

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Ian Resende da Cunha

**Construção de Mecanismo para Suportar a
Predição de Tempos de Resposta do Cassandra
a Partir de Métricas de Infraestrutura**

Uberlândia, Brasil

2019

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Ian Resende da Cunha

**Construção de Mecanismo para Suportar a Predição de
Tempos de Resposta do Cassandra a Partir de Métricas
de Infraestrutura**

Trabalho de conclusão de curso apresentado
à Faculdade de Computação da Universidade
Federal de Uberlândia, Minas Gerais, como
requisito parcial exigido à obtenção do grau
de Bacharel em Ciência da Computação.

Orientador: Rafael Pasquini

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Ciência da Computação

Uberlândia, Brasil

2019

Agradecimentos

Agradeço primeiramente aos meus pais Márcio e Edilaine, por todo apoio e incentivo em todas as situações e por não medirem esforços para me proporcionar a melhor educação possível, tanto pessoal quanto acadêmica.

A toda minha família, por sempre me apoiar e sempre desejar o melhor para mim.

A minha namorada Isadora, por estar sempre ao meu lado e por toda a paciência e incentivo nos momentos mais difíceis.

Aos meus amigos que estiveram comigo durante essa trajetória, pelo companheirismo e por me ajudar em vários momentos.

Aos professores da Faculdade de Computação, por todos os ensinamentos passados, experiências compartilhadas e pela atenção dada aos alunos.

Ao meu orientador Professor Doutor Rafael Pasquini, por confiar em mim e nesse trabalho, e pela paciência e atenção durante toda a execução do trabalho.

Resumo

Neste trabalho é realizado um estudo em torno de um serviço de armazenamento de dados não relacional, buscando a construção de um mecanismo que consiga estimar o valor de métricas de qualidade de serviço, a partir de estatísticas da infraestrutura do serviço. Essa predição pode possibilitar que degradações na qualidade do serviço sejam identificadas em tempo hábil de se efetuar contramedidas preventivas. Um ambiente de testes foi implementado para simular interações entre o serviço e clientes, ao mesmo tempo em que foram coletados os conjuntos de dados referentes às métricas de qualidade de serviço e estatísticas de infraestrutura. Experimentos foram realizados no sentido de induzir padrões de carga no serviço e observar a oscilação causada na qualidade de serviço recebida por um cliente. Com os conjuntos de dados necessários extraídos, um método de aprendizado de máquina foi utilizado para gerar um modelo que conseguisse, a partir de amostras referentes ao estado da infraestrutura do serviço, predizer os valores de uma métrica de qualidade de serviço que um cliente receberia. Demonstramos que o efeito causado pelas variações de carga são claramente perceptíveis na observação de um cliente e que é possível criar um modelo que estima valores de métricas de qualidade de serviço a partir de estatísticas de infraestrutura. Finalmente, demonstramos também que é possível reduzir drasticamente o conjunto de estatísticas utilizadas no treinamento do modelo de aprendizado de máquina sem que haja degradação na qualidade da estimação.

Palavras-chave: Predição de Métricas, Apache Cassandra, Métricas de Qualidade de Serviço, Aprendizado de Máquina.

Lista de ilustrações

Figura 1 – Modelo de Dados do Cassandra. Extraído de (PERERA, 2012)	17
Figura 2 – Arquitetura do Ambiente de Teste Cassandra Projetado.	24
Figura 3 – Organização dos Nós do Cluster Cassandra.	25
Figura 4 – Trecho do Conjunto de Métricas de QoS Coletados pela Máquina Cliente.	27
Figura 5 – Tempo Médio das Operações de Escrita e Leitura Amostrado Durante Experimento com Padrão de Escada.	29
Figura 6 – Tempo Médio das Operações de Escrita Amostrado Durante Experimento de Duas Horas com Padrão Sinusoidal.	31
Figura 7 – Parte do Conjunto X de Dados Coletado Durante um Experimento.	32
Figura 8 – Tempo Médio das Operações de Escrita Amostrado Durante Experimento de Quatro Horas com Padrão Sinusoidal.	33
Figura 9 – Tempo Médio de Escrita Amostrado e Tempo Médio de Escrita Estimado.	35

Lista de tabelas

Tabela 1 – Tempo Médio de Escrita em Diferentes Níveis de Carga.	29
Tabela 2 – NMAE e Tempo de Treinamento na Predição do Tempo Médio de Escrita e Leitura.	35
Tabela 3 – Seis Primeiros Valores Estimados e Respectivos Valores Reais.	35
Tabela 4 – NMAE e Tempo de Treinamento Obtidos para Diferentes Valores de K.	36
Tabela 5 – Dez Métricas com Melhor Classificação no Ranking do treinamento do Tempo Médio de Resposta de Escrita.	37

Lista de abreviaturas e siglas

API	Application Programming Interface
ACID	Atomicity, Consistency, Isolation, Durability
BASE	Basically Available, Soft State, Eventually Consistency
CPU	Central Processing Unit
CAP	Consistency, Availability, Partition Tolerance
DHT	Distributed Hash Table
JSON	JavaScript Object Notation
MAE	Mean Absolute Error
NMAE	Normalized Mean Absolute Error
NTP	Network Time Protocol
NoSQL	No SQL
QoS	Quality of Service
RAM	Random Access Memory
RPC	Remote Procedure Call
SGBD	Sistema de Gerenciamento de Banco de Dados
TCP	Transmission Control Protocol

Sumário

1	INTRODUÇÃO	8
1.1	Objetivos	9
1.1.1	Objetivo Principal	9
1.1.2	Objetivos Específicos	9
1.2	Método	9
1.3	Organização do Trabalho	10
2	FUNDAMENTAÇÃO TEÓRICA	11
2.1	Banco de Dados NoSql	11
2.2	Apache Cassandra	12
2.2.1	Arquitetura	13
2.2.2	Comunicação entre os Nós (<i>gossip</i>)	14
2.2.3	Replicação e Distribuição dos Dados	14
2.2.4	Particionador	15
2.2.5	<i>Snitch</i>	15
2.2.6	Modelo de dados	16
2.3	Aprendizado de Máquina	18
2.4	OpenStack	19
2.5	Trabalhos Relacionados	20
3	DESENVOLVIMENTO	23
3.1	Infraestrutura	24
3.2	Cluster Cassandra	25
3.3	Cliente	26
3.4	Gerador de Carga	27
3.5	Definindo Limites para o Padrão de Carga	28
3.6	Averiguação do Funcionamento do Gerador de Carga	30
3.7	Coleta de Métricas de Infraestrutura	31
3.8	Experimento para Coleta dos Conjuntos de Métricas	32
3.9	Predição dos Tempos de Resposta do Cassandra	33
3.10	Resultados	34
4	CONCLUSÃO	38
	REFERÊNCIAS	40

1 Introdução

Com o crescente volume de dados que precisam ser armazenados e manipulados por grandes empresas, observado principalmente após a popularização de serviços online, como redes sociais, sites de compra e serviços de nuvem (OUSSOUS et al., 2018), uma adaptação na maneira como esses dados eram armazenados e manipulados fez-se necessária. O que antes era armazenado em banco de dados relacionais, que manipulavam o crescente volume de dados com cada vez mais dificuldade e lentidão, começou a ser migrado para banco de dados não relacionais (MONIRUZZAMAN; HOSSAIN, 2013). Estes bancos de dados manipulam os dados de maneira diferente, são muitas vezes especializados para um modelo de dados e aceitam adaptações, permitindo a manipulação de grandes volumes de dados e maior escalabilidade do serviço.

O Cassandra é um sistema de armazenamento estruturado, distribuído e altamente escalável, projetado para manipular grandes quantidades de dados. Um cliente Cassandra pode realizar operações de escrita e leitura em um cluster do Cassandra, o tempo médio de resposta do Cassandra é calculado a cada segundo, sendo a média de todos os tempos de resposta das operações realizadas naquele segundo. A carga total em um cluster do Cassandra, que é a quantidade de operações requisitadas em certo momento, varia conforme clientes se conectam e se desconectam do cluster. Quando a carga de operações atinge um nível que o cluster não está preparado para atender, espera-se que a latência das operações também aumente. Assim, picos de carga em clusters despreparados podem ocasionar uma perda da qualidade do serviço oferecido.

O principal problema envolvido neste projeto é a dificuldade que se tem de prever o comportamento do cluster Cassandra, ou seja, saber qual será o tempo médio de resposta de um conjunto de operações em um certo momento. Visando amenizar o impacto de uma sobrecarga no Cluster Cassandra ao cliente, faz-se necessário o monitoramento e estudo do comportamento de um ambiente Cassandra frente às variações de carga que se observa em situações reais, para posteriormente, utilizando os dados observados, encontrar um meio de tornar o sistema mais determinístico.

Espera-se que com a utilização de um método de aprendizado de máquina seja possível gerar um modelo que consiga realizar previsões de métricas de qualidade de serviço, a partir das estatísticas de infraestrutura do cluster do serviço Cassandra no dado momento.

Assim, este projeto se trata da pesquisa e experimentação em torno da ferramenta Cassandra visando a previsão da latência de suas operações, para que seja possível saber quando degradações podem ocorrer, permitindo efetuar contra-medidas preventivas, tais

como elasticidade do serviço e/ou da infraestrutura.

1.1 Objetivos

1.1.1 Objetivo Principal

O objetivo principal do trabalho é construir um protótipo capaz de demonstrar que é possível treinar, com qualidade, um modelo de aprendizado de máquina capaz de prever a latência para uma dada operação no Cassandra conforme o estado das métricas do cluster no instante em que a operação é requisitada.

1.1.2 Objetivos Específicos

- Construção de um ambiente de testes, com um cluster do Cassandra, um módulo para simular um cliente Cassandra e um módulo para simular uma carga variável (diversos clientes);
- Durante a execução de experimentos no ambiente de teste, gerar um arquivo de log que guarde estatísticas referentes às operações realizadas a cada segundo pelo cliente. Ao mesmo tempo, extrair métricas referentes ao uso de recursos da infraestrutura do cluster Cassandra, a cada segundo;
- Encontrar um nível de carga do cliente e do gerador de carga que no resultado de um experimento evidenciem o impacto causado pela variação de carga na performance do cliente;
- A partir do log das operações realizadas e das métricas de infraestrutura, gerar um modelo de aprendizado de máquina capaz de prever a latência de novas operações.

1.2 Método

Na primeira parte do trabalho, a metodologia abordada foi a de experimentação. Um ambiente Cassandra foi criado, possuindo um cluster do sistema, uma máquina simulando operações de apenas um cliente para analisar a qualidade do serviço do banco e uma máquina simulando vários clientes para aplicar cargas variadas no cluster.

Experimentos foram realizados com o intuito de se encontrar uma configuração de carga equilibrada, no cliente e no gerador de carga, que não fosse pesada demais e nem leve demais para o cluster. A cada experimento o resultado foi analisado e comparado ao resultado de experimentos anteriores. A partir dessa análise e comparação, o próximo experimento era planejado.

Com a configuração de carga equilibrada encontrada, mais experimentos foram realizados, dessa vez aplicando padrões de carga variados no cluster, para se observar o efeito causado na performance de um cliente. Durante esses experimentos, além dos dados referentes à qualidade do serviço no cliente, também foram extraídas métricas de infraestrutura referentes ao desempenho do cluster.

De posse dos conjuntos de dados referentes à latência média das operações e ao estado da infraestrutura do cluster, um método de aprendizado de máquina por regressão é aplicado buscando obter um modelo que consiga prever os tempos de resposta das operações realizadas por um cliente com base no estado da infraestrutura do cluster do Cassandra.

1.3 Organização do Trabalho

O decorrer deste trabalho é organizado da seguinte forma: o Capítulo 2 apresenta alguns trabalhos relacionados e conceitos básicos, como aprendizado de máquina, banco de dados não relacional, o Cassandra e OpenStack; o Capítulo 3 detalha a construção do ambiente Cassandra bem como a estrutura de monitoramento, os experimentos realizados, além de apresentar os resultados obtidos; e, por fim, o Capítulo 4 traz as conclusões e trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo são apresentados os conceitos básicos necessários para compreensão deste trabalho, bem como os principais trabalhos relacionados.

2.1 Banco de Dados NoSql

No SQL (NoSQL) é um termo genérico usado para descrever uma grande classe de bancos de dados que não possui propriedades de bancos de dados relacionais, e fornece um mecanismo para armazenamento e recuperação de dados que são modelados de formas diferentes das usadas nos bancos de dados relacionais (DAVOUDIAN; CHEN; LIU, 2018). Sistemas NoSQL são sistemas de gerenciamento de banco de dados (SGBD) não relacionais e distribuídos, projetados para o armazenamento de dados em larga escala e suportam o processamento paralelo de dados em servidores distribuídos (MONIRUZZAMAN; HOSSAIN, 2013).

Os sistemas NoSQL foram difundidos por grandes empresas de tecnologia, como Google, Amazon e Facebook. Após encontrarem dificuldades em lidar com o crescente volume de dados utilizando banco de dados relacionais, essas grandes empresas criaram suas próprias soluções NoSQL como o Dynamo (Amazon), BigTable (Google) e Cassandra (Facebook), que apesar de surgirem com o intuito de atender necessidades específicas dessas empresas, foram adotados e difundidos globalmente (MONIRUZZAMAN; HOSSAIN, 2013).

Os SGBD's clássicos seguem as propriedades do ACID (Atomicidade, Consistência, Isolamento e Durabilidade) para garantir a consistência e integridade dos dados (MONIRUZZAMAN; HOSSAIN, 2013). Em busca de escalabilidade e alta performance os sistemas NoSQL abrem mão do ACID, utilizando por sua vez as propriedades BASE, onde a aplicação deve trabalhar basicamente o tempo todo (*basically available*), não precisa ser consistente o tempo todo (*eventual consistency*) mas deve estar eventualmente em um estado conhecido (*soft-state*) (DAVOUDIAN; CHEN; LIU, 2018).

O teorema CAP define que um sistema pode atender somente duas das três seguintes propriedades: consistência forte, alta disponibilidade e tolerância a partição, de modo que os sistemas NoSQL tendem a optar pela alta disponibilidade e tolerância a partição. As propriedades do CAP são explicadas a seguir (DAVOUDIAN; CHEN; LIU, 2018):

- Consistência forte: todos os clientes veem a mesma versão dos dados, mesmo em atualizações do conjunto de dados. Um sistema distribuído é normalmente conside-

rado consistente se após uma operação de atualização de algum gravador, todos os leitores veem suas atualizações em alguma fonte de dados compartilhada;

- Alta disponibilidade: todos os clientes podem sempre encontrar pelo menos uma cópia dos dados solicitados, mesmo que algumas das máquinas em um cluster estejam desativadas. O que significa que um sistema é projetado e implementado de forma a permitir que continue a operação (ou seja, permitindo operações de leitura e gravação) se os nós em um cluster falharem ou algumas peças de *hardware* ou *software* estiverem inativas devido a atualizações;
- Tolerância à partição: o sistema mantém sua característica mesmo quando está sendo implantado em servidores diferentes, transparente para o cliente. Capacidade do sistema para continuar a operação na presença de partições de rede. Isso ocorre se duas ou mais “ilhas” de nós de rede surgirem, que (temporária ou permanentemente) não podem se conectar entre si. Algumas pessoas também entendem a tolerância à partição como a capacidade de um sistema lidar com a adição dinâmica e a remoção de nós (por exemplo, para fins de manutenção; os nós removidos e novamente adicionados são considerados uma partição de rede própria nessa noção).

Os bancos de dados NoSQL possuem uma grande diversidade de características, cada implementação tem a sua peculiaridade, fazendo com que a classificação desses sistemas NoSQL não seja uma tarefa simples. Em [Leavitt \(2010\)](#), Leavitt expõe as três classificações de sistemas NoSQL mais populares:

- Armazenamentos de chave-valor: É um sistema que armazena um valor indexado por uma chave, que serve para buscar aquele valor. Os bancos de dados de chave-valor são altamente particionáveis e permitem escalabilidade horizontal em escalas que outros tipos de bancos de dados não conseguem alcançar;
- Armazenamentos orientados a coluna: Esses sistemas armazenam dados em colunas extensíveis que podem ser particionadas vertical e horizontalmente entre os nós. São projetados para processar grandes quantidades de dados e recuperar rapidamente as colunas de dados;
- Armazenamentos baseados em documentos: Armazena os dados em coleções de documentos. Os dados são representados como um objeto ou um documento do tipo JSON, por ser um modelo de dados eficiente e intuitivo.

2.2 Apache Cassandra

O Apache Cassandra é um sistema de armazenamento estruturado, distribuído e altamente escalável, projetado para manipular grandes quantidades de dados, além de

oferecer um serviço rápido, altamente disponível e sem ponto único de falha, apto para rodar em infraestruturas de centenas de nós, que podem estar espalhados por diferentes datacenters ([LAKSHMAN; MALIK, 2010](#)).

Apesar de apresentar características em comum com banco de dados convencionais, o Cassandra não suporta um modelo de dados relacional, ele trabalha com um modelo de dados simples que permite um controle dinâmico sobre a estrutura e o formato dos dados ([LAKSHMAN; MALIK, 2010](#)).

O dados no Cassandra são automaticamente distribuídos e replicados entre os nós que participam do anel ou cluster. A replicação dos dados, que é o armazenamento de cópias redundantes de dados entre os nós do cluster, torna o sistema tolerante a falhas ([Apache, 2016](#)), uma vez que se algum dos nós do cluster for comprometido, todos os dados que aquele nó armazenava ainda estarão disponíveis em outros nós em funcionamento.

As principais características do Cassandra segundo ([Apache, 2016](#)), são:

- Tolerância a falhas: Dados são replicados para múltiplos nós.
- Escalabilidade: As maiores implementações em produção incluem mais de 75000 nós armazenando mais de 10PB de dados;
- Descentralização: Não há gargalos na rede e não admite ponto único de falha, ou seja, um ponto de falha não pode causar a falha de todo o sistema.
- Elástico: A taxa de transferência de escrita e leitura aumentam linearmente conforme novos nós são adicionados;
- Durável: Pode ser usado por aplicações que não admitem a perda de dados, mesmo quando todo o datacenter sofre uma queda;

2.2.1 Arquitetura

Sua arquitetura se baseia no princípio de que falhas de sistema e de *hardware* podem e irão acontecer. Assim, o Cassandra trata as falhas através do emprego de um sistema distribuído *peer-to-peer* onde os dados são distribuídos entre todos os nós do cluster. Cada nó troca informações sobre ele mesmo e sobre outros nós do cluster, utilizando o protocolo de comunicação *gossip* ([DataStax, 2019](#)).

A arquitetura do Cassandra permite que qualquer usuário autorizado se conecte a qualquer nó. Quando um cliente se conecta a um nó e realiza uma requisição, este nó funciona como um coordenador para essa operação. O coordenador age como um *proxy* entre o cliente e os nós que possuem os dados que estão sendo requisitados. Com base na configuração do cluster, o coordenador determina quais nós do anel devem receber e resolver a requisição ([DataStax, 2019](#)).

2.2.2 Comunicação entre os Nós (*gossip*)

O *gossip* é um protocolo de comunicação *peer-to-peer* onde os nós trocam informações sobre eles mesmos e sobre nós conhecidos. O processo *gossip* é executado a todo segundo e troca mensagens com até três outros nós do cluster, assim rapidamente todos os nós aprendem sobre todos os outros nós conectados ao cluster (DataStax, 2019).

Quando um nó é iniciado pela primeira vez ele precisa se juntar ao cluster, então o arquivo de configuração do nó é utilizado. Nele há uma lista com alguns pontos de conexão com o cluster, esse pontos iniciais de conexão são os nós *seed* do cluster (LAKSHMAN; MALIK, 2010). Os nós *seed* são os designados para iniciar o processo de *gossip* com novos nós (DataStax, 2019), ou seja, passam informações sobre todos os nós do cluster para os novos nós e recebe informações desse novo nó, repassando para todo o cluster.

2.2.3 Replicação e Distribuição dos Dados

No Cassandra a distribuição e replicação dos dados estão relacionados. Os dados são organizados por tabelas ou famílias de colunas e identificados por uma chave primária que determina em qual nó o dado está armazenado. As réplicas são cópias de um linha, ou seja, cópias de um dado inserido no Cassandra (DataStax, 2019).

Para a minimizar a reorganização de dados quando nós são adicionados ou removidos em um cluster é utilizado o mecanismo de *hashing* consistente na distribuição dos dados (DataStax, 2019). No *hashing* consistente o intervalo de saída de uma função hash é tratado como um anel, ou seja, o maior valor hash da sequência é seguido pelo menor valor hash. Cada nó do sistema recebe um valor aleatório dentro desse espaço, representando sua posição no anel. Cada dado inserido deve ser atribuído a um nó, isso acontece ao aplicar uma função hash sobre a chave do dado, o hash gerado indica a posição do dado no anel, o nó responsável pelo dado é o primeiro no sentido horário com a posição maior que o dado (LAKSHMAN; MALIK, 2010).

O Cassandra utiliza a replicação dos dados para garantir alta disponibilidade e durabilidade. Cada dado pode ser replicado em diversos nós, essa quantidade de nós em que cada dado será replicado é chamado de fator de replicação e pode ser alterado na configuração de um keyspace, todas as réplicas possuem o mesmo grau de importância para o cluster (LAKSHMAN; MALIK, 2010). A maneira como os dados serão replicados nos nós do cluster é definida pela estratégia de replicação.

A estratégia de replicação *SimpleStrategy* deve ser utilizada em arquiteturas de um único datacenter. Esta estratégia armazena a primeira réplica no nó determinado pelo particionador e as demais réplicas são armazenadas nos seguintes nós no sentido horário no anel sem considerar a topologia de rede (DataStax, 2019).

A estratégia *NetworkTopologyStrategy* deve ser utilizada quando o cluster é, ou

será, implantado em vários datacenters. Esta estratégia permite especificar quantas réplicas serão colocadas em cada datacenter. O armazenamento das réplicas definidas para um datacenter é feito percorrendo o sentido horário do anel e escolhendo nós em diferentes racks, pois nós no mesmo rack podem ficar indisponíveis ao mesmo tempo devido problemas de energia, refrigeração ou rede ([DataStax, 2019](#)).

2.2.4 Particionador

Um particionador determina como os dados serão distribuídos entre os nós do cluster, incluindo as réplicas. Basicamente, um particionador é uma função *hash* que calcula o *token* de uma chave de partição. Cada linha de dados é identificada por uma única chave de partição e distribuída no cluster pelo o valor do *token* ([DataStax, 2019](#)).

2.2.5 Snitch

O *snitch* determina a quais datacenters e racks os nós pertencem. O *snitch* informa o Cassandra sobre a topologia da rede, de modo que as requisições são encaminhadas de forma eficiente e permite distribuir as réplicas por agrupamentos de máquinas em datacenters e racks. O Cassandra tenta não possuir mais do que uma réplica no mesmo rack ([DataStax, 2019](#)).

Abaixo são listados os tipos de *snitches* disponíveis ([DataStax, 2019](#)):

- *SimpleSnitch*: É o *snitch* padrão, ele não reconhece informações de datacenters ou racks e é usado em implementações de datacenter único;
- *Snitching* dinâmico: Monitora o desempenho de leituras das réplicas e escolhe a melhor réplica com base no histórico;
- *RackInferringSnitch*: Este *snitch* determina a localização dos nós pelo endereço IP, que assume que o 3º octeto do IP corresponde ao rack e 2º octeto ao datacenter;
- *PropertyFileSnitch*: Este *snitch* determina a localização dos nós usando um arquivo de configurações definido pelo usuário, que contém os detalhes da rede;
- *GossipingPropertyFileSnitch*: Automaticamente atualiza todos os nós usando o *gossip*, quando novos nós são adicionados. É recomendado para ambientes de produção;
- *EC2Snitch*: Este *snitch* é usado para implementações utilizando Amazon EC2, onde todos os nós do cluster estão dentro de uma única região;
- *EC2MultiRegionSnitch*: Este *snitch* é usado para implementações utilizando Amazon EC2, onde um cluster abrange várias regiões.

2.2.6 Modelo de dados

O modelo de dados do Cassandra se baseia na orientação a coluna. Segundo [Lakshman e Malik \(2010\)](#) uma tabela no Cassandra é um mapa multi dimensional distribuído indexado por uma chave, e o valor armazenado é um objeto altamente estruturado.

O modelo de dados é formado por uma estrutura de *keyspace* que contém tabelas, ou famílias de colunas, que são contêineres para um conjunto ordenado de linhas, onde os dados são armazenados, e cada linha é uma coleção ordenada de colunas. Na Figura 1 o modelo de dados do Cassandra é ilustrado.

Um cluster do Cassandra deve conter pelo menos um *keyspace*, ele é a estrutura mais externa para os dados do Cassandra e cada *keyspace* possui um nome e deve ser configurado para se comportar da maneira desejada. É recomendado a utilização de apenas um *keyspace* por aplicação ([HEWITT, 2010](#)).

Segundo [Hewitt \(2010\)](#), os atributos básicos de um *keyspace* que devem ser configurados são:

- Fator de replicação: Indica em quantos nós do cluster o mesmo dado será replicado. Essa configuração permite decidir entre maior desempenho ou maior consistência do cluster, uma vez que um maior número de réplicas torna o armazenamento mais consistente e em contrapartida diminui o desempenho do cluster.
- Estratégias de réplicas: Define a forma como as réplicas serão posicionadas no anel de nós. As estratégias mais populares são *SimpleStrategy* e *NetworkTopologyStrategy*.
- Família de Colunas: Um *keyspace* deve possuir uma ou mais famílias de colunas (tabelas), que representam a estrutura dos dados. Uma família de colunas é formada por um conjunto de linhas e cada linha possui uma chave e uma coleção ordenada de colunas.

Uma família de colunas é um contêiner para uma coleção ordenada de linhas, sendo cada linha uma coleção ordenada de colunas. É possível adicionar e remover colunas de famílias de colunas conforme a necessidade. ([HEWITT, 2010](#)).

Uma linha pertencente a uma família de colunas é formada por uma coleção de valores, cada valor referente à uma coluna contida na família de colunas, juntamente com um identificador único, conhecido como a chave da linha ou *row key*, e funciona como uma chave primária única que identifica aquela linha. As famílias de colunas são armazenadas em arquivos diferentes no disco, o que torna importante manter colunas relacionadas definidas juntas em uma mesma família de colunas ([HEWITT, 2010](#)).

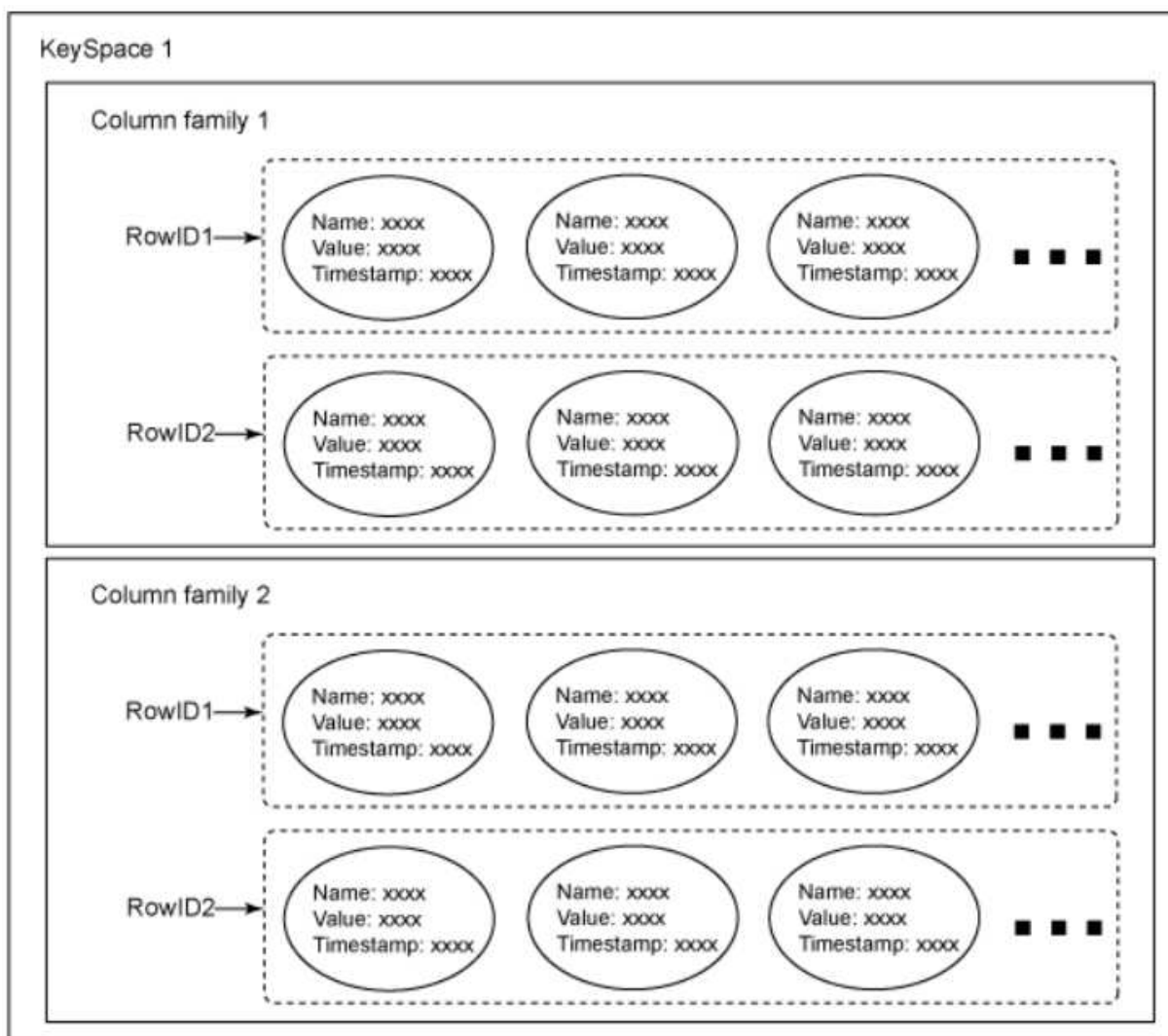


Figura 1 – Modelo de Dados do Cassandra. Extraído de (PERERA, 2012)

Na Figura 1, é possível observar duas famílias de colunas “*Column family 1*” e “*Column family 2*”, que possuem duas linhas de dados cada, indicadas pelas chaves *RowID1* e *RowID2*. Cada linha é formada por alguns valores de colunas.

O exemplo a seguir, extraído de (HEWITT, 2010), representa uma família de colunas denominada **Hotel** em uma notação parecida com JSON para melhor entendimento:

```
Hotel {
  key: AZC_043 { name: Cambria Suites, address: 400 N. Hayden Rd., city: Scottsdale, state: AZ}
  key: CAS_021 { name: W Hotel, address: 181 3rd Street, city: San Francisco, state: CA}
  key: NYN_042 { name: Waldorf Hotel, address: 301 Park Ave, city: New York, state: NY}
}
```

Neste exemplo, a *row key* é uma chave primária única para cada hotel, e as colunas são *name*, *phone*, *address*, *city*, *state*, e *zip*. Apesar de todas as linhas definirem valores para as mesmas colunas, seria possível ter linhas definindo valores para uma quantidade

maior ou menor de colunas.

2.3 Aprendizado de Máquina

O aprendizado de máquina, ou *machine learning*, é um ramo da inteligência artificial fundamentado na concepção de que sistemas computacionais podem aprender com dados e estatísticas, identificar padrões e tomar decisões com pouca ou nenhuma intervenção humana, permitindo que as aplicações se tornem mais inteligentes ao realizar análises estatísticas e predições.

Os algoritmos de aprendizado de máquina podem ser divididos em 3 categorias: Aprendizado Supervisionado, Aprendizado Não-supervisionado e Aprendizado Semi-supervisionado.

Os algoritmos de Aprendizado Supervisionado são treinados por meio de exemplos rotulados, ou seja, cada entrada tem a saída desejada conhecida. Na etapa de treino o algoritmo recebe um conjunto de entradas de treino juntamente com o respectivo conjunto de saídas. O algoritmo analisa a relação existente entre entrada e saída, gerando um modelo ou função capaz de estimar os rótulos de um novo conjunto de dados de entrada. Na fase de teste o modelo gerado é utilizado para estimar valores de saída a partir de entradas do conjunto de teste. Essas estimativas são comparadas aos os valores reais, permitindo avaliar o desempenho do algoritmo (FULMARI; CHANDAK, 2013);

O algoritmo de Aprendizado Não-Supervisionado deve aprender com um conjunto de dados coletados sem que haja qualquer informação sobre esses dados, ou seja, não se conhece os respectivos rótulos dos dados coletados. O algoritmo explora o conjunto de dados, tentando identificar padrões e formar agrupamentos entre os dados (FULMARI; CHANDAK, 2013). São utilizados para segmentar tópicos de texto, recomendar itens e identificar pontos discrepantes nos dados;

O algoritmo de Aprendizado Semi-supervisionado é utilizado nas mesmas aplicações que o aprendizado supervisionado, porém manipula tanto dados de entrada rotulados quanto dados não rotulados. A coleta de dados rotulados geralmente é difícil, custosa ou consome muito tempo. Em contrapartida a coleta de dados não rotulados apesar de ser mais fácil, apresenta limitações em seu uso. O algoritmo tenta resolver este problema utilizando um grande volume de dados não rotulados juntamente com um menor conjunto de dados rotulados (ZHU, 2005). É útil em situações onde o custo associado à rotulação é muito alto para possibilitar o processo de treinamento.

No aprendizado supervisionado, existem dois conjuntos de dados essenciais, e serão referidos como conjunto X e conjunto Y. O conjunto X é formado pelas amostras de dados coletadas para análise, enquanto o conjunto Y é formado pelos rótulos relativos aos dados

do conjunto X .

Assim o algoritmo supervisionado, recebe ambos conjuntos, X e Y , e tenta encontrar uma relação entre eles para construir uma função F que, ao ser aplicada sobre um conjunto de dados X , retorne uma estimativa dos respectivos valores de rótulo em Y .

Ainda no aprendizado supervisionado, existe uma divisão quanto a forma como se manipula os dados se o rótulo (conjunto Y) é um número real, então se trata de uma regressão se o rótulo pertence a um conjunto finito e não ordenado se trata de uma classificação.

No treinamento por classificação, as entradas (conjunto X) são rotuladas por classes discretas ou categorias, dessa forma o algoritmo busca entender os agrupamentos de dados formados por categoria no conjunto de treino, então constrói um modelo baseado no mapeamento do conjunto de treino, para suas respectivas categorias, que vincula novas entradas do conjunto de teste à uma ou mais dessas classes.

No treinamento por regressão, as entradas (conjunto X) são rotuladas por números reais, assim o algoritmo busca construir um modelo que, dada uma nova entrada com o rótulo desconhecido, seja possível estimar qual será o rótulo dessa entrada. Para isso o algoritmo tenta mapear variáveis de entrada para alguma função contínua.

No caso deste trabalho, cada entrada do conjunto X_{treino} é formada por diversas métricas (atributos) referentes aos recursos da infraestrutura e o rótulo de cada entrada é o tempo médio das operações de escrita ou leitura (conjunto Y_{treino}). Dessa forma, o algoritmo de regressão tem o objetivo de encontrar um conjunto de pesos e viés ótimo para essas métricas, de acordo com uma função de custo, por exemplo, o MAE (*Mean Absolute Error*). As estimativas retornadas pela função contínua expressam a relação entre valores de entrada e os valores previstos de saída, sendo possível compará-los ao valor verdadeiro do conjunto Y_{treino} , permitindo ajustar as previsões.

2.4 OpenStack

O OpenStack é uma combinação de projetos *open source*, que cria uma camada de orquestração de rede, utilizando e controlando pools de recursos virtuais como armazenamento, processamento e rede. Pode ser utilizado para criar e gerenciar os componentes de múltiplas infraestruturas virtualizadas, como *clouds* privadas e públicas (RED HAT, INC., 2019).

É considerado um sistema operacional, porém em uma escala de *cloud computing*. Ele fornece interfaces de programação de aplicações (API) que, em conjunto, são capazes de controlar todos os recursos disponíveis em uma infraestrutura de rede: máquinas virtuais, rede, armazenamento e balanceamento de carga (OPENSTACK, 2019). Utilizando

esse conjunto disponível de APIs, o OpenStack abstrai os recursos virtuais e os transforma em *pools* discretos que potencializam as ferramentas de *cloud computing* (RED HAT, INC., 2019).

Dentre os diversos projetos disponíveis, há 6 serviços básicos que constituem a base do OpenStack, abrangendo computação, rede, armazenamento, identidade e imagens. Esses serviços são (RED HAT, INC., 2019):

- *Nova* – É uma ferramenta de acesso e gerenciamento total para os recursos computacionais do OpenStack;
- *Neutron* - é responsável por fornecer os recursos de rede para outros serviços do OpenStack;
- *Swift* – É um serviço de armazenamento de objetos, altamente tolerante a falhas, que armazena e recupera objetos de dados não estruturados usando uma API RESTful;
- *Cinder* – Oferece armazenamento de blocos persistentes, acessível por meio de uma API de autosserviço;
- *Keystone* – Autentica e autoriza todos os serviços do OpenStack. Ele também é o catálogo de *endpoints* para todos os serviços;
- *Glance* – Armazena e recupera imagens de discos de máquinas virtuais de uma variedade de locais.

Outros componentes adicionais proveem orquestração, gerenciamento de falhas e gerenciamento de serviços, entre os serviços para garantir a alta disponibilidade das aplicações.

2.5 Trabalhos Relacionados

Esta seção apresenta alguns trabalhos relacionados. Um deles faz o estudo e experimentação de métodos para a predição de métricas de qualidade de serviço (QoS) a partir de estatísticas de infraestrutura (STADLER; PASQUINI; FODOR, 2017). Outro trabalho buscou construir um serviço para experimentação e coleta de métricas de desempenho (MATOS, 2018), se assemelhando ao cenário apresentado neste artigo.

Em Stadler, Pasquini e Fodor (2017) o cenário e metodologia apresentado tem grande relação com o trabalho desta monografia. O objetivo foi prever as métricas de QoS de um serviço de transmissão de vídeo sob demanda e métricas de tempo de resposta de operações de leitura e escrita de dados em um serviço de armazenamento chave-valor.

Os autores criaram um ambiente de rede com os dois serviços disponíveis, geradores de carga e um cliente. Configuraram a coleta das métricas e realizaram experimentos com aplicação de carga. Com as métricas coletadas compararam dois algoritmos de aprendizado de máquina (*Random Forest* e *Regression Tree*) na geração do modelo de predição de métricas e dois métodos de seleção de métricas (*forward-stepwise-selection* e *univariate-feature-selection*) na redução do conjunto de métricas utilizadas para o treinamento.

Os autores concluem que o método de *Random Forest* tem maior precisão que o *Regression Tree* na estimativa de métricas de QoS no cenário construído, já nos métodos de seleção de métricas, observam que ambos os métodos geram um conjunto de métricas que produzem um resultado com precisão parecida, sendo o *univariate-feature-selection*, ligeiramente melhor. Demonstram também que rodar apenas uma aplicação em um servidor permite uma estimativa mais precisa das métricas de serviço daquela aplicação.

Finalmente constatam que, apesar da utilização de um conjunto reduzido de métricas na geração do modelo preditivo, a taxa de erro observada não sofreu um grande aumento, e em diversos casos se manteve próximo do obtido com o conjunto completo, ao mesmo tempo em que reduziu consideravelmente o tempo e processamento necessários na regressão dos dados.

Nosso trabalho segue a mesma metodologia, porém em um ambiente mais sucinto, com um serviço distribuído em 5 nós, rodando em uma infraestrutura dedicada a ele. Este serviço sendo um sistema de armazenamento NoSQL (Apache Cassandra), onde buscaremos demonstrar os efeitos da aplicação de carga observados pelo cliente, coletar os conjuntos de dados e demonstrar que é possível realizar a predição de métricas de QoS a partir de dados coletados no cluster do serviço, além de verificar a eficácia da seleção de métricas nesse serviço e qual é o menor conjunto que se pode atingir para realizar as predições.

Em [Matos \(2018\)](#) o objetivo foi a construção de um ambiente de distribuição de vídeo sob demanda, implantado sob uma estrutura containerizada em uma plataforma de computação em nuvem. Além de implementar um mecanismo que simulasse uma carga sobre o serviço e ao mesmo tempo realizasse a coleta de métricas relativas ao QoS recebido por um cliente.

O autor conclui que foi possível coletar os conjuntos de dados necessários para uma posterior aplicação de um algoritmo de aprendizado de máquina, e que foi possível observar as alterações nas métricas de QoS do cliente, conforme a variação da geração de carga.

A metodologia do trabalho é semelhante ao deste trabalho, com a configuração de um serviço distribuído, um cliente e um gerador de carga para a realização de experimentos, visando coletar o QoS recebido pelo cliente ao utilizar o serviço conforme o gerador

de carga carrega o serviço. Neste trabalho iremos efetivamente utilizar os conjuntos de dados coletados para a realização de predições de métricas.

3 Desenvolvimento

Este capítulo aborda todo o desenvolvimento do trabalho, começando pelo projeto e construção do ambiente de testes, em seguida a realização de experimentos, a coleta dos dados e por fim a utilização de aprendizado de máquina no tratamento dos dados. O capítulo também expõe os resultados obtidos durante todo o período de desenvolvimento.

Para alcançar o objetivo principal proposto por esse trabalho, o primeiro passo dado foi projetar um ambiente de testes que simulasse o funcionamento de um ambiente do serviço Cassandra e permitisse a realização de experimentos e coleta de dados. Entende-se por ambiente Cassandra, o conjunto de todas as partes envolvidas no funcionamento típico do serviço Cassandra, são elas, a infraestrutura que suporta o serviço e os diversos clientes que se conectam e estressam o serviço.

Em uma situação típica, o serviço Cassandra é provido por meio de um cluster formado por diversos nós localizados em diferentes servidores, e em uma interação básica os clientes realizam requisições de escrita e leitura em algum dos nós do cluster, que irá resolver a requisição ou delegar para o nó responsável. O cliente então recebe a resposta requisitada ou uma confirmação de que a requisição foi atendida.

Os experimentos realizados se basearam na observação da variação do tempo médio de resposta de operações de escrita e leitura realizadas por um cliente enquanto a carga sofrida pelo serviço (clientes requisitando operações) aumenta ou diminui. Com base nesse fundamento, a arquitetura do ambiente de testes foi projetada com 3 atores principais:

- Um cluster do serviço Cassandra;
- Um Cliente Cassandra;
- Um Gerador de Carga;

O Gerador de carga fez-se necessário pois havia uma limitação de máquinas disponíveis para a instanciación de diversos clientes, assim a estratégia definida foi utilizar apenas uma máquina para instanciar diversos clientes e aplicar padrões de carga variáveis sobre o cluster. A Figura 2 ilustra a arquitetura do ambiente de testes.

O cluster do serviço Cassandra é formado por 5 nós distribuídos, responsáveis por atender as requisições dos clientes e retornar uma resposta. Nele ocorre a coleta do conjunto de dados X, ou arquivo X, durante os experimentos, contendo métricas relativas à utilização dos recursos de infraestrutura.

O Cliente é responsável por gerar uma bateria de requisições de leitura e escrita no serviço Cassandra e simultaneamente coletar o conjunto de dados Y, ou arquivo Y,

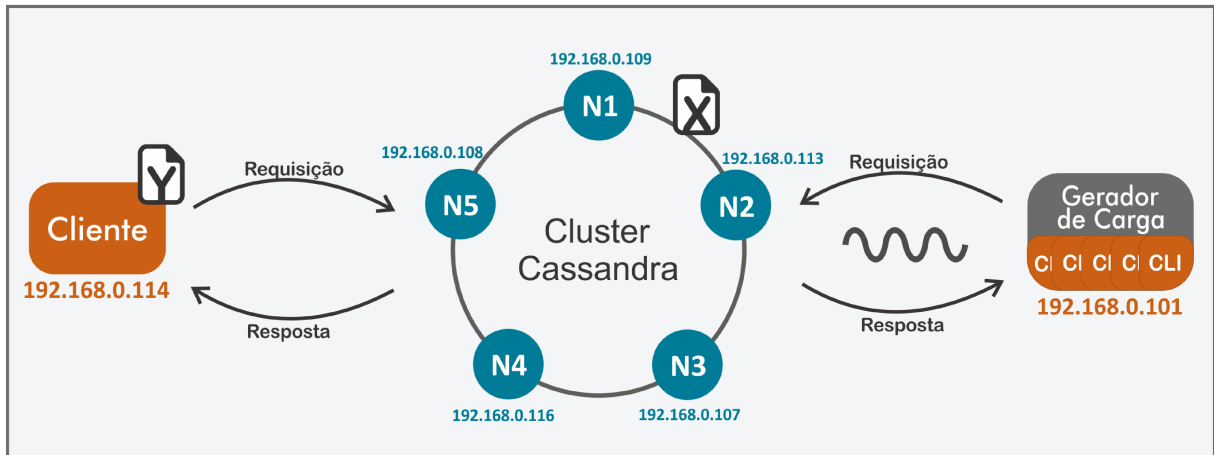


Figura 2 – Arquitetura do Ambiente de Teste Cassandra Projetado.

contendo as métricas referentes à qualidade do serviço fornecido pelo Cassandra ao Cliente.

Por fim, o Gerador de Carga é o responsável por simular uma carga que diversos clientes são capazes de induzir ao Cluster quando realizam requisições simultaneamente.

O objetivo dos experimentos realizados no ambiente de teste foi utilizar o Gerador de Carga para modular uma carga induzida no cluster de maneira que esta carga acompanhe um padrão variável. Assim atestar se a qualidade do serviço (conjunto de dados Y) observada pelo Cliente sofreria uma variação correspondente ao padrão de carga aplicado pelo Gerador de Carga.

Dessa forma, temos um cenário ideal para a coleta de amostras do conjunto X, possibilitando enfim a manipulação desses conjuntos de dados por um método de aprendizado de máquina para a predição de métricas de QoS do Cassandra, observadas por um cliente.

3.1 Infraestrutura

O ambiente de testes projetado para o desenvolvimento dos experimentos foi implementado em uma infraestrutura de servidores físicos disponibilizada pela Universidade Federal de Uberlândia. Essa infraestrutura de servidores é orquestrada pelo OpenStack, que possibilita a virtualização de máquinas e a definição dos recursos que serão providos a cada máquina.

Utilizando o OpenStack, foram virtualizadas 7 máquinas distintas: 5 delas projetadas para compor o cluster Cassandra, com sistema operacional Ubuntu 16.04, 4GB de RAM, 1vCPU e 50GB de disco; uma delas construída para trabalhar como o Cliente do nosso ambiente de teste, com sistema operacional Ubuntu 16.04, 4Gb de RAM, 1vCPU e 100Gb de disco; e a última construída para trabalhar como o Gerador de Carga, com

sistema operacional Ubuntu 16.04, 8GB de RAM, 4vCPU e 20Gb de disco.

Os algoritmos de aprendizado de máquina foram executados em um notebook com processador Intel i5 de sétima geração e 8GB de memória RAM, num ambiente com sistema operacional Ubuntu 18.04 LTS.

3.2 Cluster Cassandra

As cinco máquinas virtualizadas para compor o cluster do serviço foram configuradas da mesma forma. O primeiro passo foi baixar o pacote do Apache Cassandra v3.11.3 no site oficial ([Apache, 2016](#)). Em seguida foi configurado o arquivo `cassandra.yaml` que permite a modificação de parâmetros que ditam a maneira como aquele nó irá se comportar.

Antes de executar o serviço, as dependências devem ser instaladas, são elas, o Java 7 e o Python 2.7. O Cassandra então é executado em todas as máquinas e após atestar que os 5 nós estão ativos e conectados como cluster, o *keyspace* pode ser configurado. A Figura 3 ilustra os nós do cluster.

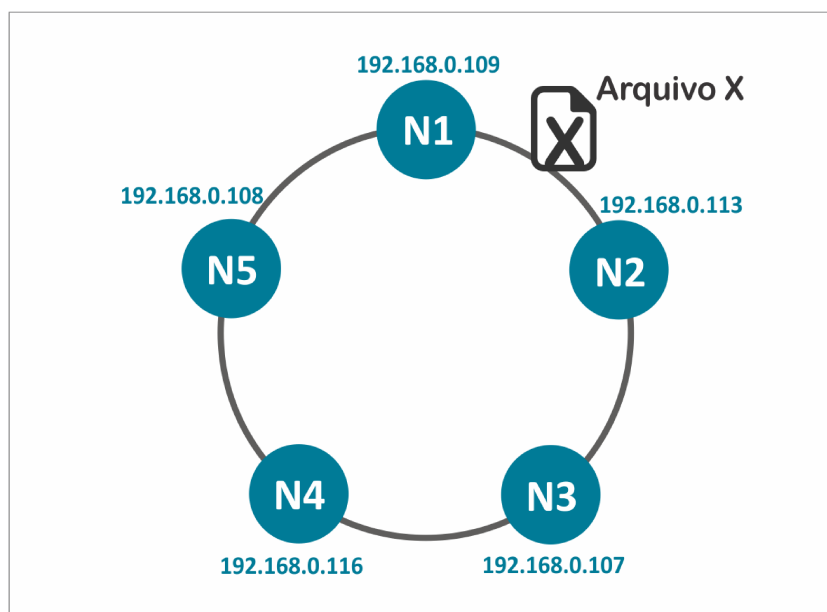


Figura 3 – Organização dos Nós do Cluster Cassandra.

O Cluster do serviço Cassandra é formado por 5 máquinas virtualizadas rodando uma instância do Apache Cassandra e configuradas para se conectarem e trabalharem como um cluster.

No arquivo `cassandra.yaml` de todas as máquinas do cluster são definidos os nós *seed*, ou seja, armazenam dados dos nós que se conectam a eles e enviam dados de todos os nós conectados para os recém-conectados, possibilitando que o cluster seja formado automaticamente, conforme os nós são reconhecidos.

Cada instância do Cassandra teve seu arquivo de configuração modificado da seguinte maneira:

- `cluster_name`: 'Cassandra Cluster'
- `seeds`: "192.168.0.116,192.168.0.108"
- `listen_address`: {IP}
- `rpc_address`: {IP}
- `endpoint_snitch`: `GossipingPropertyFileSnitch`

No exemplo acima, IP indica o endereço de IP do nó do cluster a qual o arquivo de configuração pertence. O listen address indica qual o endereço IP os demais nós do cluster irão usar para se conectar a este nó. O rpc address funciona como o listen address para chamadas de procedimento remoto (RPC).

O endpoint snitch, indica de que maneira a topologia do cluster será comunicada entre os nós, o `GossipingPropertyFileSnitch` utiliza o protocolo *gossip* para comunicar o datacenter e rack em que cada nó está localizado. Por se tratar de uma configuração de cluster simples todos os nós se encontram no mesmo datacenter e rack.

Por fim faz-se necessária a configuração do keyspace, onde os dados serão armazenados e que tipo de dados serão armazenados durante os testes de escrita e leitura. Para a realização dos experimentos foi definido a utilização do *keyspace* padrão do Cassandra, o *keyspace1*, que possui duas tabelas, a tabela *standart1* e *counter1* que não será utilizada. A tabela *standart1* possui 6 colunas, uma para armazenar uma chave de tamanho 10 bytes e 5 para armazenar dados com tamanho de 34 bytes cada, totalizando 180 bytes por inserção com as 6 colunas.

O *keyspace1* foi configurado para ser utilizado em um cluster de múltiplos nós, e seu fator de replicação foi modificado de 1 para 3, assim cada dado inserido em algum nó do cluster será replicado para mais dois nós, mantendo sempre 3 cópias de cada inserção. Após estes ajustes, o cluster está pronto para receber requisições de clientes.

3.3 Cliente

A máquina instanciada para trabalhar como o Cliente do ambiente de testes é responsável por realizar uma bateria de operações de escrita e leitura no cluster e simultaneamente coletar dados referentes à essas operações. Para realizar essa bateria de operações é utilizada a ferramenta *cassandra-stress*, que realiza diversas operações por segundo no Cluster, de acordo com os parâmetros definidos no comando, e pode armazenar diversas métricas de desempenho dessas operações em um *log*.

A cada segundo são coletadas métricas referentes às operações realizadas, tais

```
timestamp,W_totalops,W_op/s,W_pk/s,W_row/s,W_mean,W_med,W_95,W_99,W_999,
1559328766,1297,1297.0,1297.0,1297.0,3.14,2.07,9.22,18.30,44.47,45.91,0.
1559328767,2824,1527.0,1527.0,1527.0,2.60,1.09,5.53,64.19,92.93,93.78,0.
1559328768,5085,2261.0,2261.0,2261.0,1.91,0.99,4.70,11.62,77.53,79.50,0.
1559328769,7699,2614.0,2614.0,2614.0,1.69,0.91,3.62,8.68,99.09,103.02,0.
1559328770,10179,2480.0,2480.0,2480.0,1.86,0.83,3.57,17.12,72.35,72.55,0
1559328771,12482,2303.0,2303.0,2303.0,1.92,0.82,3.44,59.70,83.82,85.59,0
1559328772,15150,2668.0,2668.0,2668.0,1.49,0.99,3.15,8.52,62.78,67.44,0.
1559328773,17666,2516.0,2516.0,2516.0,1.83,0.87,3.65,15.57,83.69,85.00,0
1559328774,20065,2399.0,2399.0,2399.0,1.77,0.89,4.24,15.41,86.31,96.21,0
1559328775,22634,2569.0,2569.0,2569.0,1.68,0.74,3.70,12.36,88.87,105.05,
```

Figura 4 – Trecho do Conjunto de Métricas de QoS Coletados pela Máquina Cliente.

como, quantidade de operações de escrita e leitura realizadas, o tempo médio das operações e o 95 percentil, como mostrado pelo trecho de um arquivo de log do *cassandra-stress* na Figura 4. As métricas do conjunto de dados do cliente que serão utilizadas no aprendizado de máquina são, Tempo Médio de Escrita (*W_mean*) e Tempo Médio de Leitura (*R_mean*) individualmente. Essas métricas devem refletir a performance do Cluster diante das variações de carga aplicadas pelo Gerador de Carga.

3.4 Gerador de Carga

A máquina designada para trabalhar como o Gerador de Carga é responsável por aplicar uma carga variável no cluster com o intuito de causar alterações na performance observada pela máquina Cliente. A partir de um *script* escrito em *Python* é possível instanciar e finalizar diferentes processos do *cassandra-stress* durante o tempo de experimento, ou seja, em certo momento muitas instâncias do *cassandra-stress* estarão sendo executadas e em outro momento poucas instancias do *cassandra-stress* estarão sendo executadas.

Cada instanciação do *cassandra-stress* recebe um parâmetro *threads*, que indica a quantidade de *threads* que aquele cliente utilizará para realizar as operações. Desse modo, duas variáveis influenciam diretamente na geração de carga no cluster, a quantidade de clientes instanciados pelo *cassandra-stress* e a quantidade de *threads* que cada cliente utiliza nas operações.

Uma instanciação do *cassandra-stress* se comporta como um ou vários clientes realizando operações concorrentemente. Como o nível de carga gerada por essa ferramenta é fixo conforme a quantidade de threads definida, um *script* de geração de carga variável construído em *python*, foi adaptado para permitir a criação e deleção de instâncias do *cassandra-stress* de acordo com algum padrão de carga variável definido.

O *script* segue um padrão periódico sinusoidal na instanciação de clientes, e pode-se definir qual a quantidade inicial de clientes ativos, qual a amplitude de clientes que

irão configurar a onda, além do tempo total de experimento e o período da onda que será gerada.

Durante o tempo de experimento a quantidade inicial A de clientes será corrigida por uma amplitude B calculando o valor de Λ ($A + / - B$) do nosso processo Poisson não-homogêneo de geração de carga.

3.5 Definindo Limites para o Padrão de Carga

A primeira sessão de experimentos teve o objetivo de encontrar um padrão de carga balanceado para ser aplicado ao cluster, ou seja, uma carga que descreva uma sinusóide que em seu pico positivo não exceda a capacidade de trabalho do cluster e que em seu vale o tempo médio das operações seja significativamente menor que quando no pico positivo. O padrão de carga é definido por dois parâmetros, o máximo e o mínimo de carga que será gerado durante a utilização do mesmo.

Este balanceamento deve ser feito, pois se for utilizado um padrão de carga com ambos parâmetros muito elevados ou muito baixos, o resultado observado pelo cliente será uma média de tempo de resposta constante ou com pouca variação durante todo o tempo de experimento. Isso ocorre pois, no caso de um padrão muito baixo, o serviço Cassandra estará operando com folga, oferecendo o menor tempo de resposta possível tanto para o mínimo quanto para o máximo de carga e no caso de um padrão muito alto, o serviço estará operando em seu limite, onde o tempo de resposta é muito alto, inconsistente e imprevisível para ambos os parâmetros.

Um nível adequado de padrão de carga sinusoidal seria aquele que quando atingisse a quantidade mínima de clientes instanciados, o serviço conseguisse atender com folga as requisições, e quando atingisse a quantidade máxima o serviço tivesse que se esforçar para atender a demanda sem exceder seu limite, além disso seria importante que no intervalo entre os picos máximo e mínimo de carga, a oscilação do tempo médio das operações estivesse explícita, acompanhando o padrão sinusoidal da carga aplicada.

Para definir esse padrão, foi preciso encontrar o máximo de carga que o cluster pode sustentar, e o mínimo de carga necessária para gerar alterações na performance observada. Um experimento foi idealizado para encontrar esses pontos essenciais da performance do serviço.

No experimento o Cliente rodou uma instancia do *cassandra-stress* durante 2 horas, trabalhando com 14 *threads* e sempre realizando operações de escrita e leitura, armazenando o tempo médio dessas operações em um log. O Gerador de carga foi configurado para durante essas 2 horas gerar um padrão de carga em escada, ou seja, o experimento iniciou-se com a aplicação de uma carga leve gerada por 16 *threads*, e a cada 14 minutos

a quantidade de *threads* foi multiplicada por 1,5.

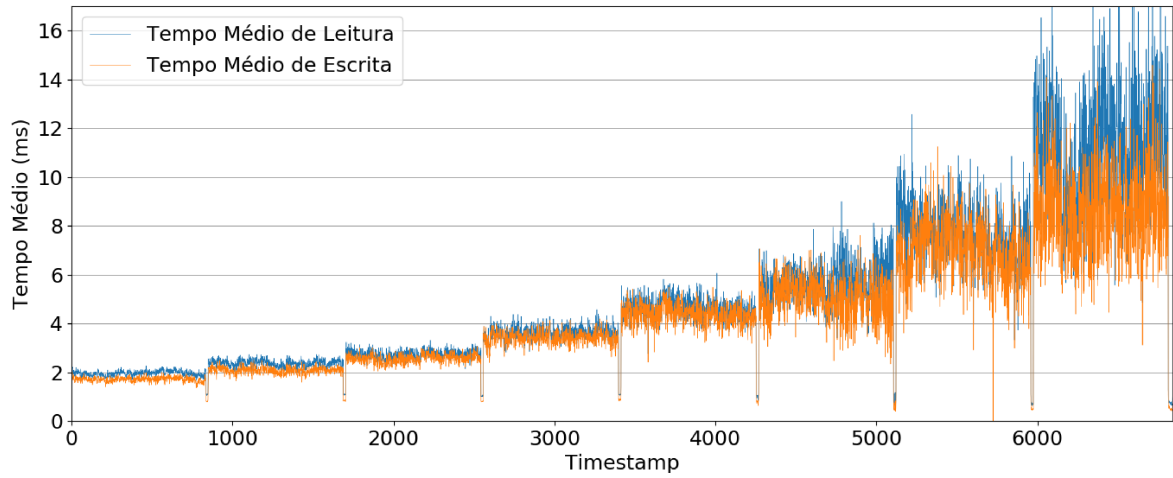


Figura 5 – Tempo Médio das Operações de Escrita e Leitura Amostrado Durante Experimento com Padrão de Escada.

A Figura 5 mostra o tempo médio das operações de escrita e leitura realizadas a cada segundo de experimento, evidenciando que a partir do *Timestamp* de número 4200 onde começam a ser utilizadas 121 *threads* para estressar o cluster, a performance do serviço fica instável, fornecendo um tempo de resposta médio inconsistente e imprevisível, com eventuais picos de tempo. A tabela 1 mostra o tempo médio observado nos 14 minutos de experimento referentes a cada nível de *threads* nas operações de escrita.

<i>Threads</i>	Tempo médio de escrita (ms)	Desvio Padrão (ms)
16	1,71	0,11
24	2,07	0,14
36	2,55	0,18
54	3,40	0,27
81	4,37	0,43
121	5,04	0,79
181	6,9	1,21
271	8,57	1,70

Tabela 1: Tempo Médio de Escrita em Diferentes Níveis de Carga.

O limite estabelecido para a carga máxima e mínima com que trabalharíamos se deu em função da quantidade de *threads*, ou seja, independentemente da quantidade de clientes instanciados, a soma total das *threads* com que cada cliente trabalha não deveria exceder 120 e o mínimo necessário de *threads* foi definido como a menor quantidade possível de ser configurada no Gerador de Carga.

3.6 Averiguação do Funcionamento do Gerador de Carga

Com os limites definidos, o próximo experimento foi projetado para atestar se, ao aplicar um padrão de carga sinusoidal, os tempos de resposta médio das operações oscilariam de acordo com a onda. Para esse experimento e todos os que o seguiram, foi definido que os clientes instanciados deveriam possuir a mesma configuração e trabalhariam com a mesma quantidade de *threads*, inclusive o cliente instanciado na máquina cliente.

Durante os primeiros experimentos um problema foi identificado, a quantidade de memória RAM disponível pela máquina do Gerador de Carga conseguia sustentar a instanciação de no máximo 18 clientes. O que foi contornado rapidamente adotando uma maior quantidade de *threads* utilizadas por cliente, permitindo atingir o pico (máximo) de carga definido anteriormente, porém ocasionando um aumento no vale (mínimo) de carga. Dessa forma os experimentos seguintes foram projetados para não ultrapassar a quantidade de 18 instanciações do *cassandra-stress*.

Após diversos experimentos com proporções de clientes e *threads* por cliente diferentes, o experimento seguinte obteve um resultado balanceado e demonstrou claramente o comportamento sinusoidal esperado como reflexo da carga aplicada no cluster.

Nesse experimento, o *script* gerador de carga foi configurado para rodar durante duas horas, iniciando com 6 clientes ativos trabalhando com 10 *threads* cada um, além de gerar uma sinusóide com período de uma hora e amplitude de 5 clientes, ou seja, o número de clientes durante o período de uma hora oscila entre 11 e 2 clientes. Além dos clientes instanciados pelo Gerador de Carga há também o cliente instanciado pela máquina Cliente que possui parâmetros idênticos aos demais. A quantidade mínima de clientes instanciados não é 1, como o esperado com uma amplitude 5, pois no Gerador de Carga o Lambda calculado para definir a quantidade de clientes em cada instante tem seu resultado arredondado para cima.

A Figura 6 apresenta os tempos médios das operações de escrita no serviço Cassandra observado na máquina Cliente durante o experimento de duas horas. É possível notar as oscilações causadas no tempo médio de resposta, visto que no pico de carga o tempo médio fica perto de 15 ms e no vale fica perto de 5 ms, sendo um reflexo do número de clientes que foram criados e removidos pelo gerador de carga.

Com o padrão de carga balanceado definido, a próxima etapa do trabalho é enfim avaliar a eficácia da utilização de métodos de aprendizado de máquina, para isso faz-se necessário a implementação de uma ferramenta para coletar o conjunto de estatísticas de infraestrutura, conjunto X, nas máquinas que formam o cluster do Cassandra.

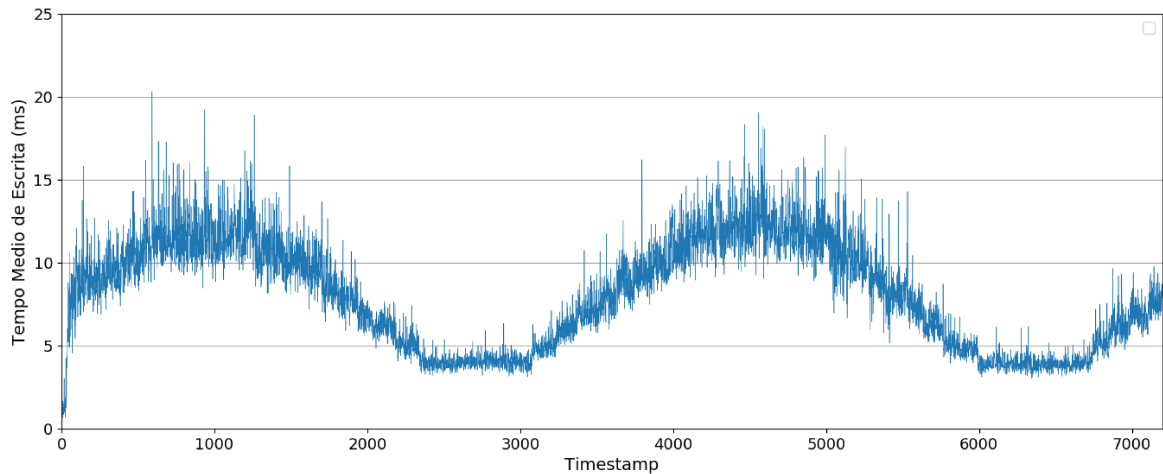


Figura 6 – Tempo Médio das Operações de Escrita Amostrado Durante Experimento de Duas Horas com Padrão Sinusoidal.

3.7 Coleta de Métricas de Infraestrutura

O conjunto X de dados deve conter métricas referentes ao consumo de recursos e de infraestrutura das máquinas que hospedam os nós do serviço Cassandra, logo essas métricas devem ser coletadas em todas as máquinas que formam o cluster do serviço.

Esse conjunto de dados é essencial para os próximos experimentos, uma vez que o objetivo será a partir dele e do conjunto de dados Y, construir um modelo de aprendizado de máquina que consiga estimar valores desconhecidos de um conjunto Y, como o tempo médio das operações de escrita, a partir do estado das métricas do conjunto X no mesmo instante.

Para realizar a coleta dos dados, utilizou-se a ferramenta *Prometheus* em conjunto com um *exporter* de dados (KOCHIE, 2018), que foi instalado em todas as máquinas do cluster. Os *exporters* enviam os dados coletados para um servidor onde o serviço *Prometheus* está sendo executado, e a partir do momento que o *Prometheus* armazena essas métricas o acesso a elas é disponibilizado. Vale destacar que o servidor que hospeda o serviço *Prometheus* está fora do ambiente de teste, não influenciando a performance das máquinas.

O arquivo gerado pelo *Prometheus* contendo todas as métricas necessárias é um CSV, onde os valores de cada coluna são referentes a uma métrica diferente e cada linha agrupa os valores das métricas coletadas no mesmo instante de tempo (*timestamp*). A Figura 7 mostra um trecho de um arquivo de log do *Prometheus*.

O intervalo definido para a coleta dos dados foi de 1 segundo, para ser compatível com os dados do conjunto Y coletados na máquina cliente, e possibilitar relacionar os

Arquivo Editar Formatar Exibir Ajuda

```
timestamp,0016237,00c6bb7,02107616,0231f7d,02e6fcf,03035ae,03301f9,036b4923
1559328776,38,2279321600,11009586,2371584,15313524423,1575,44,66682880
1559328777,38,2279321600,11009586,2371584,15313524423,1575,44,66682880
1559328778,38,2279321600,11009586,2371584,15313597401,1575,44,66715648
1559328779,38,2279239680,11009586,2371584,15313597401,1575,44,66715648
1559328780,38,2279239680,11009586,2371584,15313597401,1575,44,66715648
1559328781,38,2279239680,11009586,2371584,15313597401,1575,44,66715648
1559328782,38,2279239680,11009586,2371584,15313597401,1575,44,66715648|
1559328783,38,2279239680,11009586,2371584,15313670413,1575,44,66682880
1559328784,38,2279387136,11009586,2371584,15313670413,1575,44,66682880
1559328785,38,2279387136,11009586,2371584,15313670413,1575,44,66682880
1559328786,38,2279387136,11009586,2371584,15313670413,1575,44,66682880
```

Figura 7 – Parte do Conjunto X de Dados Coletado Durante um Experimento.

valores do conjunto de dados X com os valores de dados Y pelo respectivo *timestamp*. Vale destacar que todas as máquinas envolvidas no experimento, possuem seus relógios sincronizados pelo protocolo de tempo da rede (NTP).

O *exporter* coletou no total 691 métricas durante o experimento, cada máquina forneceu uma média de 138 métricas, além do *timestamp*, que serve para relacionar os dados de todos os nós e não é utilizada como um atributo no aprendizado de máquina.

3.8 Experimento para Coleta dos Conjuntos de Métricas

Após a definição de um padrão de carga balanceado para o Gerador de Carga, o objetivo foi aplicar esse padrão sinusoidal sobre o cluster durante um experimento mais longo, enquanto a máquina Cliente também realiza operações no cluster do serviço Cassandra e coleta o conjunto de dados Y. Porém, neste experimento o foco foi coletar o conjunto de dados X simultaneamente, para que no final do experimento tenhamos em mãos os conjuntos de dados X e Y, referentes ao mesmo intervalo de tempo.

O *script* gerador de carga foi configurado para realizar um experimento de quatro horas, iniciando com 6 clientes ativos trabalhando com 10 *threads* cada um, além de gerar uma sinusóide com período de uma hora e amplitude de 5 clientes, ou seja, o número de clientes durante o período de uma hora oscila entre 11 e 2 clientes, além do cliente instanciado pela máquina Cliente que possui parâmetros idênticos aos demais clientes instanciados.

A Figura 8 mostra o tempo médio das operações de escrita, a cada segundo de experimento, durante as 4 horas. Assim como no primeiro experimento, os efeitos das oscilações de carga aplicadas no cluster são claramente visíveis, o que fortalece a hipótese de que a utilização de aprendizado de máquina para a predição dos tempos médios de

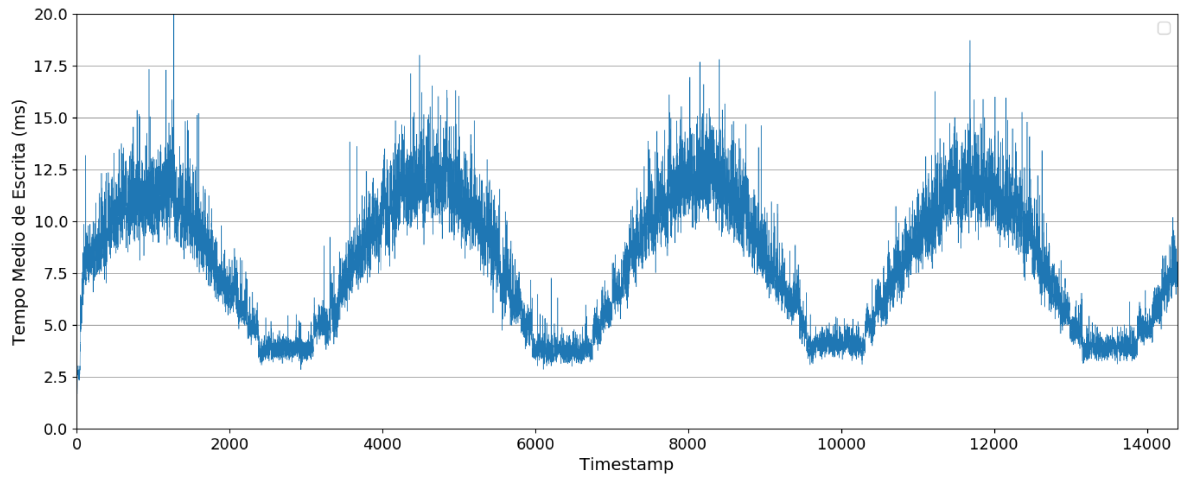


Figura 8 – Tempo Médio das Operações de Escrita Amostrado Durante Experimento de Quatro Horas com Padrão Sinusoidal.

resposta pode gerar resultados satisfatórios.

3.9 Predição dos Tempos de Resposta do Cassandra

O método de aprendizado de máquina relaciona os valores dos conjuntos de dados X e Y com a estrutura (x_t, y_t) , ou seja, o valor da métrica de QoS do conjunto Y no *timestamp* t está correlacionada com o valor da estatística de infraestrutura do conjunto X no mesmo *timestamp* t.

O objetivo dessa etapa é treinar e avaliar um modelo de aprendizado de máquina que se encaixa em um conjunto de dados particular. As etapas desse processo foram: dividir aleatoriamente as amostras (x_t, y_t) dos conjuntos X e Y em conjuntos de treinamento (X_{treino}, Y_{treino}) e conjuntos de teste (X_{teste}, Y_{teste}) . 70% das amostras foram designados para o conjunto de treinamento e 30% para o conjunto de teste. Em seguida, construir o modelo de aprendizado de máquina utilizando o conjunto de dados de treinamento e, por fim, avaliar o modelo construído comparando os valores de Y estimados pelo modelo e os valores reais do conjunto Y_{teste} .

Para avaliar os modelos treinados, duas métricas são calculadas, uma delas é o *Normalized Mean Absolute Error* (NMAE), que é expressada por:

$$\frac{1}{\bar{y}} \left(\frac{1}{m} \sum_{i=1}^m |y_i - \hat{y}| \right),$$

onde $\hat{y}$ é a estimativa da amostra do conjunto Y feita pelo modelo, y_i é o valor da amostra do conjunto Y_{teste} e $\bar{y}$ é a média de todas as amostras de y_i , m é a quantidade de amostras existentes do conjunto Y_{teste} . A outra métrica de avaliação é o tempo de treinamento, que

expressa o tempo necessário para treinar um modelo utilizando as métricas do conjunto X_{treino} .

Os mecanismos de aprendizado de máquina foram executados em um notebook com processador Intel i5 de sétima geração e 8GB de memória RAM, num ambiente com sistema operacional Ubuntu 18.04 LTS, utilizando *python* 2.7 e *scikit-learn* versão 0.18.1.

O algoritmo de aprendizado de máquina utilizado para a regressão foi o *Random Forest*, utilizando a implementação oferecida no pacote de *machine learning* para *python* *scikit-learn* (LEARN, 2018). O algoritmo cria diversas árvores de decisão com conjuntos de métricas aleatórios, e combina o resultado de todas as árvores para gerar um único resultado.

Os parâmetros de configuração do algoritmo de aprendizado de máquina foram: `max depth = 10`, `random state = 17` e `n_estimators = 120`. O `max depth` indica qual a profundidade máxima das árvores criadas, o `random state` é utilizado para possibilitar a replicação do experimento e o `n_estimators` indica quantas árvores diferentes serão geradas.

3.10 Resultados

A Figura 9 apresenta duas séries de dados, uma referente aos valores do conjunto Y_{teste} , reservado para avaliar as predições, e outra referente ao conjunto $X_{estimado}$, com os valores estimados pelo método de regressão realizado. Foram utilizadas todas as 691 métricas do conjunto X_{treino} na construção do regressor. Após realizar as estimativas dos rótulos referentes ao conjunto X_{teste} e aplicar a função Normalized Mean Absolute Error (NMAE) para análise de erro sobre o resultado, a taxa de erro observada em relação às amostras reais foi de 8,11% para os tempos médios de escrita e 7,95% para os tempos médios de leitura.

A Tabela 2 apresenta os valores de NMAE e *Naive* NMAE para ambas operações de leitura e escrita, considerando a métrica de QoS utilizada sendo o tempo médio das operações. O *Naive* NMAE é o erro obtido quando utilizada uma abordagem ingênua do NMAE, onde não se compara cada amostra de Y_{treino} com a respectiva amostra de Y_{teste} , e sim a média aritmética do conjunto Y_{treino} com os valores do conjunto Y_{teste} .

O alto valor de *Naive* NMAE em comparação com o NMAE indica que o modelo gerado realmente realiza predições que se aproximam dos valores de Y_{teste} , pois o erro é muito menor quando obtido por comparações entre as amostras de um mesmo *timestamp* do que o obtido utilizando um valor médio das amostras, ou seja, as estimativas tendem a ser mais próximas do valor real da amostra do que do valor médio.

A Tabela 3 traz um comparativo entre as 6 primeiras métricas do conjunto $Y_{estimado}$

Métrica de QoS	NMAE (%)	NMAE <i>Naive</i> (%)
Tempo Médio de Escrita	8,11	35,04
Tempo Médio de Leitura	7,95	33,89

Tabela 2: NMAE e Tempo de Treinamento na Predição do Tempo Médio de Escrita e Leitura.

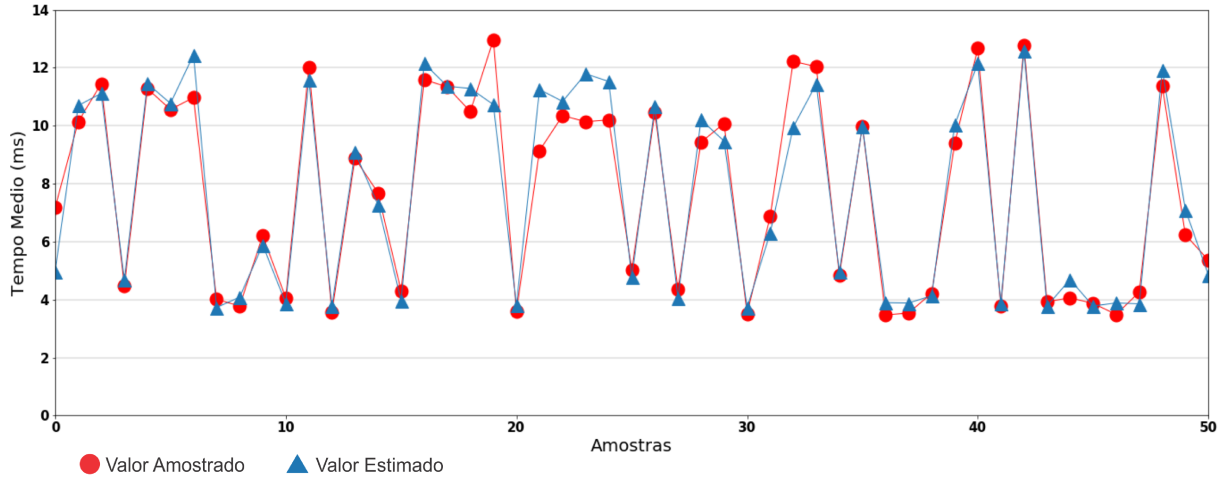


Figura 9 – Tempo Médio de Escrita Amostrado e Tempo Médio de Escrita Estimado.

e o respectivo valor amostrado no conjunto Y_{teste} , é possível observar que o valor predito varia com uma proporção próxima da variação dos valores reais e, em alguns casos, os valores possuem uma diferença mínima.

N	$Y_{estimado}$ (ms)	Y_{teste} (ms)
1	4.97	7.17
2	10.69	10.14
3	11.11	11.45
4	4.68	4.47
5	11.45	11.28
6	10.77	10.56

Tabela 3: Seis Primeiros Valores Estimados e Respectivos Valores Reais.

Estes resultados mostram que, apesar de haver uma taxa significativa de erro, as estimativas acompanham a variação dos valores amostrados, ou seja, a partir das métricas de infraestrutura coletadas, foi possível treinar um modelo de aprendizado de máquina que realiza predições dos tempos de resposta do serviço Cassandra, com uma certa taxa de erro.

Conforme mencionado anteriormente, o conjunto X_{treino} utilizado na construção da função de regressão possui todas as 691 métricas coletadas pelo *prometheus*. A quantidade de métricas podem variar dependendo da infraestrutura utilizada e da quantidade de nós ativos no cluster, podendo tornar a extração e regressão dos dados um trabalho custoso e demorado se houver uma quantidade massiva de métricas. Além de gerar esse alto custo

de processamento, o conjunto total de métricas contém algumas que estão pouco ou nada relacionadas à performance do serviço Cassandra, atribuindo um ruído ao resultado das predições ao serem utilizadas na regressão.

Uma maneira de amenizar os problemas observados pela utilização de um grande volume de métricas no experimento, é reduzir o conjunto de métricas necessárias para a construção do regressor, de maneira que a taxa de erro obtida não sofra grandes alterações.

Uma estratégia de seleção de métricas foi utilizada, considerando que o algoritmo de aprendizado de máquina Random Forest classifica todas as métricas do conjunto X pelo seu grau de correlação com o conjunto Y (BREIMAN, 2001), pequenos conjuntos com as melhores métricas foram utilizados para realizar as predições.

Ao selecionar apenas as K (quantidade de métricas) métricas melhor classificadas para realizar o treinamento do modelo, ou seja, as métricas com maior correlação com o conjunto Y, espera-se obter resultados semelhantes ao obtido pelo conjunto total de métricas nas predições e, ao mesmo tempo, menor custo e tempo de processamento.

K	Tempo Médio de Escrita		Tempo Médio de Leitura	
	NMAE(%)	Tempo de Treinamento (s)	NMAE(%)	Tempo de treinamento (s)
1	8,438	0,544	8,763	0,531
3	8,439	0,597	8,763	0,597
5	8,136	0,716	8,278	0,714
10	8,046	0,920	8,151	0,924
20	8,059	1,333	8,183	1,442
50	8,080	8,980	7,918	8,799
100	8,094	23,217	7,914	23,245
200	8,100	49,076	7,939	50,153
691	8,117	199,782	7,956	200,863

Tabela 4: NMAE e Tempo de Treinamento Obtidos para Diferentes Valores de K.

A Tabela 4 apresenta o NMAE e tempo de treinamento obtidos com a utilização de diferentes valores de K para realizar a regressão dos dados. Foram utilizados os valores $K = 1, 3, 5, 10, 20, 50, 100$ e 200 para realizar predições do tempo médio das operações de escrita e de leitura.

Analisando o resultado para cada valor de K , pode-se observar que a menor taxa de erro obtida na estimativa de tempo médio de operações de escrita, foi quando utilizadas apenas as 10 métricas melhor classificadas, com a taxa de erro de 8,046%, enquanto que na estimativa de tempo médio de operações de leitura a menor taxa de erro obtida foi de 7,914%, quando utilizadas as 100 melhores métricas.

A Tabela 5 mostra as 10 melhores métricas de acordo com o ranking gerado pelo algoritmo de aprendizado de máquina durante o treinamento utilizando a métrica de QoS Tempo médio de resposta de escrita. Dentre as melhores métricas estão:

Classificação	Estatística	Nó
1	node_sockstat_sockets_used	192.168.0.113
2	node_sockstat_TCP_inuse	192.168.0.113
3	node_netstat_Tcp_CurrEstab	192.168.0.113
4	node_sockstat_TCP_alloc	192.168.0.113
5	node_sockstat_TCP_inuse	192.168.0.116
6	node_sockstat_TCP_alloc	192.168.0.116
7	node_netstat_Tcp_CurrEstab	192.168.0.116
8	node_sockstat_sockets_used	192.168.0.116
9	node_sockstat_sockets_used	192.168.0.107
10	node_sockstat_TCP_inuse	192.168.0.107

Tabela 5: Dez Métricas com Melhor Classificação no Ranking do treinamento do Tempo Médio de Resposta de Escrita.

- `node_sockstat_sockets_used`: Indica a quantidade de *sockets* em uso no nó;
- `node_sockstat_TCP_inuse`: Indica a quantidade de *sockets* sendo utilizados para conexões TCP;
- `node_netstat_Tcp_CurrEstab`: Indica a quantidade de conexões TCP com o estado de Estabelecida;
- `node_sockstat_TCP_alloc`: Indica a quantidade de *sockets* TCP em estado alocado.

O resultado obtido ao aplicar a redução do conjunto de métricas utilizadas no treinamento confirma que é possível utilizar um conjunto reduzido de métricas na realização de estimativas e, ainda assim, manter a taxa de erro bem perto do que foi obtida utilizando o conjunto total de métricas. Note que em alguns casos uma taxa de erro até menor foi observada, pois o ruído é eliminado dos dados para treinamento ao reduzirmos o conjunto de métricas.

A significativa redução do conjunto de métricas, otimiza a coleta e armazenamento do conjunto de dados por se tratar de um conjunto que ocupa menos espaço de disco e demanda um nível menor de processamento em sua manipulação, além de extinguir métricas que não possuíam correlação com o conjunto Y, conforme mencionado anteriormente.

Por fim a redução no tempo gasto para o treinamento do regressor é facilmente observada na 4. Quando utilizado o conjunto inteiro de métricas, o tempo necessário para treinar o modelo foi de 200,836 segundos, para as estimativas de tempo escrita, e 199,78 segundos, para as estimativas de tempo de leitura. Quando utilizado o conjunto reduzido de 10 métricas, que provê um resultado satisfatório em ambos os casos, o tempo de treinamento é de 0,920 e 0,924 segundos respectivamente, um resultado bastante significativo para a efetiva utilização desta abordagem em tempo de execução.

4 Conclusão

O principal intuito do trabalho foi o de provar que é possível treinar um modelo de aprendizado de máquina que conseguisse realizar predições de métricas de qualidade de serviço de um sistema de armazenamento NoSql, a partir de estatísticas de sua infraestrutura. Para isso, primeiramente foi necessário provar que é possível observar as alterações dessas métricas de serviço recebidas por um cliente, conforme a carga recebida pelo cluster sofresse também oscilações.

A construção do ambiente de testes foi alcançada com sucesso, o que permitiu a realização dos experimentos projetados bem como a coleta dos conjuntos de dados necessários para a análise do comportamento do cluster Cassandra e do Cliente.

Demonstramos através dos experimentos que ao gerar um padrão de carga variável sobre o cluster, foi possível observar uma variação na qualidade de serviço recebida pelo cliente, seguindo o padrão imposto pelo gerador de carga, ou seja, conforme a carga aumenta a qualidade do serviço provido diminui e vice-versa. Os experimentos permitiram também definir um nível de carga balanceado para ser aplicado pelo gerador de carga.

Após a configuração de uma estrutura de coleta de dados, realizou-se experimentos com o intuito de coletar conjuntos de dados referentes às métricas de qualidade de serviço e de infraestrutura. Com esses conjuntos de dados, finalmente o aprendizado de máquina foi empregado com o objetivo de treinar um modelo de predição para as métricas de qualidade de serviço. Utilizando o conjunto total de estatísticas disponíveis para o treinamento do modelo, demonstramos que é possível estimar, com uma margem de erro, os valores para as métricas de serviço.

Demonstramos ainda, depois de experimentos com diferentes tamanhos do conjunto de estatísticas usados no treinamento do modelo, que a redução do conjunto de estatísticas necessárias para o treinamento, por meio do método de filtragem das estatísticas melhor classificadas pelo algoritmo de aprendizado de máquina, é totalmente plausível, uma vez que a diminuição da qualidade das predições é pouco degradada e em certos casos até melhorada, além de se atingir um tempo de treinamento significativamente menor do que o observado quando utilizado o conjunto total de estatísticas.

Para gerar as estimativas, utilizou-se apenas um padrão de carga durante a coleta de dados, que foi o padrão sinusoidal periódico, dessa forma seria interessante para trabalhos futuros explorar a eficácia da predição de métricas para diferentes padrões de carga, tal como o *Flash Crowd*, onde cargas altas repentinas são geradas aleatoriamente. Outro aspecto que pode ser explorado futuramente é analisar o impacto causado pela utilização de diferentes tamanhos e tipos de dados, na estimativa das métricas.

Considerando que utilizamos apenas um método de aprendizado de máquina, pode ser interessante também o estudo e experimentação de outros métodos buscando aprimorar os resultados estimados. E por fim, para outro trabalho futuro, fica o objetivo de se construir uma ferramenta que realize o monitoramento e aplicação do modelo de aprendizado de máquina em tempo real, dando um passo a mais em direção à efetiva utilização deste trabalho na identificação e prevenção de desastres.

Resultados prévios desta monografia foram publicados anteriormente. Um artigo foi apresentado e publicado nos anais do XXXVII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos ([MARQUES et al., 2019](#)). Além disso, um pôster foi apresentado no XIII Workshop de Teses e Dissertações em Ciência da Computação ([CUNHA; PASQUINI, 2019](#)).

Referências

Apache, S. F. *What is Cassandra?* 2016. Disponível em: <<http://cassandra.apache.org/>>. Citado 2 vezes nas páginas 13 e 25.

BREIMAN, L. Random forests. *Machine Learning*, v. 45, n. 1, p. 5–32, Oct 2001. ISSN 1573-0565. Disponível em: <<https://doi.org/10.1023/A:1010933404324>>. Citado na página 36.

CUNHA, I. R. da; PASQUINI, R. Construção de mecanismo para suportar a predição de tempos de resposta do cassandra a partir de métricas da infraestrutura openstack. In: *Poster apresentado no XIII Workshop de Teses e Dissertações em Ciência da Computação*. UFU, 2019. Disponível em: <<http://www.eventos.ufu.br/wtdcc2019>>. Citado na página 39.

DataStax, I. *DataStax Distribution of Apache CassandraTM* 3.11. 2019. Disponível em: <<https://docs.datastax.com/en/ddac/doc/>>. Citado 3 vezes nas páginas 13, 14 e 15.

DAVOUDIAN, A.; CHEN, L.; LIU, M. A survey on nosql stores. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 51, n. 2, p. 40:1–40:43, abr. 2018. ISSN 0360-0300. Disponível em: <<http://doi.acm.org/10.1145/3158661>>. Citado na página 11.

FULMARI, A.; CHANDAK, M. B. A survey on supervised learning for word sense disambiguation. 2013. ISSN 2278-1021. Disponível em: <<https://pdfs.semanticscholar.org/58bb/8f4b9a0e7257ca15555e505e9fd35992f66c.pdf>>. Citado na página 18.

HEWITT, E. *Cassandra: The Definitive Guide*. 1. ed. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc., 2010. Citado 2 vezes nas páginas 16 e 17.

KOCHIE, B. 2018. Disponível em: <https://github.com/prometheus/node_exporter/releases>. Citado na página 31.

LAKSHMAN, A.; MALIK, P. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, ACM, New York, NY, USA, v. 44, n. 2, p. 35–40, abr. 2010. ISSN 0163-5980. Disponível em: <<http://doi.acm.org/10.1145/1773912.1773922>>. Citado 3 vezes nas páginas 13, 14 e 16.

LEARN scikit. 2018. Disponível em: <<https://scikit-learn.org/stable/>>. Citado na página 34.

Leavitt, N. Will nosql databases live up to their promise? *Computer*, v. 43, n. 2, p. 12–14, Feb 2010. ISSN 0018-9162. Citado na página 12.

MARQUES, G. S. et al. Arcabouço de um sistema inteligente de monitoramento para cloud slices. In: *Anais do WSlice 2019*. SBRC, 2019. p. 58–70. Disponível em: <<http://sbrc2019.sbc.org.br/wp-content/uploads/2019/05/wslice2019.pdf>>. Citado na página 39.

MATOS, J. R. F. *Construção de um serviço containerizado de vídeo sob demanda baseado em DASH para experimentação e coleta de métricas de desempenho*. Tese (Doutorado) —

Universidade Federal de Uberlândia, Uberlândia, Minas Gerais, Brasil, 2018. Disponível em: <<https://repositorio.ufu.br/handle/123456789/24068>>. Acesso em: 11 jul. 2019. Citado 2 vezes nas páginas 20 e 21.

MONIRUZZAMAN, A. B. M.; HOSSAIN, S. Nosql database: New era of databases for big data analytics - classification, characteristics and comparison. *Int J Database Theor Appl*, v. 6, 06 2013. Citado 2 vezes nas páginas 8 e 11.

OPENSTACK. *OpenStack Governance: Vision for OpenStack Clouds*. 2019. Disponível em: <https://governance.openstack.org/tc/reference/technical-vision.html?_ga=2.212067028.1721168066.1562879255-2088814429.1561406660>. Acesso em: 11 jul. 2019. Citado na página 19.

OUSSOUS, A. et al. Big data technologies: A survey. *Journal of King Saud University - Computer and Information Sciences*, v. 30, n. 4, p. 431 – 448, 2018. ISSN 1319-1578. Disponível em: <<http://www.sciencedirect.com/science/article/pii/S1319157817300034>>. Citado na página 8.

PERERA, S. *Considerações sobre o Banco de Dados Apache Cassandra*. 2012. Disponível em: <<https://www.ibm.com/developerworks/br/library/os-apache-cassandra/index.html>>. Citado 2 vezes nas páginas 4 e 17.

RED HAT, INC. *Introdução ao OpenStack*. 2019. Disponível em: <<https://www.redhat.com/pt-br/topics/openstack>>. Acesso em: 11 jul. 2019. Citado 2 vezes nas páginas 19 e 20.

STADLER, R.; PASQUINI, R.; FODOR, V. Learning from network device statistics. *Journal of Network and Systems Management*, v. 25, n. 4, p. 672–698, Oct 2017. ISSN 1573-7705. Disponível em: <<https://doi.org/10.1007/s10922-017-9426-z>>. Citado na página 20.

ZHU, X. *Semi-Supervised Learning Literature Survey*. [S.l.], 2005. Disponível em: <http://pages.cs.wisc.edu/~jerryzhu/pub/ssl_survey.pdf>. Citado na página 18.