento de carga;

1.2 Organização da Monografia

O presente trabalho divide-se em seções, sendo esta a primeira (Introdução). As demais seções estão dispostas na seguinte ordem:

- **Seção 2 - Fundamentação Teórica:** apresentação dos conceitos teóricos envolvidos no trabalho.
- **Seção 3 - Metodologia:** descrição do método de pesquisa e desenvolvimento, bem como as ferramentas utilizadas.
- **Seção 4 - Arquitetura do Projeto:** descrição da arquitetura do projeto e da abordagem utilizada na implementação e testes.
- **Seção 5 - Resultados:** apresentação dos resultados obtidos no trabalho.
- **Seção 6 - Conclusão:** considerações finais sobre o trabalho e possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção apresentam-se conceitos, fundamentos e teorias que dão suporte ao presente trabalho. Dentre eles, Computação em Nuvem, Balanceamento de Cargas e Redes Neurais Artificiais.

2.1 Computação em Nuvem

Computação em Nuvem ou *Cloud Computing* refere-se tanto à aplicações entregues como serviços via Internet quanto ao *hardware* e sistemas de *software* em *data centers* que provém tais serviços (ARMBRUST et al., 2010). Atualmente, serviços em nuvem tem criado novas oportunidades por oferecerem serviços a valores mais acessíveis para empresas de diversos tamanhos. Nos últimos anos, os benefícios econômicos tem se tornado evidentes tanto pela aceitação deste conceito, quanto pela adesão das empresas ao modelo de Computação em Nuvem (RAZA et al., 2015).

A Computação em Nuvem pode ser definida como um “modelo que permite acesso a rede de forma ubíqua, conveniente, *on-demand* para compartilhar um grupo de recursos computacionais configuráveis (como por exemplo rede, servidores, armazenamento, aplicações e serviços) que podem ser rapidamente fornecidos e gerados com mínimo esforço de gerenciamento ou interação com o provedor”(MELL; GRANCE, 2011). Ainda segundo Mell (2011), para ser considerado um modelo em nuvem, uma plataforma de Computação em Nuvem deve atender as características essenciais, a saber:

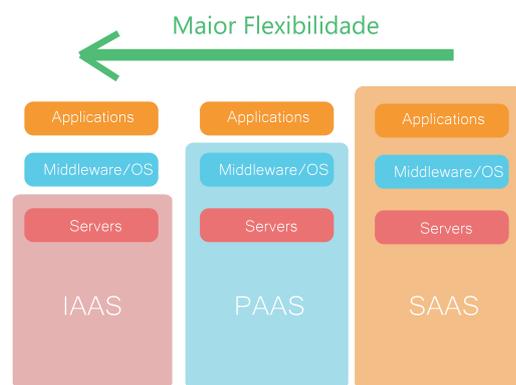
- a) *On-demand self-service*, definido como a capacidade do usuário de autorregular características como armazenamento de acordo com sua necessidade e de forma automática;
- b) *Broad network access*, que define a disponibilidade de acesso através da rede utilizando dispositivos diversos;
- c) *Resource pooling*, fornecimento dos serviços a múltiplos usuários, associando dinamicamente recursos heterogêneos de acordo com a demanda solicitada. Há ainda um senso de independência, visto que o usuário geralmente não possui conhecimento ou controle sobre a localização física do recurso, devido ao alto nível de abstração;
- d) *Rapid elasticity*, referente a capacidades relacionadas a elasticidade de fornecimento e alocação, em alguns casos automática, para garantir escalabilidade, dando a impressão de um serviço com capacidades ilimitadas;

- e) *Measured service*, referente ao controle e otimização automática de recursos usados aproveitando a capacidade de medição no nível de abstração apropriado para o tipo de serviço (como por exemplo armazenamento, processamento, largura de banda e número de usuários ativos). Recursos podem ser monitorados e controlados, o que garante a transparência tanto para o fornecedor do serviço quanto para o consumidor.

2.1.1 Modelos de Computação em Nuvem e Tipos

Além das características essenciais, há também a separação por tipos de serviço. Neste caso, o tipo de serviço em nuvem é dividido com base em como funciona o fornecimento do serviço, bem como qual o nível de abstração atendido por aquele serviço, tal como apresentado na Figura 1. Neste sentido, há três tipos de separação:

Figura 1 – Modelos de Computação em Nuvem



Fonte: PBXL (2017)

Software as a Service (SaaS): modelo no qual o consumidor utiliza o serviço fornecido por uma aplicação executada em uma infraestrutura de nuvem, sendo que esta aplicação fica acessível a partir de vários dispositivos do tipo cliente, através de uma interface que pode ser um navegador web ou programa. Neste caso especificamente, o cliente não gerencia a infraestrutura da nuvem, tampouco da aplicação utilizada (MELL; GRANCE, 2011).

Platform as a Service (PaaS): este modelo permite que se tenha acesso a uma plataforma com infraestrutura de nuvem onde o usuário pode criar aplicações utilizando linguagens de programação, bibliotecas, serviços e ferramentas suportadas pelo fornecedor do serviço. O usuário não gerencia ou controla camadas mais baixas da infraestrutura tais como servidores, sistema operacional ou armazenamento, embora seja responsável pela

aplicação e pelo ambiente servidor da mesma(MELL; GRANCE, 2011). Como exemplo dessa plataforma é possível citar o Windows Azure (MICROSOFT, 2016)

Infrastructure as a Service (IaaS): a oferta de infraestrutura como serviço é o menos abstrato dos modelos. Neste caso o usuário tem a capacidade de gerenciar recursos fundamentais tais como sistema operacional e demais aplicações. Embora não tenha controle sobre a camada de *hardware* da infraestrutura da nuvem, ele pode controlar recursos de sistema, armazenamento e instalar aplicações e, em alguns casos, até ter certo controle sobre certos componentes de rede, como *firewalls*(ZHANG; CHENG; BOUTABA, 2010). Como exemplo de fornecedores de serviço em nuvem que seguem esse modelo temos a *Amazon EC2*(SERVICES, 2016a).

Além da identificação do modelo há também a separação por tipos (ZHANG; CHENG; BOUTABA, 2010). Geralmente, são divididos em três: públicas, privadas e híbridas. Nuvens públicas tem como característica principal oferecer serviços para o público em geral, sendo gerenciadas por uma organização. Nuvens privadas possuem infraestrutura fornecida e gerenciada por uma organização proprietária da nuvem, sendo gerenciada e operada pela mesma. Nuvens híbridas, por sua vez, são compostas por duas ou mais infraestruturas distintas (privada e pública, por exemplo), que, embora mantenham suas características, são ligadas de alguma maneira habilitando a portabilidade entre elas(MELL; GRANCE, 2011) (ZHANG; CHENG; BOUTABA, 2010) (RIMAL; CHOI; LUMB, 2009).

Atualmente, há diversas plataformas e empresas que fornecem *softwares* e serviços para habilitar a Computação em Nuvem, seja em nuvens públicas, privadas ou híbridas. Há também uma grande quantidade de modelos, tipos e serviços de maneira acessível.

AWS - Amazon Web Services

A *AWS - Amazon Web Services* consiste em uma plataforma que fornece diversos serviços em nuvem, incluindo infraestrutura sob demanda para computação de alto desempenho, armazenamento, aplicações, dentre outros recursos.(SERVICES, 2016c). A *AWS* oferece serviços que vão desde serviços para computação, como por exemplo a *EC2*(SERVICES, 2016a), passando por armazenamento e bancos de dados (SERVICES, 2017a), até ferramentas para infraestruturas de Internet das Coisas (IoT)(SERVICES, 2017b).

CloudStack

O Apache CloudStack é um *software open source* desenvolvido para instalar e gerenciar redes de máquinas virtuais baseados em IaaS para plataformas em nuvem. Contém diversos serviços públicos, podendo ser utilizado em conjunto com nuvens privadas, gerando estruturas híbridas. Sua funcionalidade mais procurada é a possibilidade de utilizar

a rede como um serviço, pela facilidade de controle e gerenciamento.

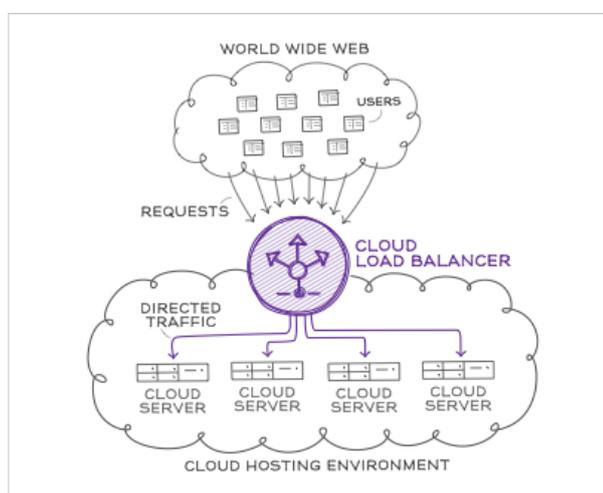
Outras Plataformas e Softwares

Atualmente há um grande número de plataformas e empresas que fornecem serviços de computação em nuvem, como por exemplo a Microsoft (Azure), IBM (IBM Cloud) e Google (Google Cloud Platform), Eucalyptus, dentre outras.

2.2 Balanceamento de Carga

Balanceamento de Carga ou *Load Balancing* consiste em transformar uma grande carga de processamento em cargas menores com o intuito de aumentar o desempenho do sistema através da distribuição entre nós desse sistema. Em um ambiente de computação em nuvem, o balanceamento de cargas é necessário para realizar de maneira dinâmica a distribuição local da carga de maneira igualitária, entre todos os nós (KAUR; LUTHRA, 2012). Cabe ressaltar que os nós de uma nuvem referem-se a infraestrutura de *hardware* distribuída em localidades distintas e transparentes ao usuário. A Figura 2 representa uma estrutura simplificada de como funciona o balanceamento de carga.

Figura 2 – Estrutura Simplificada de Balanceamento de Carga



Fonte: rackspace (2017)

Tal conceito se torna essencial quando aplicado a computação em nuvem, uma vez que esse tipo de modelo depende necessariamente da disponibilidade e do tempo de resposta. Isto, acrescido dos conceitos essenciais para Computação em Nuvem apresentam o balanceamento de carga como um problema que requer atenção no contexto de computação em nuvem (SRAN; KAUR, 2013). Assim, a distribuição de carga através da nuvem impacta diretamente no desempenho geral do sistema.

Identificar os desafios encontrados no balanceamento de carga em sistemas de nuvem é um dos caminhos para entender quais características um algoritmo de balanceamento de carga deve funcionar de modo satisfatório. Segundo Nuaimi N. Mohamed (2012), pode-se citar os seguintes tópicos:

- distribuição esparsa dos nós da nuvem: consiste em um desafio visto que criar um algoritmo capaz de funcionar em arquiteturas esparsas e altamente distribuídas torna-se difícil, visto que é preciso levar em consideração a distância entre os nós e conseqüentemente, adequar o algoritmo a atrasos;
- armazenamento/replicação: a replicação de dados entre os nós pode resultar em maior custo, uma vez que mais capacidade de armazenamento é requerida;
- complexidade do algoritmo: é preferível que algoritmos de balanceamento não sejam tão complexos em termos de implementação, pois isto pode resultar em requisições maiores de informação aumentando o tempo de resposta, o que diminui a eficiência;
- ponto de falha: a coleta de dados acerca das informações sobre nós diferentes deve ser projetada com o intuito de evitar erros no algoritmo. Geralmente algoritmos centralizados tendem a ser mais eficientes para certos tipos de arquiteturas, mas costumam falhar em outras. Algoritmos distribuídos, por sua vez, possuem uma abordagem melhor, mas mais complexa.

Os algoritmos de balanceamento de carga podem ser divididos, de forma generalizada, em estáticos - associa cargas com base na habilidade do nó de processar tarefas, sem levar em consideração mudanças em tempo real - e dinâmicos - considera diferentes aspectos dos nós, incluindo capacidade e largura de banda (NUAIMI N. MOHAMED, 2012).

É importante ressaltar que o balanceamento preferivelmente deve ser feito de forma dinâmica, visto que o comportamento da nuvem pode variar e a infraestrutura geralmente é composta de *hardwares* distintos. Além disto, a distribuição de carga busca melhorar a eficiência de sistemas distribuídos considerando questões que ainda apresentam-se como desafios na área de computação em nuvem, tais como gerenciamento eficiente de tráfego, gerenciamento de energia e alocação dinâmica de recursos (ZHANG; CHENG; BOUTABA, 2010). Tendo isto em vista, existem critérios que devem ser considerados na definição das métricas que quantificam a efetividade de algoritmos de balanceamento. Ainda segundo Nuaimi N. Mohamed (2012), podemos citar os seguintes critérios:

1. taxa de transferência - relacionado ao número de tarefas completadas;
2. custo associado ao algoritmo de balanceamento;
3. tolerância a falhas;

4. tempo de resposta;
5. escalabilidade e flexibilidade na capacidade de distribuição das cargas de acordo com a demanda, bem como se adequar rapidamente às demandas;
6. utilização de recursos;
7. desempenho do sistema.

Pode-se considerar que os objetivos do balanceamento de carga são, de maneira geral, melhorar o desempenho do sistema a um custo mais baixo; garantir a escalabilidade e flexibilidade, de forma que o algoritmo se adapte facilmente à possíveis mudanças no sistema e, por fim, garantir que a prioridade (caso haja) seja respeitada (KAUR; LUTHRA, 2012).

Diversos algoritmos de balanceamento foram propostos, com abordagens distintas, mas sempre tentando atender da melhor forma possível os objetivos de balanceamento de carga. Dentre eles pode-se citar o *Round Robin* (SONI; KALRA, 2014), *Biased Random Sampling* (RANGLES; LAMB; TALEB-BENDIAB, 2010), *Active Clustering* (RANGLES; LAMB; TALEB-BENDIAB, 2010), *Ant Colony* (NISHANT et al., 2012) e *Honeybee Foraging Behavior* (RANGLES; LAMB; TALEB-BENDIAB, 2010).

2.3 Redes Nerais Artificiais

Técnicas de reconhecimento de padrões tem por objetivo aprender funções de decisão que separam um conjunto de dados em agrupamentos de amostras (*clusters*) que compartilham propriedades similares (DUDA; HART; STORK, 2000). Tal processo de aprendizado de funções de decisão podem ser geralmente associados a três abordagens: (i) supervisionada, na qual, a princípio, se conhece informações sobre todo o conjunto de treinamento; (ii) semi-supervisionado, no qual parte das informações sobre o conjunto de treinamento é conhecido e (iii) não-supervisionado, no qual não se tem informações prévias a respeito das amostras do conjunto de treinamento.

Técnicas supervisionadas são conhecidas por apresentarem melhores taxas de acurácia, uma vez que a quantidade de informações disponíveis acerca das amostras de treinamento permite que tais técnicas aprendam a classificar de maneira específica determinadas propriedades, bem como permite a construção de estruturas mais complexas de aprendizagem de forma a aprimorar a qualidade do processo de treinamento.

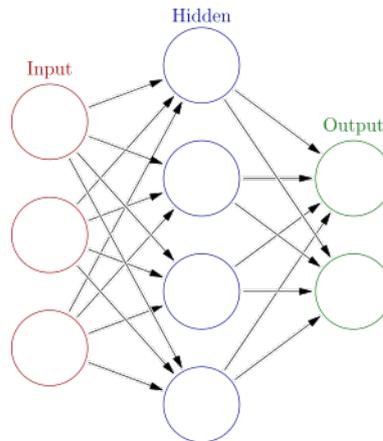
O estado da arte neste campo tem como referência técnicas como Máquinas de Vetores Suporte (*Support Vector Machines – SVM*), proposta por Cortes e Vapnik (1995), Redes Neurais (*Neural Networks*), amplamente discutidas por Haykin (2007), classificadores Bayesianos e os amplamente conhecidos *k*-médias (*k-nearest neighbours – k-NN*), dentre outros (DUDA; HART; STORK, 2000).

Dentro do contexto de reconhecimento e classificação de padrões, as Redes Neurais Artificiais tem sido amplamente estudadas em diferentes contextos de aplicações. Há, atualmente, diversos tipos de configurações de arquiteturas destas redes. De forma geral, sua arquitetura é formada por camadas de neurônios, sendo a configuração mais trivial composta por uma camada de entrada (*input layer*), uma ou mais camadas escondidas (*hidden layer*) e uma camada de saída (*output layer*). Cada neurônio em sua respectiva camada recebe um estímulo que é processado pelo neurônio e propagado à camada seguinte, como representado na figura 3.

As Redes Neurais Artificiais ou *Artificial Neural Networks - ANN* são compostas por elementos chamados neurônios, os quais, segundo Haykin (1999, p. 11), são unidades de processamento que ao receber um valor de entrada (ou *input*) são responsáveis por processá-lo e propagar o sinal para a próxima camada. Este processamento pode ocorrer, basicamente, de três formas diferentes:

- multiplicação da entrada por um peso;
- o neurônio funciona tal qual um somador, ou combinador linear de suas entradas;
- o sinal de entrada é parâmetro de uma função de ativação responsável por limitar a amplitude do sinal de saída, como ocorre em uma sigmoide, por exemplo.

Figura 3 – Estrutura de uma ANN

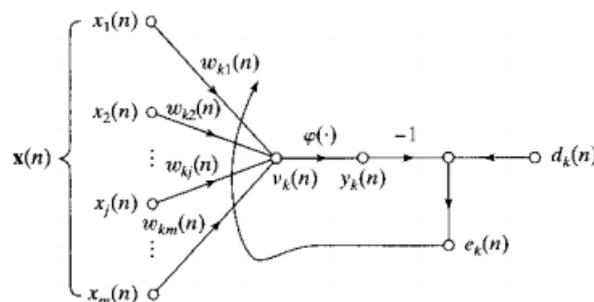


Fonte: WIKIPEDIA (2013)

Dentro deste contexto, há atualmente uma diversidade muito grande de arquiteturas nas quais é possível organizar essas pequenas unidades. O poder de aprendizado das redes neurais permite efetuar desde tarefas mais simples, até atuar em complexos sistemas de aprendizado e classificação, como por exemplo, no reconhecimento de imagens e voz - basta ver exemplos como os das Redes Neurais Recorrentes (*Recurrent Neural Networks - RNN*) (BULLINARIA, 2015) ou Redes Neurais de Convolução (*Convolutional Neural Networks - CNN*). Dentre as aplicações de ANNs, podemos citar o uso no controle, classificação, previsão, como filtros para extrair informações, dentre outras.

No entanto, nem todos os problemas exigem arquiteturas tão complexas. Segundo Haykin (1999, p. 21-23), há três tipos fundamentais de estruturas: *Single Layer Feedforward Networks* (Redes Neurais de Camadas Únicas), *Multilayer Feedforward Networks* (Redes com Múltiplas Camadas) e *Recurrent Networks* (Redes Neurais Recorrentes).

Figura 4 – Fluxo de aprendizado de uma rede neural



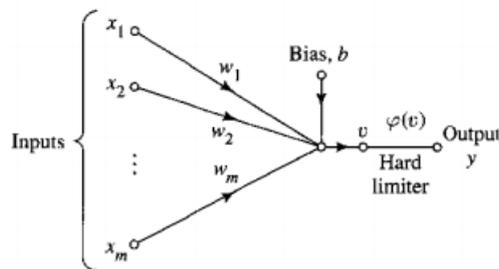
Fonte: Haykin (1999, p. 52)

É importante ressaltar que a quantidade de camadas em uma rede impacta diretamente no desempenho em relação ao processamento e na velocidade de aprendizado. Além disso, a escolha de implementações de arquiteturas que atendam à um problema específico ou sejam demasiadamente especializadas não é recomendado pois pode tornar a arquitetura ineficiente na resolução de problemas diferentes daqueles na qual foi treinada.

2.3.1 Perceptron

Perceptrons de única camada são uma alternativa simples de implementação de rede neural para identificar dados que são linearmente separáveis. O conceito aplicado é de uma rede que recebe vetores de características que treinam o Perceptron para identificar padrões e classificar amostras. Um neurônio também forma uma base de um filtro adaptativo, que pode ser aplicado em um sistema dinâmico. De forma geral, a Figura 4 representa a propagação de sinal por um modelo adaptativo. No caso de um Perceptron, a propagação de sinal é feita por meio de uma função não-linear. A representação do fluxo de sinal de um Perceptron, ilustrado na Figura 5, denota exatamente este fluxo de processamento.

Figura 5 – Fluxo de sinal em um Perceptron



Fonte: Haykin (1999, p. 136)

O Perceptron tem em seu modelo um fluxo (representado na figura 5) no qual os pesos w_1, w_2, \dots, w_m são sinapses do Perceptron, correspondentes à cada uma de suas respectivas entradas, denotadas por x_1, x_2, \dots, x_m . Uma *bias* pode ser aplicada externamente, denotada por b . O nó v recebe a propagação do sinal e, a partir da aplicação de uma função propaga a saída y (*Output*). A partir do modelo, pode-se inferir que o limitador da camada de entrada, ou local de indução dos neurônios é denotado pela equação 2.1.

$$v = \sum_{i=1}^m (w_i \times x_i + b) \quad (2.1)$$

O objetivo de um Perceptron, de forma geral, é classificar corretamente a entrada, ou seja, x em uma de duas classes, C_1 ou C_2 . Neste contexto, a regra de decisão associa

o ponto representado pelo vetor x em uma das classes, dada sua saída. Os pesos em w podem ser ajustados a cada iteração e, para isso, usa-se uma função de ajuste, seja por correção de erro, seja por regras de convergência Haykin (1999). Nesse caso, um dos algoritmos que pode ser seguido está representado a seguir:

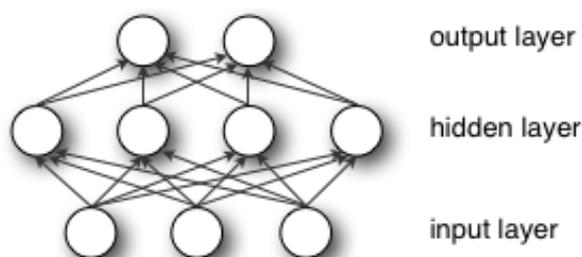
1. *inicialização de parâmetros*: inicialização do vetor de pesos (lembrando que a atualização é feita em n steps ou iterações);
2. *ativação*: em cada *step*, receber a entrada (ou *input*) e ativar o Perceptron;
3. *computar a saída*: resposta do perceptron $y(n)$, ou seja, a saída da função de ativação;
4. *ajuste*: ajuste dos pesos de w
5. *continuação*: repetição do processo

2.3.2 ANNs com Múltiplas Camadas

Como variação do Perceptron tem-se uma rede de múltiplas camadas, chamada Perceptron Multicamada (PMC), que são constituídas de um conjunto de neurônios sensoriais que formam uma camada de entrada, uma ou mais camadas ocultas de nós computacionais e uma camada de saída, também com nós computacionais. O sinal de entrada, recebido pela camada de entrada, é propagado para a próxima camada, repetindo esse processo até a camada de saída Haykin (1998, p. 98).

Cada neurônio possui uma função de ativação, geralmente não-linear. A não linearidade dessa função é importante para a saída da rede, visto que permite que valores de classificação não binária sejam atribuídos, o que permite que haja classificação de duas ou mais classes. A camada escondida capacitam a rede a aprender tarefas complexas extraindo progressivamente as características mais significativas. A Figura 6 apresenta a estrutura generalizada de um Perceptron Multicamadas.

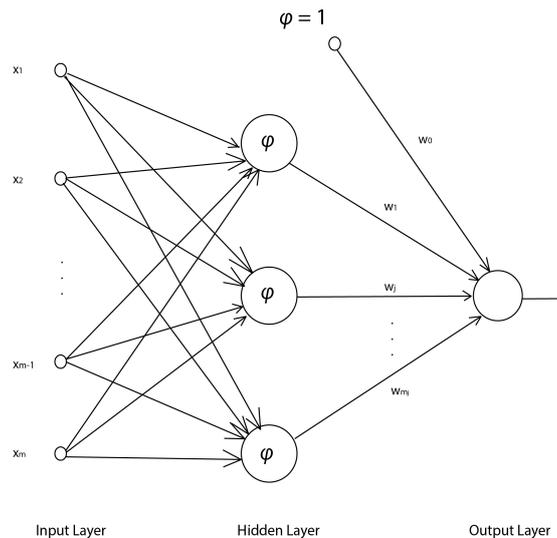
Figura 6 – Estrutura de um Perceptron Multicamada



Fonte: lab. (2016)

Redes Neurais de Base Radial (*Radial Basis Function - RBF*) é uma rede neural semelhante ao PMC constituída por no mínimo três camadas, sendo uma de entrada, uma escondida e uma de saída. Nesta rede, a camada de entrada recebe um vetor de características ao qual é aplicada uma transformação não-linear na camada escondida. A Figura 7 apresenta a estrutura de uma RBF genérica.

Figura 7 – Estrutura de uma Rede Neural com função de Base Radial



Fonte: Baseado em Haykin (1999, p. 278)

O diferencial da RBF para o PCM é, além da estrutura, a função de ativação aplicada para aprender as características mais relevantes. Neste caso, a função de ativação mais empregada é a Gaussiana, por sua formulação simples.

Existem outros tipos de redes, tais quais as Redes Neurais Probabilística (*Probabilistic Neural Network - PNN*) (SPECHT, 1990b), que funcionam de maneira semelhante a uma RBF. Sua camada escondida, no entanto, é dividida em duas partes: uma camada de padrões e uma camada de somatório. A Figura 6 apresenta a estrutura generalizada de uma PNN. A primeira tem por objetivo calcular a probabilidade de um determinado vetor de característica y pertencer à uma classe x . A segunda camada, por sua vez, realiza um somatório dessas probabilidades. Na camada de saída, os dados recebidos da camada anterior são utilizados para classificar as amostras, atribuindo a y o rótulo correspondente ao neurônio com maior probabilidade.

3 METODOLOGIA

Neste capítulo estão descritos as ferramentas e metodologia utilizadas para o desenvolvimento do projeto, além da apresentação da arquitetura do protótipo de balanceamento proposta.

3.1 Tecnologias e Ferramentas Utilizadas

Nesta seção são apresentadas as ferramentas e tecnologias usadas na implementação do projeto.

3.1.1 AWS - *Amazon Web Services*

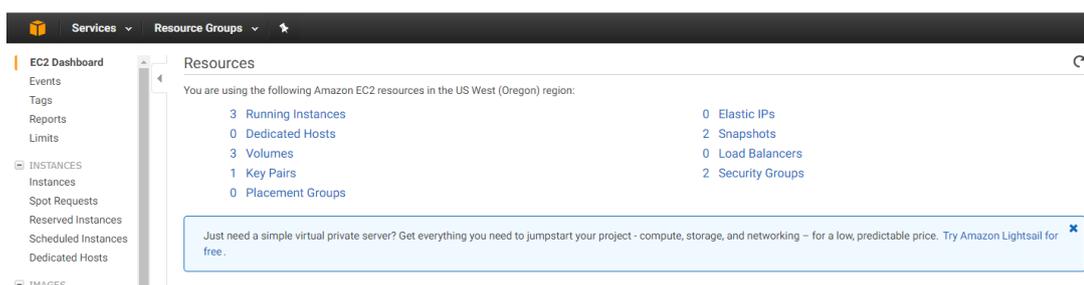
A Amazon oferece um vasto portfólio de serviços baseados em Computação em Nuvem, concentrados na Amazon Web Services. Seus produtos variam desde a oferta de IaaS, com servidores virtuais, nuvens privadas, dentre outras ferramentas, até aplicações do tipo SaaS, como o serviço de armazenamento em nuvem (*Amazon Cloud Drive*) (AMAZON, 2016).

Sua plataforma de serviços web é centrada no Amazon Elastic Compute Cloud ou EC2. Trata-se de um “serviço que fornece capacidade de computação redimensionável na nuvem” (SERVICES, 2016a). Por ter foco em desenvolvedores, possui uma interface com maior controle de recursos computacionais, incluindo configurações de memória, processamento, sistema operacional e armazenamento. Além disso, há suporte para outras ferramentas, como banco de dados, *Big Data* e até mesmo Internet das Coisas. Além do baixo custo, uma das vantagens da Amazon está na flexibilidade dos planos de serviços, que são ajustáveis de acordo com a demanda (EC2, 2016).

O acesso é feito por meio do AWS Management Console, que consiste em uma interface web para gerenciamento dos serviços. No caso da EC2, a visualização e acompanhamento de estado das instâncias é feito pela EC2 Dashboard, como exemplificado na figura 8.

A AWS oferece diversos tipos de instâncias, as quais os tipos de uso variam de acordo com a aplicação. Instâncias do tipo T2, por exemplo, são indicadas para aplicações web, ambientes de desenvolvimento e teste e preparação de aplicações. Os preços também variam de acordo com o tipo de instância, uso e capacidade computacional (SERVICES, 2016d). Para este projeto, os servidores utilizados, bem como suas configurações estão descritos na tabela 1. Todas as instâncias possuem como sistema operacional Ubuntu Server 16.04 LTS.

Figura 8 – EC2



Fonte: elaborada pela autora

Tabela 1 – Servidores

Servidor	Tipo de instância	Memória	CPUs	Armazenamento
Servidor de balanceamento	t2.medium	4	2	Somente EBS
Servidores do <i>cluster</i> de aplicação	t2.micro	1	1	Somente EBS

Elaborado pela autora.

Os servidores de requisição descritos na tabela 1 foram configurados para executar um código semelhante ao apresentado em 3.1.2 e são acessados apenas pelo servidor de balanceamento, através de seu DNS público, disponibilizado pela AWS para cada instância.

3.1.2 Node.js e npm

Node.js¹ é uma plataforma de aplicações baseada no motor de Javascript do Google Chrome chamado V8(NODEJS, 2017). Nessa plataforma, o Javascript que geralmente é utilizado no navegador, como forma de tornar dinâmica a interação do lado do cliente, é utilizado como linguagem para fornecer programação direta para o sistema operacional, permitindo a criação e execução no servidor. Seu código fonte está disponível² para acesso, colaboração e comentários. Essa plataforma funciona com base em um modelo voltado à eventos, ou seja, a execução não é bloqueada com eventos de entrada e saída (I/O), como nos demais modelos. O Node dispara funções específicas quando os eventos são acionados. Além disso, o Nodejs também possui um gerenciador de pacotes, o *Node Package Manager* (denominado *npm*). Seu objetivo é a instalação e distribuição de pacotes e bibliotecas criadas pela comunidade e que tenham código livre, o que permite que qualquer desenvolvedor disponibilize sua própria biblioteca.

Pacotes criados da maneira descrita anteriormente têm suas informações salvas em um arquivo chamado *packages.json*, o qual contém informações como Nome do projeto,

¹ <<https://nodejs.org/>>

² <<https://github.com/nodejs/node>>

Figura 9 – Código exemplo de servidor em nodejs

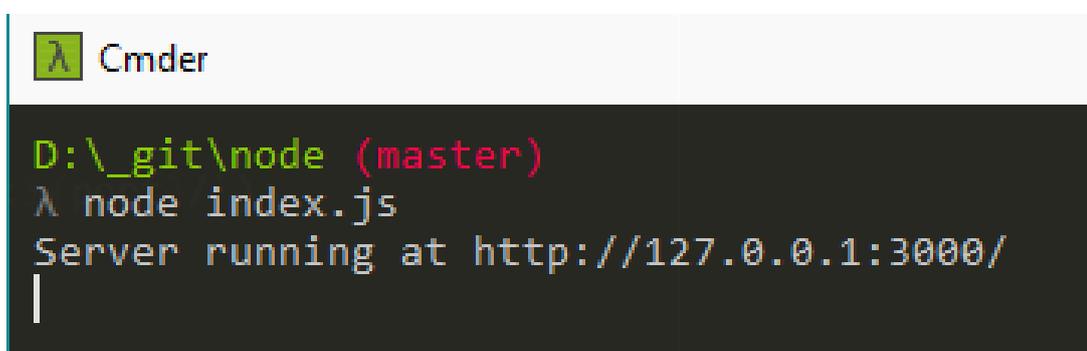
```
1  const http = require('http');
2
3  const hostname = '127.0.0.1';
4  const port = 3000;
5
6  const server = http.createServer((req, res) => {
7    res.statusCode = 200;
8    res.setHeader('Content-Type', 'text/plain');
9    res.end('Hello World\n');
10 });
11
12 server.listen(port, hostname, () => {
13   console.log(`Server running at http://${hostname}:${port}/`);
14 });
```

Fonte: elaborado pela autora

do criador, link para o repositório no GitHub, dependências de bibliotecas com a versão específica instalada, entre outros. Através do comando *npm install* o gerenciador instala as dependências do módulo para executar a aplicação. A plataforma é acessível via terminal de comandos e está disponível para Windows, Mac OS e sistemas operacionais baseados em Linux. Neste projeto, a versão utilizada da plataforma Nodejs foi a 6.6.0.

A figura 9 apresenta um exemplo de código para servidor configurado localmente, na porta 3000. A figura 10 ilustra a execução do código descrito na 9 utilizando a plataforma do Nodejs.

Figura 10 – Exemplo de execução de código em nodejs



```
λ Cmder
D:\_git\node (master)
λ node index.js
Server running at http://127.0.0.1:3000/
|
```

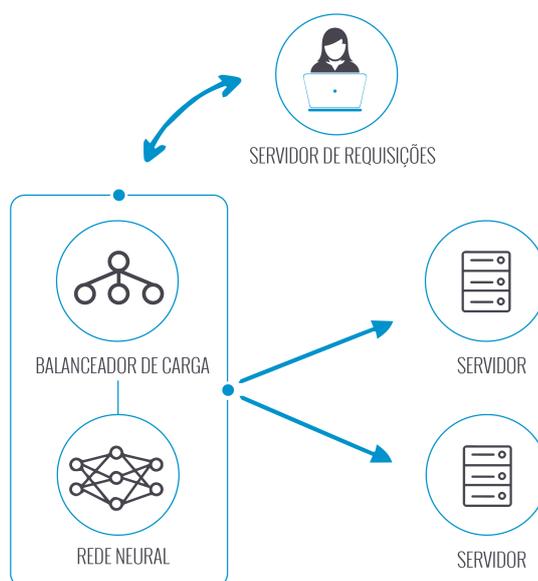
Fonte: elaborado pela autora

3.2 Métodos e Etapas

A primeira fase deste trabalho se concentrou-se no levantamento bibliográfico sobre *Cloud Computing*, balanceamento de carga e Redes Neurais Artificiais. Nesta fase, o estudo se concentrou na pesquisa de serviços de *Cloud* disponíveis no mercado, visto que o teste da ferramenta foi realizado a partir de uma infraestrutura hospedada em uma *Cloud* real.

A fase seguinte concentrou-se na análise e levantamento das tecnologias utilizadas na implementação da estrutura para testes, inicialmente estudando tecnologias como a ferramenta CloudStack (STACK, 2016) e a OpenStack (OPENSTACK, 2016). Devido à necessidade de possuir uma estrutura de servidores física, ambas as opções citadas foram descartadas e o serviço escolhido foi a EC2 (SERVICES, 2016a), fornecido pela AWS. Durante essa fase, ainda, houve o planejamento e estruturação do módulo que engloba a RNA, bem como sua arquitetura, posteriormente integrada ao módulo de balanceamento de carga. A arquitetura escolhida segue a estrutura descrita pela figura 11.

Figura 11 – Arquitetura simplificada do projeto



Fonte: elaborado pela autora

A arquitetura do protótipo pode ser dividida em três módulos, como ilustrado na figura 11: o primeiro, consiste no servidor responsável pelo balanceamento de carga, o qual recebe todas as requisições que serão posteriormente direcionadas para o *cluster* de servidores de dados; o segundo módulo é composto pelo *cluster* de servidores de dados; e o terceiro é um servidor que gera requisições. Nesta arquitetura, a rede neural é um submódulo do balanceador, sendo responsável pela atribuição de pesos à cada um dos

servidores presentes no *cluster* de dados. Estes pesos são utilizados pelo algoritmo de balanceamento para direcionar as requisições aos servidores de modo equilibrado.

A terceira fase do projeto concentrou-se na implementação e testes do módulo de balanceamento. Nessa fase, o estudo das tecnologias Nodejs e EC2 permitiu a implementação da aplicação e os testes com *clusters* em nuvem. Essa fase contemplou, ainda, a análise do desempenho da abordagem escolhida, cujo resultado está descrito na seção de Resultados. A fase final do projeto se concentrou na escrita do relatório e apresentação dos resultados obtidos.

4 ARQUITETURA DO PROJETO

A arquitetura implementada para o módulo de balanceamento segue a proposta apresentada na figura 11. O processo de desenvolvimento foi dividido em módulos: *Load Balancer* (balanceador de carga), *Neural Network* (Rede Neural), *Request Server* (servidor de requisições) e *Servers*, sendo os dois primeiros parte integrante do protótipo de balanceamento de carga e os dois últimos utilizados para teste do protótipo.

O **módulo de servers** é composto por três servidores do tipo t2.micro, conforme descrito na tabela 1, cada um deles configurado como um servidor que recebe requisições em uma porta específica e retorna o conteúdo solicitado. O objetivo do módulo do balanceador de carga é distribuir a carga de requisições entre estes servidores. Para executar e configurar estes servidores, fez-se uso do Nodejs, com código semelhante ao apresentado na seção 3.1.2, cujo acesso está ilustrado na figura 12.

Figura 12 – Conexão ao servidor de Requisições

```

Welcome to Ubuntu 16.04.1 LTS (GNU/Linux 4.4.0-57-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

22 packages can be updated.
0 updates are security updates.

*** System restart required ***
Last login: Fri Dec 30 05:38:34 2016 from 187.66.121.105
ubuntu@ip-172-31-22-48:~$ ls
nodejs
ubuntu@ip-172-31-22-48:~$ cd nodejs/hello/
ubuntu@ip-172-31-22-48:~/nodejs/hello$ node index.js

```

Fonte: elaborada pela autora

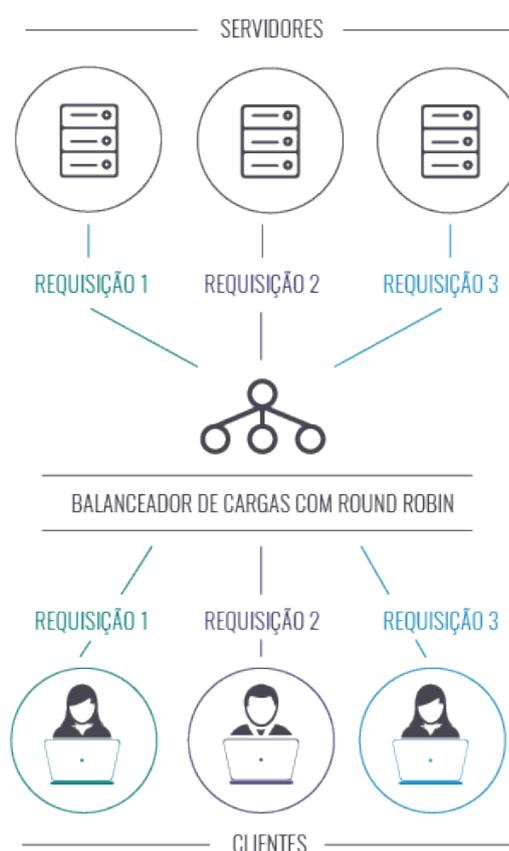
O **módulo de servidor de requisições** simula um cliente. Seu algoritmo gera aleatoriamente uma quantidade de solicitações ao servidor de balanceamento e aguarda a resposta da requisição, gravando em um arquivo de *log* se a solicitação foi atendida com sucesso.

O último módulo é o protótipo de aplicação em si, composto por duas partes: a primeira, é responsável pelo algoritmo de distribuição das requisições e a segunda implementa a Rede Neural Artificial.

O **módulo de balanceamento de carga** recebe todas as requisições geradas pelo módulo de requisições e distribui entre os servidores. Para efetuar esta distribuição, escolheu-se utilizar o conceito do algoritmo de Round Robin (NGINX, 2017b), no qual

as requisições são distribuídas entre os servidores seguindo a ideia de turnos, ou seja, em uma lista de servidores, o balanceador de carga direciona a requisição para o primeiro, a próxima para o segundo da lista e assim por diante, até atingir o último servidor da lista, reiniciando o *loop*. A figura 13 ilustra essa distribuição.

Figura 13 – Round Robin



Fonte: elaborada pela autora

Round Robin é um algoritmo que tem como principal vantagem a estabilidade e a simplicidade de implementação e execução. Porém, sua eficiência ou precisão não é tão grande quando comparada à abordagens mais complexas pois muitos balanceadores não consideram as diferenças entre os servidores que recebem as requisições (NGINX, 2017b). Por esta razão, há variantes de implementações de *Round Robin* que buscam minimizar esta desvantagem, dentre elas o *Round Robin* com pesos e *Round Robin* dinâmico. O primeiro, atribui pesos aos servidores associados na lista e a distribuição é feita com base nesse atributo. Servidores com maior peso recebem mais requisições. A segunda variante associa os pesos dinamicamente, de acordo com a carga atual e a ociosidade de cada servidor.

O módulo de balanceamento de carga, então, utiliza o conceito de Round Robin com pesos, atribuindo pesos aos servidores do módulo de *servers*. Os pesos são definidos com base no **módulo de Rede Neural**, descrito a seguir. Ainda em relação ao módulo de balanceamento, a implementação do protótipo incorpora os pesos atribuídos, ou melhor, a classificação advinda da Rede Neural Artificial para executar o balanceamento. Este módulo também é responsável por atualizar as informações das características dos servidores, utilizadas posteriormente na Rede Neural.

Por fim, o **Módulo de Rede Neural** implementa a rede responsável por atribuir os pesos aos servidores presentes na lista de servidores. As características consideradas no processamento dos pesos estão descritas na tabela 2. A saída deste módulo é ligada diretamente ao módulo de balanceamento, através de um vetor que indica quais servidores estão aptos a receber requisições.

4.1 Arquitetura da Rede Neural

A seção 2.3 apresenta alguns modelos de arquiteturas que podem ser aplicadas para resolver problemas de classificação. Como discutido na seção 2.3, a arquitetura do tipo de RNA ou abordagem utilizada para resolver um problema que envolva aprendizado está diretamente relacionada ao tipo de problema que se deseja resolver com determinada técnica. Este trabalho tem definido em seus objetivos aplicar uma Rede Neural Artificial para aprender se os servidores gerenciados pelo **módulo de balanceamento** estão aptos a realizar uma requisição.

Como discutido na seção 2.2, este tipo de aplicação deve considerar elementos como distribuição esparsa da arquitetura em nuvem, capacidade de armazenamento, complexidade do algoritmo e pontos de falhas, tempo de resposta, utilização de recursos, dentre outros critérios que determinam a eficiência de uma aplicação de balanceamento.

Diante do problema, este trabalho considerou que, ao incorporar ao algoritmo de balanceamento uma RNA, um critério crucial para a escolha de sua arquitetura está diretamente ligada às características citadas anteriormente. Portanto, elementos como tempo de resposta e complexidade do algoritmo receberam certa prioridade em relação ao desempenho da rede na classificação em si. Como discutido na seção 2.3, a escolha da arquitetura, suas funções de ativação, tipo de aprendizado, e, principalmente, número de camadas e neurônios, são características que impactam diretamente no tempo de processamento. Redes com muitas camadas, como as CNNs, por exemplo, levam um tempo consideravelmente alto no aprendizado e possuem uma arquitetura sofisticada.

Considerando que a RNA escolhida seria incorporada ao **módulo de balanceamento** e que, por ser uma aplicação de balanceamento de carga de requisições - onde o tempo de resposta é importantíssimo - o modelo escolhido foi o Perceptron, por ser de

simples implementação. Além disso, a definição das características também permitiu que este modelo se apresentasse como uma boa escolha, visto que o número de características é pequeno e o problema de classificação, neste caso, não exige uma função tão complexa de separação dos dados no espaço de características.

Portanto, este trabalho considerou como critério de escolha da rede a simplicidade da implementação da RNA para diminuir a complexidade do **módulo de balanceamento**, dimensão do problema (em relação às suas características) e tempo de resposta. Tendo isto em vista, o modelo escolhido foi o Perceptron e a arquitetura da rede está descrita a seguir.

Definição de características

A tabela 2 descreve as características que a RNA recebe como *input*. Cada uma delas impacta diretamente no desempenho da distribuição das requisições. O tipo de servidor, apesar de ser um parâmetro estático (ou seja, não muda ao longo da execução), quando colocado em conjunto com as demais características, torna-se importante. Por exemplo, um servidor que possui maior capacidade de processamento e está à uma distância maior do que um servidor com menor capacidade e mais próximo pode ter um tempo de resposta semelhante, equilibrando o peso do servidor em relação aos demais.

Tabela 2 – Características

Característica	Descrição
Tipo de servidor	t2.micro ou t2.medium - servidores com maior poder de processamento podem receber um número maior de requisições
Tempo de resposta	Tempo para receber e responder à requisição
Estado	Indica se o servidor está apto à receber requisições
Distância	Atribui um valor de distância (física) para o servidor - Neste caso, servidores mais distantes demoram mais para responder

Elaborado pela autora.

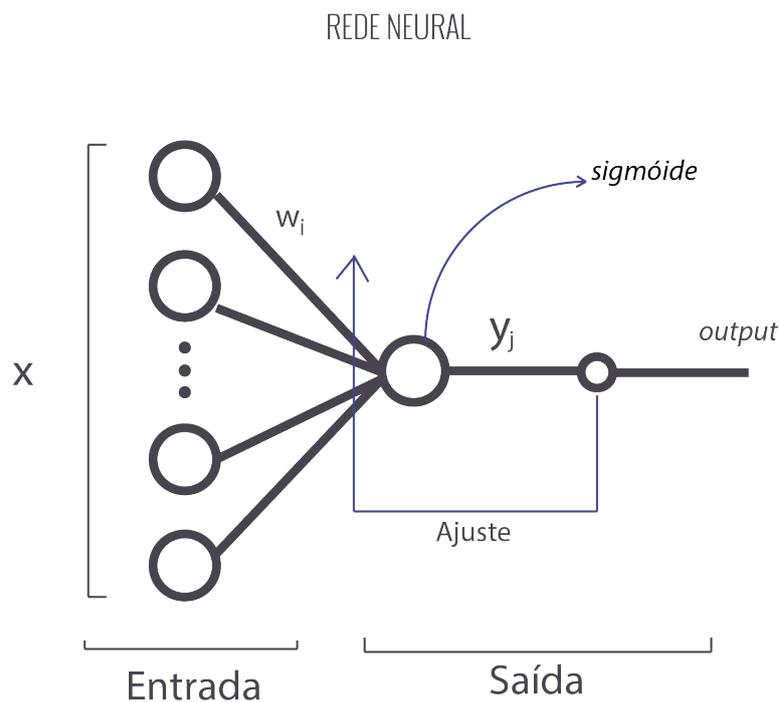
Características como tempo de resposta e estado são dinâmicas, ou seja, mudam ao longo da execução do algoritmo. Essas características impactam diretamente na decisão, visto que, um servidor que recebe muitas requisições (ou seja, está sobrecarregado) em uma iteração pode, na seguinte, estar livre, caso consiga responder à todas as requisições em sua fila. A discussão acerca da escolha destas características e sua efetividade é apresentada no capítulo de 5.

Implementação da Rede Neural com Perceptron

A Rede Neural escolhida segue a ideia de Perceptron apresentada na seção 2.3. A rede contém duas camadas: uma de entrada e uma camada de saída. Por se tratar de uma rede neural aplicada à balanceamento de carga, alguns conceitos foram levados em conta

ao estabelecer a forma de aprendizado dinâmico. Pelas razões elencadas no início desta seção, a rede neural escolhida possui poucas camadas, como ilustrado pela figura 14, com o objetivo de aumentar a velocidade de processamento e garantir que o tempo de resposta do balanceador não seja tão afetado pelo tempo de processamento da RNA. Além disso, a quantidade de características e o tipo de saída esperada na arquitetura proposta por este projeto permitem uma implementação de uma rede deste tipo.

Figura 14 – Camadas implementadas na rede neural



Fonte: elaborada pela autora

A primeira camada presente na arquitetura da rede neural é a camada de entrada. É composta por um vetor x de tamanho n , sendo n igual ao número de características (apresentadas na tabela 2) do servidor. Os pesos que essas características tem na classificação são armazenados em w . Em teoria, w consiste em uma matriz que possui n linhas (quantidade de *inputs*) e m colunas (quantidade de *outputs*). Como a saída é a classificação se o servidor está apto à receber a requisição, ou seja, o *output* da rede é a classificação como "apto" ou "não apto", um único neurônio nesta camada é suficiente, logo, neste caso, w passa a ser um vetor de pesos.

As sinapses ou propagação de sinal da primeira camada para a segunda é feita seguindo o cálculo na equação 4.1.

$$x_i = x_i \times w_i \quad (4.1)$$

A segunda camada realiza um somatório das características (4.2) e aplica a função sigmoide da equação 4.3 ao resultado. y_j é a variável que guarda o valor do somatório da iteração j . Antes de aplicar a sigmoide, os valores são multiplicados por 0.02 para diminuir a faixa de valores na saída.

$$y_j = \sum_{i=1}^n x_i \quad (4.2)$$

$$S(y_j) = \frac{1}{(1 + e^{y_j})} \quad (4.3)$$

A última camada utiliza o *output* para ajustar os pesos em w . Esta é a fase de treinamento da rede, que ocorre de maneira dinâmica. Ao fim da classificação, o *output* identifica o servidor como "apto" ou "não apto" a receber requisições. Se for classificado como "apto", e a requisição for direcionada para ele então a classificação foi correta o algoritmo ajusta o peso que mais influenciou a decisão, seguindo a equação 4.4. Caso a classificação seja incorreta, e a requisição retornou uma falha ou o servidor não conseguiu atender a requisição, o ajuste de peso diminui o peso da característica que mais influenciou a decisão, seguindo a equação 4.5

$$w_i = w_i + 0.5 \quad (4.4)$$

$$w_i = w_i - 0.5 \quad (4.5)$$

Ao fim do processamento, os valores de w são salvos para uso posterior. A ideia de manter o treinamento constante da rede advém de sua implementação mais simples, que permite velocidade no processamento e atribuição dos valores de pesos utilizados pelo balanceador de cargas, visto que os as entradas mudam de maneira rápida e o treinamento depende dos resultados anteriores.

4.2 Implementação e Testes

Por se tratar de uma aplicação de *backend* implementada em Nodejs, o protótipo não possui interface gráfica, de modo que sua interface de comandos é feita pelo terminal, como pode ser observado na Figura 15. Ao iniciar o acesso, o balanceador verifica quais servidores estão ativos e os coloca em uma lista, como apresentado na Figura 16. Então, a aplicação verifica a conexão com cada um dos servidores na lista, para garantir que o mesmo possui o estado saudável.

Em seguida, são atribuídas os valores das características para cada servidor na lista, seguindo a definição apresentada na tabela 2. Caso seja a primeira vez que a aplicação de

balanceamento é executada, w é preenchido com valores padrões. Caso contrário, dados salvos em um arquivo que guarda as informações de peso do processamento anterior são carregados.

A implementação foi feita em Nodejs e o código está disponível no Github¹ da autora. O algoritmo implementado para balanceamento de cargas com a rede neural segue os seguintes passos:

1. *inicialização de parâmetros*: inicialização do vetor de pesos, objeto que armazena dados dos servidores;
2. *se nova requisição recebida*:
 - a) classificar cada servidor (chamar a RNA)
 - b) escolher a melhor opção para direcionar a requisição
 - c) verifica se a requisição foi atendida
 - d) manda o resultado para o cliente
3. *Ajustar os pesos*: conforme descrito em 4.1;
4. *Caso desativado*: salvar parâmetros de peso;
5. *Senão*: voltar ao primeiro item;

As figuras 15, 16 e 17 apresentam a interface no console da aplicação de balanceamento em execução. As informações apresentadas são a lista de servidores ativos e os pesos mais recentes.

Esta parte da aplicação corresponde ao **módulo de balanceamento**, executado no servidor t2.medium.

¹ <<https://github.com/ksetoue>>

Figura 15 – Screenshot de teste realizado com a Rede Neural

```
Cmdr
*****
      Load Balancer
*****
Active servers = 3
-----
Servers:
http://ec2-52-26-51-200.us-west-2.compute.amazonaws.com:8000
http://ec2-35-167-36-190.us-west-2.compute.amazonaws.com:8000
http://ec2-35-167-32-77.us-west-2.compute.amazonaws.com:8000
-----
Running load Balancer...
-----
Weights:
responseTime: 1047.5
state: 1
type: 1
distance: 1
```

Fonte: elaborada pela autora

Figura 16 – Screenshot Dados do Balanceador - Servidores Ativos

```
Active servers = 3
-----
Servers:
http://ec2-52-26-51-200.us-west-2.compute.amazonaws.com:8000
http://ec2-35-167-36-190.us-west-2.compute.amazonaws.com:8000
http://ec2-35-167-32-77.us-west-2.compute.amazonaws.com:8000
```

Fonte: elaborada pela autora

Figura 17 – Screenshot Dados do Balanceador - Pesos mais recentes (RNA)

```
-----
Running load Balancer...
-----
Weights:
responseTime: 1047.5
state: 1
type: 1
distance: 1
```

Fonte: elaborada pela autora

5 RESULTADOS

Os experimentos realizados para verificar o desempenho do balanceador foram realizados em ambiente real, de maneira dinâmica através da execução dos três módulos da aplicação. Foram alocados três servidores do tipo t2.micro como servidores, um servidor t2.medium para alocar a aplicação de balanceamento e um computador para executar a aplicação para executar as requisições, no qual é armazenado o log das requisições. Definiu-se um total de 100 requisições para avaliar o desempenho do balanceador com a

Figura 18 – *Screenshot* Aplicação para testar o balanceador

```
D:\_git\ann-load-balance\test (master)
λ node index.js --help

Usage: index [options]

Options:
  -h, --help                output usage information
  -V, --version              output the version number
  -R, --roundrobin           Set Round Robin Output
  -P, --perceptron          Set perceptron output [DEFAULT]
  -O, --output [file]       Set a output log file[default: log.txt]
  -i, --iterations [iterations] Set a number of iterations [default: 1000]
  -u, --uri [uri]           Set a uri with port [default: http://localhost:8000]
```

Fonte: elaborada pela autora

Rede Neural. Para realizar os testes foi implementado uma aplicação na qual seleciona-se o tipo de algoritmo de balanceamento utilizado - no caso, a RNA ou Round Robin puro - número de iterações e url onde está hospedado o servidor de balanceamento (Figura 18). Esta aplicação armazena um arquivo de log com as informações de todas as requisições enviadas e as respostas (incluindo o servidor que atendeu a requisição).

A avaliação do desempenho da aplicação de balanceamento foi comparada ao balanceamento realizado com Round Robin puro, através da análise dos arquivos de log gerados em ambos os casos de teste, com o mesmo número de requisições com o qual foi testada a aplicação de RNA. É fato que a aplicação de balanceamento teve desempenho satisfatório, visto que conseguiu distribuir a carga das requisições entre os servidores com sucesso, sem apresentar uma diferença significativa em relação ao algoritmo de Round Robin puro.

Durante o processo foi possível notar que a característica que mais influenciou o processo de escolha foi a quantidade de requisições associadas ao servidor. Percebe-se também que servidores do mesmo tipo tendem a deixar o balanceamento mais equilibrado.

A tabela 3 apresenta a quantidade de requisições atribuídas a cada um dos k servidores alocados após executar 100 iterações na aplicação que gera requisições. Nos testes, os $k = 3$ servidores são tipo t2.micro. A comparação é feita entre o balanceamento

realizado com Round Robin, com a RNA sem treinamento e com a RNA após o aprendizado dos pesos.

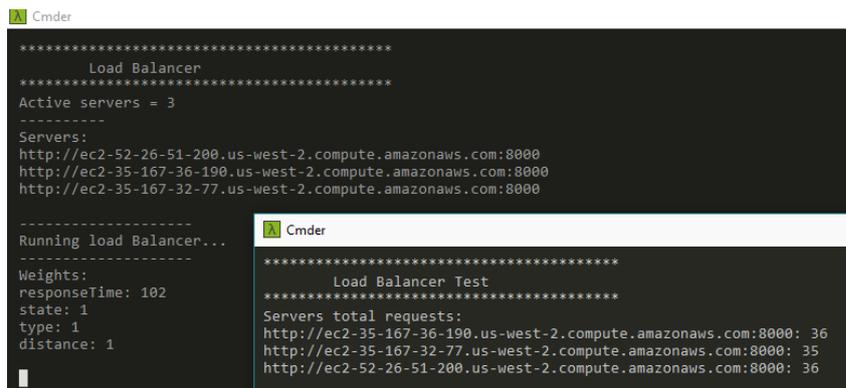
Tabela 3 – Comparação entre Distribuição de requisições entre servidores

Servidores	Round Robin	RNA	RNA - treinada
Servidor 1	34	36	37
Servidor 2	35	35	37
Servidor 3	34	36	32

Elaborado pela autora.

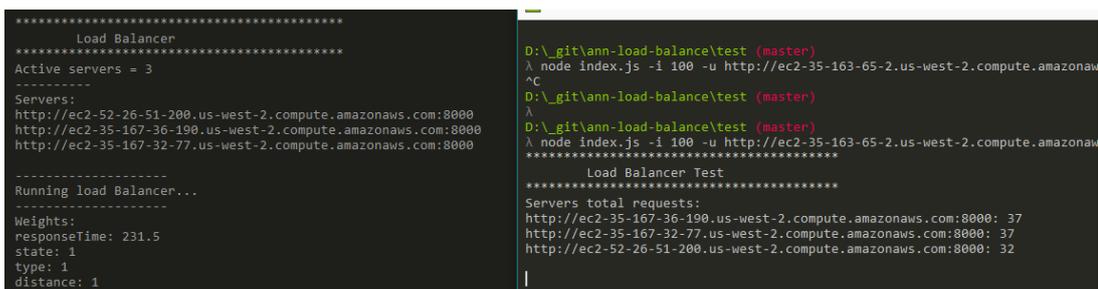
É possível notar que não há diferença significativa entre os dois algoritmos, em relação à atribuir requisições de forma desigual. Na última coluna da tabela, nota-se que o Servidor 3 recebeu um número menor de requisições, em relação aos demais. Isto pode ter ocorrido devido à oscilações na rede. As Figuras 19, 20 e 21 apresentam *screenshots* dos testes.

Figura 19 – Screenshot Teste com RNA



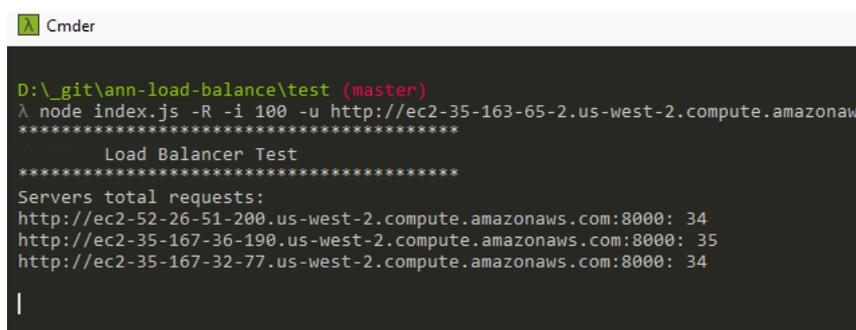
Fonte: elaborada pela autora

Figura 20 – Screenshot Teste com RNA já treinada



Fonte: elaborada pela autora

Figura 21 – Screenshot Teste com Round Robin



```
Cmder
D:\_git\ann-load-balance\test (master)
λ node index.js -R -i 100 -u http://ec2-35-163-65-2.us-west-2.compute.amazonaws.com
*****
Load Balancer Test
*****
Servers total requests:
http://ec2-52-26-51-200.us-west-2.compute.amazonaws.com:8000: 34
http://ec2-35-167-36-190.us-west-2.compute.amazonaws.com:8000: 35
http://ec2-35-167-32-77.us-west-2.compute.amazonaws.com:8000: 34
|
```

Fonte: elaborada pela autora

6 CONCLUSÃO

O presente trabalho apresenta uma aplicação de balanceamento de cargas utilizando Redes Neurais Artificiais como um módulo funcional de balanceamento de requisições. Seu desempenho foi analisado através da comparação com um algoritmo conhecido e bastante usado, o Round Robin.

Os testes não demonstram ganho significativo no uso da RNA. Portanto, nas condições em que os testes foram realizados, pode-se considerar o Round Robin como melhor alternativa, se levarmos em conta a simplicidade de sua implementação. Considerando o ambiente em que foram realizados os testes, que pode ser considerado estável e com servidores do mesmo tipo, o algoritmo de balanceamento com RNA não se destacou. Porém, foi possível perceber que a característica que mais influenciou a classificação foi o tempo de resposta dos servidores.

O modelo proposto neste trabalho leva em consideração os estados mais recentes dos servidores, levando em conta seu último estado, o que o torna mais dinâmico se comparado com o Round Robin. Esta implementação pode apresentar melhor desempenho em ambientes mais instáveis, visto que a distribuição das cargas se dá pela avaliação de estados como falhas de conexão e estado do servidor, por exemplo e seria interessante simular ambientes deste tipo em trabalhos futuros, para avaliar o desempenho da abordagem proposta.

6.1 Trabalhos futuros

Como sugestão de trabalhos futuros pode-se considerar analisar o desempenho da arquitetura desta aplicação de balanceamento em ambientes instáveis, com servidores distintos. Além disto, propor outras características e avaliar o desempenho com base nesta escolha. Em relação à rede, testar e avaliar o desempenho da aplicação com outras arquiteturas de Redes Neurais Artificiais, com mais camadas, por exemplo.

REFERÊNCIAS

- AMAZON. *Amazon Cloud Drive*. 2016. Disponível em: <<https://www.amazon.com/clouddrive/home>>. Acesso em: 14 Maio 2016.
- ARMBRUST, M. et al. A view of cloud computing. *Commun. ACM*, ACM, New York, NY, USA, v. 53, n. 4, p. 50–58, abr. 2010. ISSN 0001-0782. Disponível em: <<http://doi.acm.org/10.1145/1721654.1721672>>.
- BULLINARIA, J. A. *Recurrent Neural Networks - Neural Computation : Lecture 12*. 2015. Disponível em: <<http://www.cs.bham.ac.uk/~jxb/INC/l12.pdf>>. Acesso em: 18 Janeiro 2016.
- CORTES, C.; VAPNIK, V. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995.
- DUDA, R. O.; HART, P. E.; STORK, D. G. *Pattern Classification (2nd Edition)*. [S.l.]: Wiley-Interscience, 2000. ISBN 0471056693.
- EC2, A. W. S. *Definição de preço do Amazon EC2*. 2016. Disponível em: <<https://aws.amazon.com/pt/ec2/pricing/>>. Acesso em: 14 Maio 2016.
- HAYKIN, S. *Neural Networks: A comprehensive foundation*. [S.l.]: Prentice-Hall, 1998.
- HAYKIN, S. *Neural Networks: A Comprehensive Foundation (2rd Edition)*. [S.l.]: Prentice-Hall, Inc., 1999.
- HAYKIN, S. *Neural Networks: A Comprehensive Foundation (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2007. ISBN 0131471392.
- IBM Cloud. *Balanceamento de Carga*. 2016. Disponível em: <<http://www.softlayer.com/pt-br/load-balancing>>. Acesso em: 14 Maio 2016.
- KAUR, R.; LUTHRA, P. Load balancing in cloud computing. In: *Proceedings of International Conference on Recent Trends in Information, Telecommunication and Computing, ITC*. [S.l.: s.n.], 2012.
- LAB., L. *Multilayer Perceptron Copyright 2008–2010*. 2016. Disponível em: <<http://deeplearning.net/tutorial/mlp.html>>. Acesso em: 04 Janeiro 2017.
- MELL, P.; GRANCE, T. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg, 2011.
- MICROSOFT. *Microsoft Azure*. 2016. Disponível em: <<https://azure.microsoft.com/pt-br/>>. Acesso em: 14 Maio 2016.
- NGINX. *What is load balancing*. 2017. Disponível em: <<https://www.nginx.com/resources/glossary/load-balancing/>>. Acesso em: 09 Janeiro 2017.
- NGINX. *WHAT IS ROUND-ROBIN LOAD BALANCING?* 2017. Disponível em: <<https://www.nginx.com/resources/glossary/round-robin-load-balancing/>>. Acesso em: 09 Janeiro 2017.

NISHANT, K. et al. Load balancing of nodes in cloud using ant colony optimization. In: IEEE. *Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on*. [S.l.], 2012. p. 3–8.

NODEJS. *About Node.js®*. 2017. Disponível em: <<https://nodejs.org/en/about/>>. Acesso em: 11 Janeiro 2017.

NUAIMI N. MOHAMED, M. A. N. J. A.-J. K. A. A survey of load balancing in cloud computing: Challenges and algorithms. *Second Symposium on Network Cloud Computing and Applications*, IEEE Computer Society, p. 137 – 142, 2012.

OPENSTACK. *OpenStack*. 2016. Disponível em: <<https://www.openstack.org/>>. Acesso em: 14 Maio 2016.

PANDEY, S. et al. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In: IEEE. *Advanced information networking and applications (AINA), 2010 24th IEEE international conference on*. [S.l.], 2010. p. 400–407.

PBXL. *SaaS, PaaS, IaaS?* 2017. Disponível em: <<http://pbxl.co.jp/en/saas-paas-iaas/>>. Acesso em: 09 de Janeiro 2017.

RACKSPACE. *How Do Cloud Load Balancers Work*. 2017. Disponível em: <<http://es.rack.ly/cloud/load-balancers/how-it-works>>. Acesso em: 09 de Janeiro 2017.

RANGLES, M.; LAMB, D.; TALEB-BENDIAB, A. A comparative study into distributed load balancing algorithms for cloud computing. In: IEEE. *Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on*. [S.l.], 2010.

RAZA, M. H. et al. The slow adoption of cloud computing and {IT} workforce. *Procedia Computer Science*, v. 52, p. 1114 – 1119, 2015. ISSN 1877-0509. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015). Disponível em: <<http://www.sciencedirect.com/science/article/pii/S187705091500928X>>.

RIMAL, B. P.; CHOI, E.; LUMB, I. A taxonomy and survey of cloud computing systems. *Networked Computing and Advanced Information Management, International Conference on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 44–51, 2009.

SALLAMI, N. M. A.; ALOUSI, S. A. A. Load balancing with neural network. *IJACSA International Journal of Advanced Computer Science and Applications*, Citeseer, v. 4, n. 10, p. 138 – 145, 2013.

SERVICES, A. W. *EC2*. 2016. Disponível em: <<https://aws.amazon.com/pt/ec2/>>. Acesso em: 14 Maio 2016.

SERVICES, A. W. *Hospedagem de servidor virtual*. 2016. Disponível em: <<http://aws.amazon.com/pt/ec2/>>. Acesso em: 14 Maio 2016.

SERVICES, A. W. *Produtos em Nuvem*. 2016. Disponível em: <https://aws.amazon.com/pt/products/?nc2=h_ql_ny_livestream_blu>. Acesso em: 14 Maio 2016.

SERVICES, A. W. *Tipos de instância do Amazon EC2*. 2016. Disponível em: <<https://aws.amazon.com/pt/ec2/instance-types/>>. Acesso em: 10 Janeiro 2016.

- SERVICES, A. W. *Bancos de Dados na nuvem com AWS*. 2017. Disponível em: <<https://aws.amazon.com/pt/products/databases/>>. Acesso em: 10 Janeiro 2017.
- SERVICES, A. W. *Internet das Coisas*. 2017. Disponível em: <<https://aws.amazon.com/pt/iot/>>. Acesso em: 10 Janeiro 2017.
- SONI, G.; KALRA, M. A novel approach for load balancing in cloud data center. In: IEEE. *Advance Computing Conference (IACC), 2014 IEEE International*. [S.l.], 2014. p. 807–812.
- SPECHT, D. F. Probabilistic neural networks. p. 109–118, 1990.
- SPECHT, D. F. Probabilistic neural networks. *Neural networks*, Elsevier, v. 3, n. 1, p. 109–118, 1990.
- SPECHT, D. F. Enhancements to probabilistic neural networks. p. 761–768, 1992.
- SRAN, N.; KAUR, N. Comparative analysis of existing dynamic load balancing techniques. *International Journal of Computer Applications*, Foundation of Computer Science, v. 70, n. 26, 2013.
- STACK, A. C. *Cloud Stack*. 2016. Disponível em: <<https://cloudstack.apache.org/>>. Acesso em: 14 Maio 2016.
- TALEB-BENDIAB, M. R. . D. L. . A. A comparative study into distributed load balancing algorithms for cloud computing. *Advanced Information Networking and Applications Workshops (WAINA)*, p. 551 – 556, 2010.
- WIKIPEDIA. *File Colored neural network*. 2013. Disponível em: <https://en.wikipedia.org/wiki/Artificial_neural_network#/media/File:Colored_neural_network.svg>. Acesso em: 10 Janeiro 2017.
- ZHANG, Q.; CHENG, L.; BOUTABA, R. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, Springer, v. 1, n. 1, p. 7–18, 2010.