UNIVERSIDADE DE CAXIAS DO SUL
CAMPUS DA REGIÃO DOS VINHEDOS
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

CASSIANO DA SILVA CARRARO

PROPOSTA DE ESTRATÉGIA PARA ANÁLISE DE ESCALABILIDADE DO SGBD MYSQL

BENTO GONÇALVES
2019

CASSIANO DA SILVA CARRARO

PROPOSTA DE ESTRATÉGIA PARA ANÁLISE DE ESCALABILIDADE DO SGBD MYSQL

Monografia apresentada como requisito para a obtenção do grau de Bacharel em Ciência da Computação da Universidade de Caxias do Sul.

Orientador Prof. Dr. Daniel Luis Notari

BENTO GONÇALVES

2019

CASSIANO DA SILVA CARRARO

PROPOSTA DE ESTRATÉGIA PARA ANÁLISE DE ESCALABILIDADE DO SGBD MYSQL

Monografia apresentada como requisito para a obtenção do grau de Bacharel em Ciência da Computação da Universidade de Caxias do Sul.

Aprovado(a) em 27/11/2019

Banca Examinadora

_____

Prof. Dr. Daniel Luis Notari
Universidade de Caxias do Sul – UCS

_____

Prof. Ma. Gabriele Dani
Universidade de Caxias do Sul – UCS

_____

Prof. Dra. Helena Graziottin Ribeiro
Universidade de Caxias do Sul – UCS

RESUMO

Base de dados possuem um papel fundamental no desenvolvimento de *software*. Com o avanço da utilização da computação em diversas áreas de atividade, a necessidade de fornecer capacidade de processamento, armazenamento e acesso passaram a ser essenciais para as aplicações. Nesse contexto, é comum as pessoas pensarem em escalabilidade do sistema somente quando se deparam com os efeitos colaterais relacionados ao aumento de carga de trabalho. Dessa forma, este trabalho propõem uma estratégia para definição da técnica de escalabilidade para sistemas de gerenciamento de base de dados (SGBD). O objetivo do trabalho é propor uma estratégia que possa ser utilizada como base para definição da técnica de escalabilidade mais adequada do MySQL, conforme a necessidade específica de cada aplicação. Para propor a estratégia, foram necessários o desenvolvimento de quatro etapas principais: configuração dos ambientes, execução de *benchmarks*, análise quantitativa e elaboração da proposta. A estratégia foi validada por meio de estudos de casos de aplicações obtidos de artigos relacionados. Como resultado da validação, foi possível identificar que a estratégia proposta garante uma avaliação inicial das técnicas de escalabilidade e orienta a avaliação contínua da arquitetura do SGBD.

Palavras-chave: Escalabilidade. MySQL. *Benchmark* MySQL.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Programming Interface*
AWS	*Amazon Web Services*
BDD	Base de Dados Distribuídas
DBA	*Database administrator*
DDL	*Data Definition Language*
ERP	*Enterprise Resource Planning*
E/S	Entrada/Saída
IaaS	*Infrastructure as a Service*
OLAP	*Online Analytical Processing*
OLTP	*Online Transaction Processing*
SGBD	Sistemas de Gestão de Base de Dados
SGBDD	Sistema de Gerenciamento de Banco de Dados Distribuídos
SO	Sistema Operacional
SQL	*Structured Query Language*
TPC	*Transaction Processing Performance Council*
TPM	*Transaction Per Minute*
TPS	*Transaction Per Second*
USL	*Universal Scalability Law*
WWW	*World Wide Web*

SUMÁRIO

1 INTRODUÇÃO

Os bancos de dados estão provocando um grande impacto no crescimento do uso de computadores. Eles representam um papel crítico em quase todas as áreas em que os computadores são utilizados, incluindo negócios, comércio eletrônico, engenharia, medicina, direito, educação e as ciências da informação (ELMASRI et al., 2011, p. 4). Com o avanço da utilização dessa tecnologia em diversas áreas de atividades, a necessidade de fornecer capacidade de processamento, armazenamento e acesso passaram a ser requisitos essenciais para as aplicações atuais.

Com base nesses requisitos, os Sistema de Gerenciamento de Banco de Dados (SGBD) desenvolveram, e estão aprimorando, ferramentas e técnicas que permitam o gerenciamento distribuído das bases de dados utilizando abordagens de sistema de banco de dados e redes de computadores. A maior motivação do uso de sistemas de banco de dados é o desejo de realizar a integração dos dados operacionais de um negócio e prover gerenciamento e acesso controlado de forma centralizada. Sendo assim o objetivo mais importante de uma tecnologia de banco de dados é a integração, não a centralização. Esses dois termos não interferem um ao outro, pois é possível alcançar a integração sem a centralização, sendo exatamente este o principal objetivo das tecnologias de Base de Dados Distribuídas (BDD) (ÖZSU et al., 2011, p. 2). ELMASRI et al. (2011, p. 578) complementa essa ideia afirmando que as organizações têm estado muito interessadas na descentralização do processamento em nível de sistema, enquanto obtêm uma integração dos recursos de informação a nível lógico dentro dos seus sistemas geograficamente distribuídos de banco de dados, aplicações e usuários.

Dessa forma, é de suma importância conhecer como funcionam as estratégias de escalabilidade e quais SGBDs disponibilizam ferramentas para que possam ser utilizadas nas aplicações. Neste trabalho será apresentado um estudo quantitativo sobre as estratégias de escalabilidade disponíveis no SGBD MySQL.

1.1 PROBLEMA DE PESQUISA

Esse trabalho pretende verificar a possibilidade da utilização de uma estratégia como base para definição da técnica de escalabilidade adotada de acordo com o tipo de aplicação. Entre as hipóteses da pesquisa estão:

- É possível utilizar um modelo de classificação para a escolha da estratégia de escalabilidade mais adequada?

1.2 OBJETIVO

O objetivo deste trabalho de pesquisa é propor uma estratégia que possa ser utilizada como base para definição da técnica de escalabilidade, de base de dados MySQL, mais adequada de acordo com a necessidade da aplicação. O objetivo será validado utilizando-se estudos de casos de aplicações, de artigos relacionados, para verificar se a estratégia é aplicável.

1.3 JUSTIFICATIVA

É comum as pessoas começarem a pensar em escalabilidade somente quando começam a enfrentar problemas em lidar com o aumento de carga, o que por sua vez acaba por gerar efeitos colaterais como limitação de E/S (Entrada/Saída), aumento de latência, problemas de memória e até travamento geral em caso de picos de trabalho. Para grandes volumes de dados e carga de trabalho elevada é importante selecionar uma estratégia apropriada para que a aplicação possa ser escalada de forma dinâmica e segura, assim, poupa-se dinheiro e tempo que poderão ser investidos em outras necessidades. Desse modo, com uma análise quantitativa pode-se expor os benefícios e limitações de cada estratégia disponível, gerando padrões e diretrizes que poderão ser utilizados pelos *database administrators* (DBA) na escolha da estratégia mais adequada para uma aplicação.

1.4 ESTRUTURA DO TRABALHO

O trabalho está estruturado em seis capítulos principais: introdução, fundamentação teórica, proposta de solução, análise de escalabilidade, proposta de estratégia e considerações finais. Na introdução é realizada a contextualização do tema do trabalho, sendo abordado o problema de pesquisa, acompanhado da justificativa e objetivos.

Após é apresentado o referencial teórico, que elenca alguns dos conceitos necessários para o entendimento do problema e da solução proposta. Sobre o problema são apresentados os conceitos relacionados a escalabilidade de banco de dados MySQL, acompanhado das estratégias que são avaliadas no decorrer do trabalho. Os conceitos envolvidos na solução estão descritos nas sessões de *benchmarking* e análise quantitativa.

No capítulo da proposta de solução, é explicado quais são as etapas planejadas que foram executadas no desenvolvimento, com o objetivo de resolver a questão de pesquisa. Também são apresentadas nesse capítulo, as ferramentas selecionadas para obtenção dos resultados dos *benchmarks,* e quais os princípios quantitativos que orientaram a análise das estratégias de escalabilidade.

No capítulo análise de escalabilidade, é realizado o detalhamento das etapas de configuração dos ambientes, execução dos *benchmarks* e apresentação dos resultados. Esse capítulo é composto por exemplos e explicações das ferramentas utilizadas e especificação dos testes executados. Por fim, são apresentadas as análises dos resultados por meio de gráficos comparativos.

No capítulo proposta de estratégia é realizado o desenvolvimento da estratégia, baseado na análise de escalabilidade, e a descrição dos trabalhos relacionados que foram utilizados para a validação da mesma. Por fim tem-se as considerações finais, que apresentam as conclusões e os trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Para atingir o objetivo da pesquisa é necessário a utilização de processo, metodologias e ferramentas. Nesse sentido, essa seção apresenta os conceitos das tecnologias utilizadas na pesquisa, fundamentadas nas leituras especializadas na área.

Na seção de princípios de bancos de dados distribuídos, são apresentados os principais conceitos relacionados a sistema de gerenciamento de banco de dados distribuído (SGBDD), com foco na escalabilidade, que servem para entendimento do contexto do trabalho. Na seção MySQL, é apresentado o SGBD, utilizado no trabalho, e um resumo de melhorias implementadas em cada versão. Após são apresentadas as estratégias de escalabilidade disponíveis para base de dados relacionais. Por fim, é abordado o assunto de *benchmarking* e análise quantitativa, que são os conhecimentos relacionados com a proposta de solução do problema.

2.1 PRINCÍPIOS DE BANCO DE DADOS DISTRIBUÍDOS

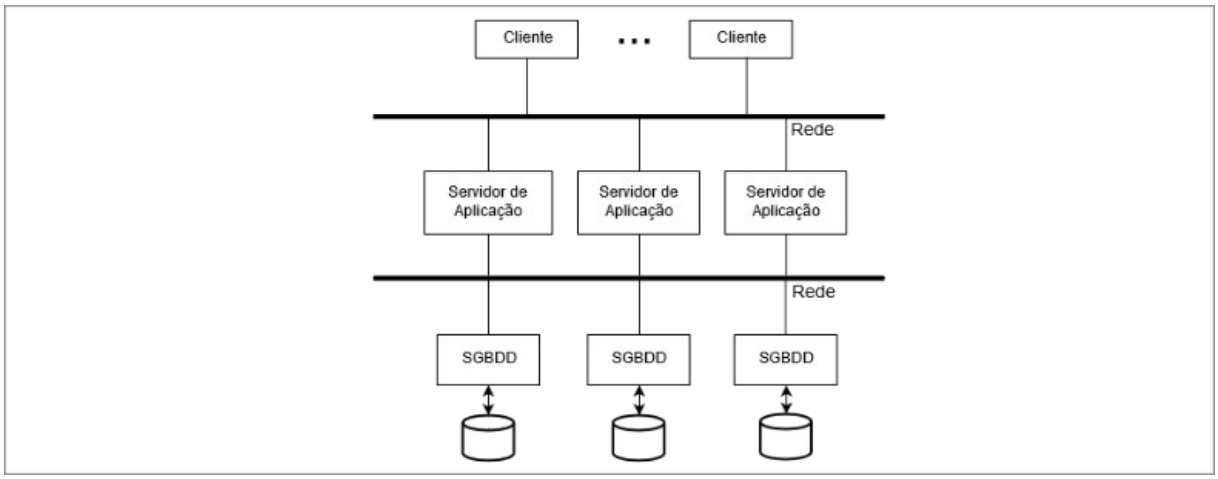
Pode-se definir um BDD como uma coleção de múltiplos bancos de dados logicamente inter-relacionados distribuídos por uma rede de computadores, e um SGBDD como um sistema de *software* que gerencia uma base de dados distribuída enquanto torna a distribuição transparente para o usuário (ÖZSU et al., 2011, p. 3). Este conceito é a base para entender como funcionam as estratégias atuais de escalabilidade fornecidas pelos SGBDs. De acordo com ELMASRI et al. (2011, p. 579), a tecnologia está se modificando em uma direção tal que a utilização de bancos de dados distribuídos na *World Wide Web* (WWW) se tornarão cada vez mais frequentes.

Para Özsu et al. (2011, p. 15), em um ambiente distribuído, é fácil de se acomodar bases de dados que estão em crescimento. Grandes revisões da arquitetura do sistema são raramente necessárias; a expansão pode ser gerenciada adicionando poder de processamento e armazenamento na rede.

A Figura 1 apresenta um exemplo de uma arquitetura que utiliza base de dados distribuídas. Na figura, há duas camadas de comunicação, a primeira é a

camada localizada entre os clientes e os servidores de aplicação. Essa camada é responsável por receber e interpretar as solicitações dos clientes. Já a camada entre os servidores de aplicação e os SGBDDs é a camada gerenciada pelo SGBDD, cujas aplicações se conectam sem ter conhecimento da estrutura distribuída que está por trás.

Figura 1 – Arquitetura de banco de dados distribuída



Fonte: Autor (2019).

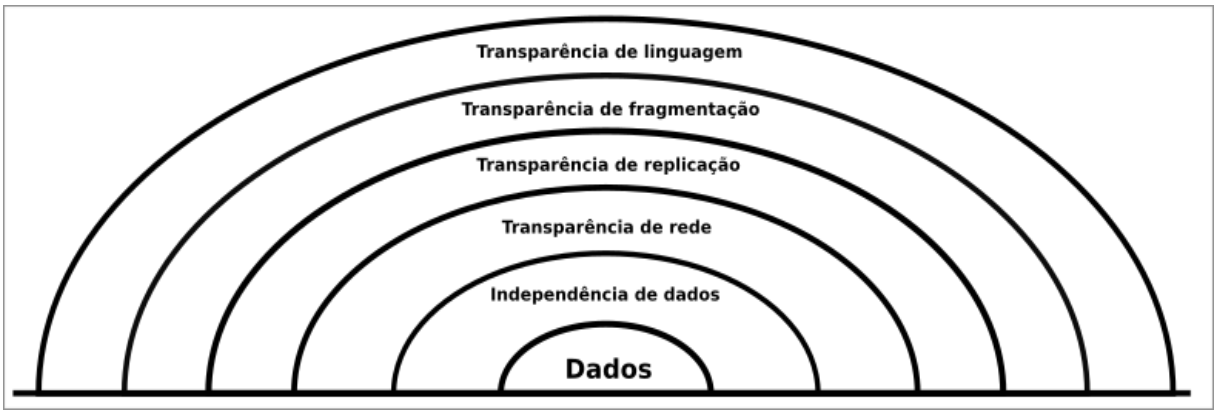
A utilização de bases de dados distribuídas possuem conflitos de escolha que devem ser considerados ao se optar pela utilização desse tipo de tecnologia. Entre as vantagens se destacam, o gerenciamento de dados distribuídos com níveis diferentes de transparência, melhoria na confiabilidade e disponibilidade, melhoria de desempenho e facilidade de expansão (ELMASRI et al., 2011, p. 580).

Na forma ideal um SGBDD deve ser transparente na distribuição física dos arquivos, tendo como os principais tipos de transparência a distribuição de rede e de fragmentação. A confiabilidade e disponibilidade refere-se, respectivamente, a capacidade do sistema estar em operação em um determinado momento e a probabilidade do sistema estar disponível continuamente. O aumento de tamanho dos banco de dados, ou acréscimo de mais processadores, é facilitado por estar lidando com um ambiente fragmentado (ELMASRI et al., 2011, p. 580).

O contraponto está na transparência, pois ela possui um compromisso entre a facilidade de uso e o custo da sobrecarga para proporcioná-la. Na situação de transparência total é proporcionado ao usuário uma visão do SGBDD inteiro como se fosse um único sistema centralizado (ÖZSU et al., 2011, p. 7).

A Figura 2 demonstra como estão dispostas as camadas de transparência, em um SGBDD com transparência total, elas são implementadas sobre a camada de dados. A camada de independência de dados refere-se a imunidade das aplicações a mudanças na definição e organização dos dados. A camada de transparência de rede está relacionada com a capacidade do SGBDD de gerenciar a comunicação de rede, de modo que o cliente não saiba da existência da mesma. Na camada de replicação deve-se garantir que os dados são replicados de modo a garantir a confiabilidade e disponibilidade, no entanto o cliente deve visualizar os dados como uma única cópia. A transparência de fragmentação é a capacidade do sistema de dividir as relações da base de dados em fragmentos menores e tratá-las separadamente. Por fim, está a camada de transparência de linguagem que deve prover acesso de alto nível aos dados (ÖZSU et al., 2011, p. 7).

Figura 2 – Camadas de transparência



Fonte: Autor (2011).

No entanto, o autor Özsu et al. (2011, p. 16), cita que a complexidade adicional, necessária por conta da transparência, dá origem a novos problemas influenciados por três fatores: replicação dos dados, possibilidade de algum nodo falhar e sincronização instantânea da informação entre os nodos.

Essas complicações são geradas pois, em um ambiente distribuído, deve-se garantir que a alteração dos dados seja aplicada em todos os nodos que possuem a cópia do dado. Caso ocorra algum problema durante a alteração de um dado, o sistema deve garantir que as alterações sejam refletidas nos nodos que falharam. Isso torna o gerenciamento dos dados mais complicado do que em um sistema centralizado (ÖZSU et al., 2011, p. 16).

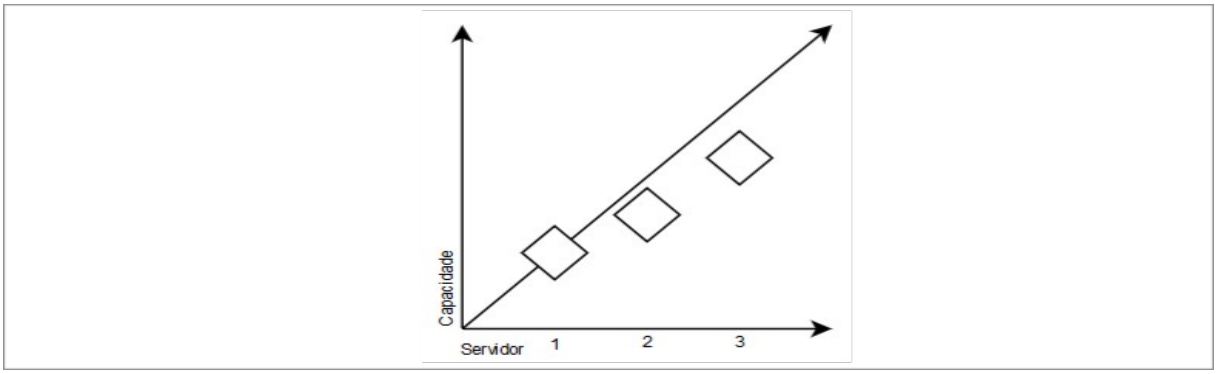
2.1.1 Definição de escalabilidade

Gunther (2007, p. 41), relaciona escalabilidade com a ciência de materiais, ele cita uma observação realizada por Galileo Galilei (1638) no Discursos Sobre as Duas Novas Ciências, que diz existir um limite natural na dimensão de estruturas físicas e a força inerente dos materiais não permite dimensões arbitrárias para objetos reais. Com isso, o autor complementa, explicando que a noção mais simples de escalabilidade é pegar todas as dimensões de um objeto e multiplicá-las pelo mesmo fator, por exemplo o dobro das dimensões, isso fará o objeto ter o dobro de capacidade.

Para Schwartz et al. (2012, p. 523), escalabilidade, no contexto de banco de dados, é a proporção na qual o sistema fornece um retorno sobre investimento à medida que recursos são adicionados para lidar com a carga e aumentar a capacidade. Essa definição é importante, pois o objetivo de todo sistema, no ponto de vista de capacidade, é fornecer um serviço de qualidade para os usuários com o menor custo possível.

No entanto, a escalabilidade de sistemas computacionais normalmente não se comporta de forma linear. A curva de escalabilidade é ligeiramente menor que a linear em pequenos fatores de escala, na medida que os fatores se tornam mais altos o desvio da linearidade se torna mais acentuado (SCHWARTZ et al., 2012, p. 525). A Figura 3 demonstra essa situação, onde a quantidade de servidores é o fator de escala e os quadrados representam o ganho de capacidade. Por conta desse comportamento algumas leis de poder surgiram para discutir os limites e possibilidades de escalabilidade de sistemas computacionais.

Figura 3 – Sistema com escalabilidade não linear



Fonte: Autor (2019).

2.1.1.1 Lei de Amdahl

A Lei de Amdahl, conforme citado por Patterson et al. (2014, p. 41), pode ser utilizada para calcular o ganho de desempenho obtido com a melhoria de algumas partes de um sistema. Essa lei estabelece que a melhoria de desempenho que pode ser atingida é limitada pela fração de tempo que essa parte otimizada pode ser utilizada. O ganho de desempenho, ou *speedup*, está relacionado diretamente com o ganho de velocidade na execução de uma tarefa por um sistema computacional podendo ser representado pela razão (2.1):

$$Ganho\,de\,velocidade = \frac{Desempenho\,para\,a\,tarefa\,sem\,melhoria\,(T_1)}{Desempenho\,para\,a\,tarefa\,utilizando\,a\,melhoria\,(T_p)} \quad (2.1)$$

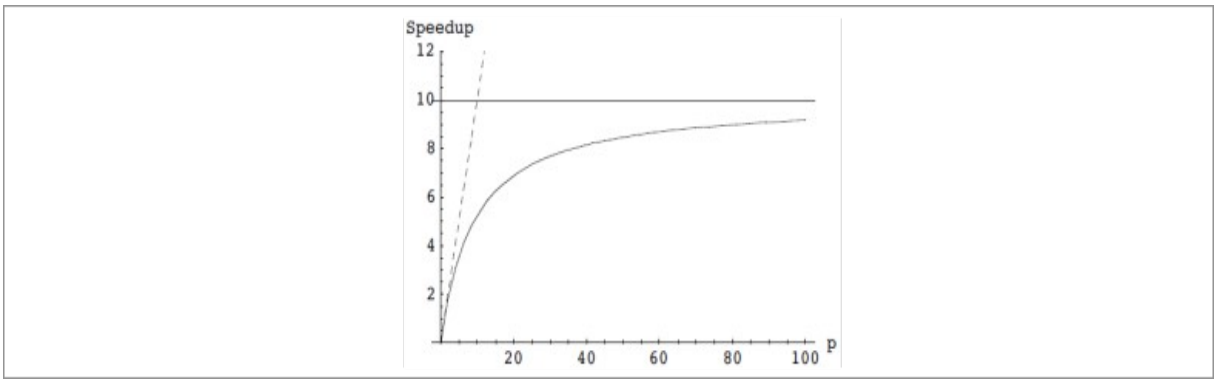
Gunther (2007, p. 49), explica que a Lei de Amdahl está baseada na execução de tarefas que podem ser executadas de forma paralela. A lei reconhece que o paralelismo ideal não pode ser alcançado em geral, pois há certas tarefas que só podem ser executadas sequencialmente. Amdahl (1967), concluiu que o esforço para se alcançar altas taxas de processamento paralelo é um desperdício, a menos que seja acompanhada de ganhos na taxa de processamento sequencial na mesma magnitude.

Dessa forma, o cálculo de *speedup* do Amdahl é definido como a razão do tempo de execução em um processamento único (T_1) sobre o tempo de execução em *p* processamentos (T_p), quanto menor for o tempo de execução T_p maior será o *speedup* alcançado. Portanto, para prever quanto poderá ser o *speedup* teórico alcançado utiliza-se (2.2) que possui como parâmetro a variável *p*, que indica a quantidade de unidades de processamento, e a constante σ que representa a fração do tempo gasto com processamento serial.

$$S(p) = \frac{p}{1 + \sigma(p-1)} \quad (2.2)$$

Sendo assim, a Figura 4 apresenta a característica da curva da função do cálculo de *speedup*, onde a linha tracejada representa a escalabilidade linear e a linha reta representa o limite assintótico da função levando em consideração um problema com fração serial de 10%.

Figura 4 – Característica da curva da função de *speedup* da Lei de Amdahl



Fonte: GUNTHER (2007).

Um dos objetivos da Lei de Amdahl é convencer as pessoas de que o processamento paralelo nem sempre é a opção de melhor custo-benefício, pois dependendo do problema a ser resolvido, há a dificuldade em se obter todo o potencial do processamento paralelo disponível. O autor Patterson indica que,

> […] a Lei de Amdahl pode servir de guia para o modo como uma melhoria incrementará o desempenho e como distribuir recursos para melhorar o custo-desempenho. O objetivo, claramente, é investir recursos proporcionais onde o tempo é gasto. A Lei de Amdahl é particularmente útil para comparar o desempenho geral do sistema de duas alternativas (PATTERSON et al., 2014, p. 41).

Com base nisso, pode-se aplicar essa lei em análises quantitativas, a fim de estimar qual será a capacidade que poderá ser atingida em um sistema adicionando mais nós de processamento. Entretanto, deve-se levar em consideração o tipo de problema que a aplicação se propõe a resolver e qual a porcentagem que pode ser resolvida de forma paralela.

2.1.1.2 *Universal Scalability Law* (USL)

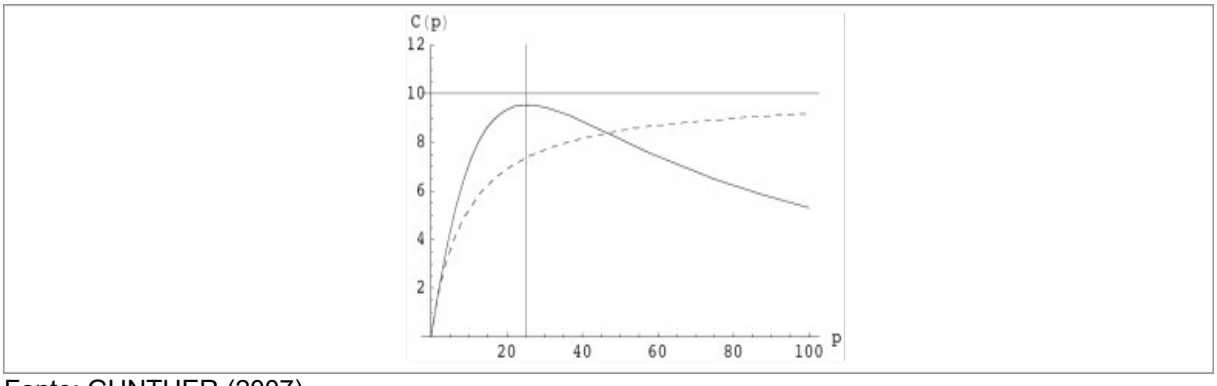
A Lei de Escalabilidade Universal foi apresentada por Neil J. Gunther e foi baseada na Lei de Amdahl. Ela leva em consideração a sobrecarga da comunicação entre nodos de processamento no caso de processamento paralelo, enquanto a Lei de Amdahl considera somente a degradação do tempo de execução baseado nos efeitos do processamento que deve ser realizado de forma serial. De acordo com Gunther (2007, p. 56), entre as origens de trabalho adicional em processamento paralelo, estão:

- Caminhos de código no SO (sistema operacional).
- Troca de dados gravados entre as memórias dos processadores.
- Troca de dados entre os processadores e a memória principal.
- Serialização de escrita de dados compartilhados para sincronização de acessos.
- Espera por instrução de E/S ou acesso à memória.

Schwartz et al. (2012, p. 525) simplifica a explicação da USL dizendo que o desvio da escalabilidade linear pode ser modelada por dois fatores: pela parte do trabalho que não pode ser realizada de forma paralela e pela porção de trabalho necessária para comunicação das unidades de processamento. O primeiro fator resulta no que foi definido na Lei de Amdahl, enquanto o segundo fator refere-se a questão de que o custo de comunicação depende diretamente do número de canais, que cresce de forma quadrática em relação ao número de nós do sistema. Sendo assim, eventualmente o custo pode crescer mais rápido do que o benefício, tendo-se então uma escalabilidade retrógrada.

A Figura 5, apresenta a comparação da aplicação da Lei de Amdahl, curva tracejada, e a função da USL, curva sólida, sobre um sistema hipotético, onde o atributo *p* é a quantidade de nodos e *C(p)* a capacidade do sistema. O principal ponto dessa comparação é que na USL um valor máximo de ganho será alcançado, que na imagem é quando o atributo *p* é igual a 25. Para valores maiores que 25 começará a ocorrer a degradação da capacidade disponível por conta do custo de comunicação.

Figura 5 – Comparação entre a função da curva da Lei de Amdhal e da USL



Fonte: GUNTHER (2007).

A USL incorpora três efeitos importantes no denominador em uma equação única, tendo como resultado a equação (2.3) para cálculo do *speedup* (GUNTHER, 2007, p. 58):

1. Concorrência: se não houver comunicação no processamento a função de capacidade deve escalar de forma linear, sendo parametrizado por *p.*
2. Contenção: representa o grau da serialização necessária para gerenciamento dos dados compartilhados que são graváveis, sendo parametrizado pela constante *σ*.
3. Coerência: representa a penalidade gerada por manter a consistência das variáveis compartilhadas, sendo parametrizado pela constante *k.* Quando a constante *k* = 0, a USL se reduz a escalabilidade de Amdahl.

$$C(p)=\frac{p}{1+\sigma(p-1)+kp(p-1)} \tag{2.3}$$

Desse modo, a utilização da USL se torna mais interessante e mais próxima ao mundo real do processamento distribuído de base de dados, justamente por permitir que a sobrecarga de comunicação e consistência dos dados nesse tipo de ambiente seja considerada no cálculo. No entanto, mesmo assim deve-se atentar aos parâmetros informados no cálculo para que eles reflitam as principais necessidades da aplicação, podendo assim identificar os principais pontos que poderão ser otimizados com a utilização do processamento distribuído.

Na próxima seção será apresentado o MySQL que é o SGBD utilizado para aplicação das estratégias de escalabilidade. O MySQL fornece meios para criação de base de dados distribuídas que serão testadas e comparadas utilizando os conceitos das leis de escalabilidade.

2.2 MYSQL

O MySQL é um SGBD relacional distribuído pela Oracle Corporation sobre duas licenças, GNU General Public License e licença comercial (MYSQL, 2019). Possui uma arquitetura flexível que funciona bem em ambientes com alta carga de

trabalho. Ele pode ser utilizado em aplicações *web,* aplicações embarcadas, *data warehouses,* sistemas de indexação de conteúdo, redundância, sistemas de transações *online*, entre outras. Sua arquitetura está disposta em camadas com os serviços e execução de consultas no topo e os mecanismos de armazenamento por baixo. Essa separação de responsabilidades permite escolher como os dados serão armazenados (SCHWARTZ et al., 2012, p. 1).

Atualmente está em 2º lugar no ranking de popularidade, como indicado na Figura 6[1], na categoria de base de dados relacionais, na classificação realizada pela empresa Solid IT. O método utilizado para gerar a pontuação leva em consideração o número de menções do SGBD na *web*, interesse geral, frequência de discussões técnicas, número de vagas de emprego cujo o SGBD é mencionado, número de menções em perfis em redes sociais de negócios e relevância nas redes sociais.

Figura 6 – Ranking de popularidade do SGBDs

Rank			DBMS	Database Model	Score		
Apr 2019	Mar 2019	Apr 2018			Apr 2019	Mar 2019	Apr 2018
1.	1.	1.	Oracle	Relational, Multi-model	1279.94	+0.80	-9.85
2.	2.	2.	MySQL	Relational, Multi-model	1215.14	+16.89	-11.26
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1059.96	+12.11	-35.55
4.	4.	4.	PostgreSQL	Relational, Multi-model	478.72	+8.91	+83.25
5.	5.	5.	IBM Db2	Relational, Multi-model	176.05	-1.15	-12.89
6.	6.	6.	Microsoft Access	Relational	144.65	-1.55	+12.43
7.	7.	7.	SQLite	Relational	124.21	-0.66	+8.23
8.	8.	↑ 9.	MariaDB	Relational, Multi-model	85.23	+0.92	+20.67
9.	9.	↓ 8.	Teradata	Relational	75.35	+0.13	+1.67
10.	10.	↑ 11.	Hive	Relational	74.71	+1.71	+17.31

Fonte: Solid IT (2019).

2.2.1 Evolução das versões

Desde seu lançamento o MySQL passou por diversas modificações contendo correções de *bugs*, adição de novas funcionalidades, otimizações, depreciação de funcionalidades e melhorias de segurança. A distribuição atual do SGBD é composta por três versões principais, que atualmente ainda recebem suporte, MySQL 5.6[2],

1 Disponível em: https://db-engines.com/en/ranking/relational+dbms.
2 https://dev.mysql.com/doc/refman/5.6/en/mysql-nutshell.html.

MySQL 5.7[3] e MySQL 8.0[4]. O Quadro 1 apresenta algumas funcionalidades relevantes e os principais pontos que estão relacionados com a questão de escalabilidade de cada versão.

O suporte do MySQL atualmente é fornecido pela equipe da Oracle, que disponibiliza uma equipe de especialistas para trabalhar nas melhorias de cada versão do SGBD. As atualizações no *software* são liberadas para todos os usuários de forma gratuita, no entanto para um suporte técnico especializado é necessário contratar um dos planos fornecidos pela Oracle.

2.3 ESTRATÉGIAS DE ESCALABILIDADE

Existem duas abordagens principais quando fala-se de escalabilidade em banco de dados, sendo escalabilidade vertical e horizontal. Na escalabilidade vertical adiciona-se recurso de *hardware*, em um único servidor, de acordo com a necessidade da aplicação. Segundo Schwartz et al. (2012, p. 529), um único servidor é mais simples de se manter comparado com uma arquitetura com múltiplos servidores por conta de questões como *backup*, consistência e gerenciamento. No entanto, o autor enfatiza que se a utilização da aplicação crescer, essa abordagem não funcionará por conta do custo e também por conta da limitação dos computadores.

Na escalabilidade horizontal a ideia principal é a distribuição de trabalho entre múltiplos servidores. Atualmente o MySQL fornece suporte para três principais grupos de táticas de escalabilidade: replicação, particionamento funcional, *sharding* e *clustering*.

3 https://dev.mysql.com/doc/refman/5.7/en/mysql-nutshell.html.
4 https://dev.mysql.com/doc/refman/8.0/en/mysql-nutshell.html

Quadro 1 – Evolução das versões do MySQL

MySQL 5.6	MySQL 5.7	MySQL 8.0
Otimizações no *InnoDB*[5] que reduziram o uso de recursos do SO.	Suporte nativo a colunas do tipo JSON.	Dicionário de dados transacional, permitindo a execução atômica de instruções de Data Definition Language (DDL).
Responsabilidade da gravação dos dados em disco movida da *thread* principal para uma *thread* separada, permitindo paralelizar operações de E/S.	Suporte a replicação múltiplas origens (*multi-master*).	Adicionado funções de agregação, suporte a expressões XPath[6] e atualização parcial de valores para colunas do tipo JSON.
Lançamento do MySQL NDB *Cluster*[7] como um produto separado, baseado do MySQL 5.6.	Suporte a múltiplas *threads* para gravação de dados de *buffer* em disco para o mecanismo de armazenamento *InnoDB*.	Suporte para atualizações parciais em campos JSON via replicação.
Suporte *multithreading* na replicação de dados nos nodos *slaves.*		
Suporte a replicação baseado em um identificador global de transações. Facilita a questão de disponibilidade do sistema, pois com a falha de um *master* é possível promover um servidor *slave*, sincronizado, para a função.		

Fonte: Autor (2019).

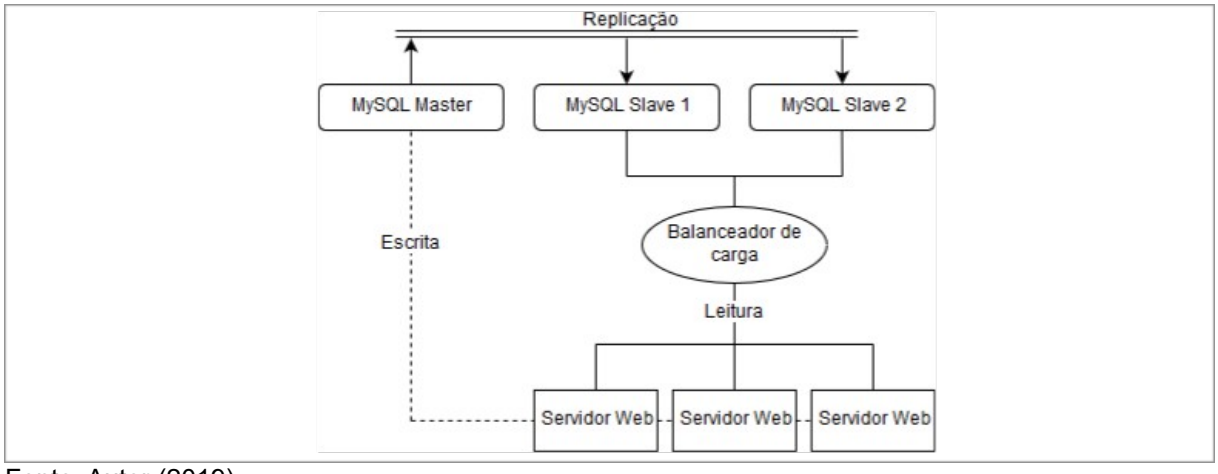
5 *InnoDB* é um mecanismo de armazenamento de dados para MySQL.
6 Linguagem de consulta que pode ser utilizada para selecionar nós dos documentos JSON.
7 MySQL NDB (*Network Database*) Cluster é um SGBDD que utiliza uma combinação de duas tecnologias: base de dados distribuída e um mecanismo de armazenamento MySQL.

2.3.1 Replicação

A replicação é a estratégia mais comum e simples de escalabilidade horizontal, nessa técnica os dados são enviados para múltiplos servidores. Essa técnica pode funcionar bem para aplicações com perfil de carga de leitura, as desvantagens são a duplicação de *cache* e o custo de armazenamento dos dados replicados (SCHWARTZ et al., 2012, p. 531).

Conforme a documentação do MySQL (2019), a replicação permite a cópia dos dados de um servidor que recebe escrita (denominado como *master*), para outros servidores MySQL (denominado como *slave*), a Figura 7 exemplifica um ambiente que utiliza esta técnica. No MySQL a replicação é assíncrona por padrão, isso indica que os servidores *slaves* não precisam estar conectados permanentemente para receber as atualizações dos *masters*. Outro tipo de replicação suportada é a semissíncrona, nesse caso um *commit* disparado no *master* é bloqueado, na sessão que disparou a transação, até que um dos servidores *slaves* confirme que recebeu e gravou em seus *logs* os eventos da transação. Por fim, o MySQL também permite a replicação com atraso, onde um tempo de atraso pode ser configurado para determinar em que momento os *slaves* devem aplicar as alterações, esse tipo de replicação pode ser utilizado em aplicações que permitam ao usuário retornar alterações realizadas nos dados.

Figura 7 – Exemplo de utilização da replicação



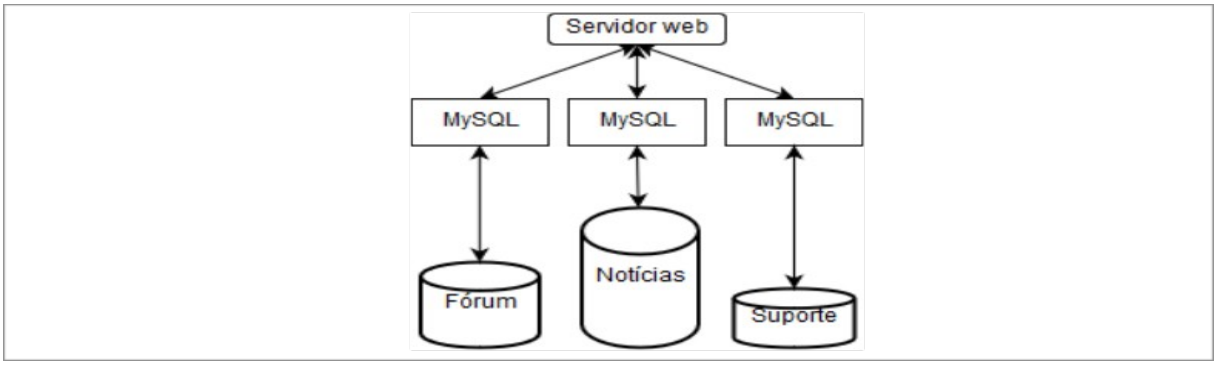
Fonte: Autor (2019).

Todas as técnicas de replicação funcionam por meio transmissão do *binary log*[8] dos *masters* para os servidores *slaves.* Nesse arquivo são enviados os eventos das transações executadas, e os *slaves* precisam manter seus próprios arquivos de *logs* e posições de leitura sincronizados (MYSQL, 2019).

2.3.2 Particionamento funcional

De acordo com Schwartz et al. (2012, p. 531), no particionamento funcional os servidores são divididos para executarem diferentes tarefas dentro da aplicação. Sendo assim, cada SGBD separado gerenciará informações somente da sua área específica. Essa técnica é interessante para se utilizar caso a aplicação possua grupos de tabelas que não estão relacionadas. No entanto, esse tipo de abordagem possui algumas limitações como: toda a aplicação terá um limite de particionamento funcional e uma área específica poderá crescer muito ao ponto de ser necessário utilizar outra técnica. A Figura 8 demonstra um exemplo de estrutura que utiliza particionamento funcional e poderia ser aplicado em um site de notícias.

Figura 8 – Exemplo de utilização do particionamento funcional



Fonte: Autor (2019).

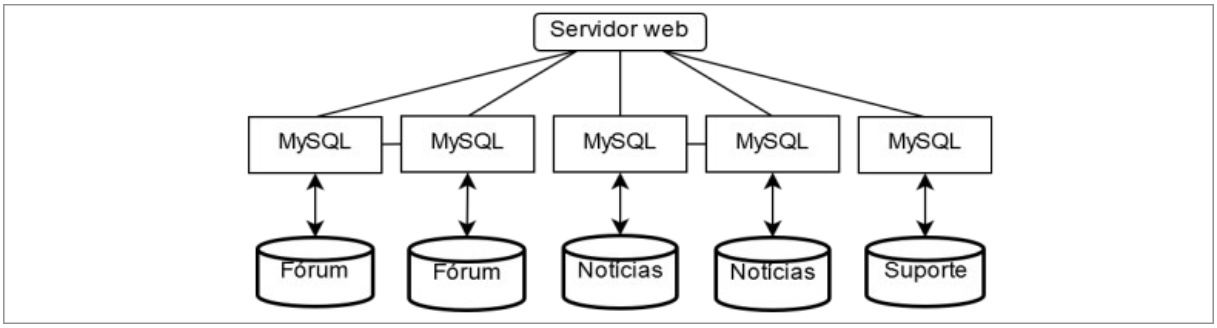
2.3.3 *Data sharding*

A estratégia de *data sharding* consiste na divisão dos dados em partes menores que serão distribuídas na quantidade de nodos disponíveis, conforme demonstrado na Figura 9. Para Schwartz et al. (2012, p. 533), esta é a estratégia

8 O *binary log* contém os eventos que descrevem as alterações realizadas no banco de dados, como alterações de estrutura e nos dados.

mais comum e bem-sucedida para escalar aplicações MySQL com grandes volumes de dados. No entanto, para a divisão dos dados, recomenda-se levar em consideração os conjuntos de dados que crescerão de forma rápida. O autor complementa dizendo que, o desenvolvimento dessa estratégia é mais complexa, comparado com as já vistas, e que normalmente é utilizada em situações que há a necessidade de escalar operações de escrita de uma aplicação.

Figura 9 – Exemplo de utilização de *data sharding*



Fonte: Autor (2019).

Já Özsu et al. (2011, p. 75), relata que o ponto principal da fragmentação dos dados é a seleção da chave que será utilizada como parâmetro para a distribuição. Ele afirma que as relações não são chaves adequadas, por três motivos principais:

1. Os dados das aplicações são geralmente vistos como subconjuntos de relações, portanto a localidade de acessos não é definido em suas relações mas em seus subconjuntos. Por isso, é natural considerar subconjuntos de relações como unidade de distribuição.
2. Se a aplicação tiver a visão definida baseando-se nas suas relações, há alternativas para se realizar a fragmentação. No entanto, a estratégia resultará em um alto volume de acessos de dados remotos. Ou haverá replicação desnecessária que poderá causar problemas na execução de atualizações, o que não é desejável se o armazenamento é limitado.
3. A decomposição dos dados por meio de sua relação resultará em uma execução paralela de uma única consulta dividindo-a em subconsultas que deverão operar sobre cada fragmento.

Ambos os autores confirmam a ideia de que a estratégia de fragmentação dos dados geram novas dificuldades e adicionam novas variáveis de custo para gravação e principalmente consulta de dados. Özsu et al. (2011, p. 75) comenta que a aplicação pode sofrer degradação do desempenho quando há muita fragmentação sendo necessário juntar os resultados de cada fragmento. Schwartz et al. (2012, p. 535), chega direto ao ponto, respondendo a questão de quando deve-se fragmentar os dados, para ele a fragmentação não deve ocorrer a menos que seja necessário, ou seja, quando os arquivos de dados ou a carga de escrita se tornam insustentáveis para um único servidor.

2.3.4 *Clustering*

Esta estratégia de escalabilidade é o cenário ideal para escalar base de dados, pois a ideia central dessa abordagem é fornecer um sistema no estilo caixa-preta, onde a aplicação entenda esse sistema como um único servidor centralizado, sem ter conhecimento da implementação distribuída. Dessa forma, as alterações para escalar o SGBD ficam centralizadas unicamente na base de dados, sem interferir no funcionamento da aplicação. No entanto, deve-se salientar o teorema de CAP (*Consistency*, *Availability*, *Partition tolerance*), Figura 10, que explica as três propriedades desejadas quando há o desenvolvimento de serviços distribuídos, sendo elas: consistência, disponibilidade e tolerância a particionamento. De acordo com essa conjectura é impossível possuir as três propriedades em um mesmo instante, é possível somente escolher duas (BREWER, 2012, p. 23).

Figura 10 – Teorema CAP

Fonte: Autor (2019).

Conforme explica Schwartz et al. (2012, p. 549), o MySQL garante as propriedades de consistência e disponibilidade em sua essência. No entanto, novas tecnologias denominadas como *NewSQL* estão implementando a ideia de banco de dados transacional distribuído. Como exemplo tem-se o MySQL *Cluster* (também conhecido como NDB *Cluster*)[9] que é um banco de dados escalável que pode ser acessado através de uma API nativa, parecido com o sistema de bases *NoSQL*, com a diferença que a comunicação é realizada com SQL quando adicionado um mecanismo de armazenamento do MySQL. Isso permite a distribuição total, alta performance, compartilhamento automático e um banco de dados transacional.

O MySQL *Cluster* foi desenvolvido para suportar ambientes com alta taxa de escrita e leitura com a capacidade de escalar os nós dinamicamente, permitindo com que aplicações possam dimensionar seus recursos de acordo com sua demanda. Sua arquitetura é composta por três tipos de nodos: nodos de dados, de aplicação e de gerenciamento (ORACLE, 2016, p. 4).

Os nodos de dados gerenciam o armazenamento e acesso aos dados, os dados são automaticamente distribuídos por meio desses nodos que gerenciam de forma transparente a distribuição de carga, replicação, detecção e prevenção de erros. Os nodos de aplicação fornecem o meio de comunicação entre a lógica de aplicação e os nodos de dados, utilizando a interface padrão SQL. Por fim, os nodos de gerenciamento são utilizados para configurar o *cluster* e arbitrar eventos de particionamento (ORACLE, 2016, p. 5).

Portanto, as técnicas de gerenciamento de dados paralelos estendem as técnicas de base de dados distribuídas para se obter alta performance, alta disponibilidade e extensibilidade. As soluções de gerenciamento de transações podem ser reutilizadas, como por exemplo, controle de concorrência, confiabilidade, atomicidade e replicação. No entanto, entre os pontos críticos desse tipo de estratégia estão questões como localização dos dados, execução paralela de consultas, processamento paralelo de dados, otimização de consultas paralelas e distribuição de carga (ÖZSU et al., 2011, p. 547). Engenheiros de banco de dados trabalham para buscar as soluções para essas questões críticas, com o objetivo de fornecer um ambiente de *cluster* estável melhorando a performance e disponibilidade, por meio da capacidade da replicação de dados. Entre os resultados

9 https://dev.mysql.com/doc/refman/8.0/en/mysql-cluster-overview.html

dessas soluções estão novas técnicas de replicação, distribuição de carga, processamento de consultas e tolerância a falhas.

2.4 *BENCHMARKING*

No meio de arquitetura de computadores, *benchmarking* é um processo que compara alternativas de projeto, quando normalmente deseja relacionar o desempenho de dois computadores diferentes para saber qual é mais rápido (PATTERSON et al., 2014, p. 32). Esse processo também pode ser utilizado para comparar o desempenho de cada estratégia de escalabilidade executada em diferentes tipos de aplicações.

Schwartz et al. (2012, p. 35), explica que o processo de *benchmarking* é importante pois é excepcionalmente conveniente e efetivo para avaliar o que acontece em um sistema quando é executado uma carga de trabalho. Conforme suas palavras,

> Um *benchmark* pode auxiliar na observação do comportamento do sistema sob carga, determinar a capacidade do sistema, saber quais as alterações são importantes ou ver o desempenho da aplicação com dados diferentes. O *benchmarking* permite criar circunstâncias fictícias, além das condições reais que podem ser observadas (SCHWARTZ et al., 2012, p. 35, tradução nossa).

Como cita Patterson et al. (2014, p. 32), dizer que um computador é mais rápido que o outro é uma afirmação ambígua, pois para o usuário de um computador *desktop* mais rápido pode-se referir a um programa rodar em menos tempo, já para um administrador de banco de dados pode-se referir a quantidade de transações por hora que o sistema é capaz de executar. Dessa forma, normalmente os usuários que executam rotineiramente os programas são os candidatos perfeitos para avaliar um novo computador ou estratégia, pois eles simplesmente comparam como o sistema opera com a carga de trabalho utilizando uma nova abordagem. No entanto, essa abordagem é pouco utilizada na realidade, atualmente utilizam-se outros métodos para avaliar as estratégias, esperando que elas prevejam o desempenho da utilização em aplicações reais.

Para se desenvolver um *benchmarking* deve-se definir o que será medido durante a execução, qual será o padrão da carga de trabalho, em que condições

serão executados e de que forma o resultado deverá ser apresentado a fim de fornecer informações úteis sobre a execução. Patterson et al. (2014, p. 34) comenta que séries de aplicações de *benchmark,* chamadas pacotes de *benchmark,* são uma medida popular do desempenho dos processadores com uma variedade de aplicações. O objetivo desses pacotes é caracterizar o desempenho relativo de dois computadores, particularmente para programas não incluídos, que os clientes provavelmente usarão.

No ambiente de banco de dados as ferramentas de *benchmark* podem ser classificadas em dois grupos: *full-stack,* que se encarregam de testar a aplicação como um todo; e *single-component,* cujo o foco é testar somente o serviço do SGBD. Schwartz et al. (2012, p. 37) comenta a possibilidade do desenvolvimento de ferramentas próprias ou então a utilização de pacotes de *benchmark* padrões.

Entre os pacotes de *benchmark full-stack* estão as ferramentas *ab*[10], específica para executar requisições no servidor *web* Apache; *http_load*[11], possui o mesmo conceito da *ab*, no entanto, é mais flexível e pode executar requisições em outros servidores *web*; e JMeter[12], que pode simular a utilização de uma aplicação e medir sua performance (SCHWARTZ et al., 2012, p. 51).

Já na categoria *single-component* existem as ferramentas, *mysqlslap*[13]*,* simula carga concorrente no servidor MySQL e reporta informações de tempo de execução; *MySQL Benchmark Suite (sql-bench)*[14]*,* simula carga com uma única conexão, o benefício dessa ferramenta é que ela possui vários testes predefinidos e é de fácil utilização, pode ser interessante para comparações mais superficiais; *Database Test Suite*[15]*,* propõem um kit de testes que são utilizadas para alguns *benchmarks* padrões da indústria; *Percona's TPCC-MySQL Tool*[16]*,* implementa os conceitos do teste TPC-C com ferramentas específicas desenvolvidas para *benchmarking* do MySQL; *sysbench*[17]*,* é uma ferramenta *multithreaded flexível,* permite a utilização de *scripts* em *Lua*, com foco na obtenção da performance do banco de dados, SO e desempenho de *hardware* (SCHWARTZ et al., 2012, p. 51).

10 http://httpd.apache.org/docs/2.0/programs/ab.html.
11 http://www.acme.com/software/http_load/.
12 http://jmeter.apache.org/
13 https://dev.mysql.com/doc/refman/8.0/en/mysqlslap.html
14 https://dev.mysql.com/doc/refman/5.5/en/mysql-benchmarks.html
15 https://sourceforge.net/projects/osdldbt/
16 https://github.com/Percona-Lab/tpcc-mysql
17 https://wiki.gentoo.org/wiki/Sysbench

Por fim, é importante salientar que o princípio orientador da apresentação dos relatórios de medições de desempenho devem possuir a propriedade de serem reproduzíveis. Além das descrições de *hardware, software* e linha de referência é essencial que sejam demonstrados os tempos de desempenho real, representados tanto em formato de tabulação quanto como gráfico (PATTERSON et al., 2014, p. 37).

2.4.1 Análise quantitativa

Os resultados das execuções das ferramentas de *benchmarking* são dados brutos da performance do sistema durante a execução da carga de trabalho, para que seja possível obter conhecimento a partir deste dados é necessário realizar uma análise baseado em alguns princípios quantitativos. Schwartz et al. (2012, p. 49) fala sobre a importância da utilização de recursos gráficos para a visualização dos resultados, pois é possível detectar um comportamento instantaneamente visualizando um gráfico, quando seria difícil ou impossível detectar apenas examinando os dados brutos.

Outra questão importante a se considerar na análise quantitativa são os métodos matemáticos aplicados sobre os dados brutos. Gunther, (2007, p. 27) explica a importância dos dígitos significativos, da definição das regras de arrendondamento, da definição dos valores calculados de modo que façam sentido e da representação dos erros de medição e cálculo. Um dígito significativo é aquele que é realmente medido, já o número de dígitos significativos em uma medição depende do dispositivo de medição. Não importa qual seja o dispositivo, sempre haverá alguma incerteza na medição seja por parte do dispositivo ou do observador. Com isso, o autor cita um exemplo de uma medição cujo os resultados são 1100, 11, 0.011 o que distingue os valores são a escala, a precisão é a mesma em todos os casos, dois dígitos significativos, mas a escala ou unidade de medida que eles referem-se são diferentes. Portanto, todas as medições de performance contém erros que precisam ser monitorados para evitar conclusões equivocadas e resultados enganosos.

No contexto da análise quantitativa de arquiteturas computacionais, é importante resumir os resultados das medições geradas por ferramentas de *benchmark* em um único número. Uma técnica simples para um cálculo resumido seria comparar as médias aritméticas dos tempos de execução dos programas no pacote. No entanto, alguns programas podem levar mais tempo que outros para executar, assim esses programas seriam mais importantes se média fosse o único número utilizado para resumir o desempenho. Uma alternativa seria a utilização de pesos para cada programa para que pudesse ser calculada uma média aritmética ponderada, mas o novo problema seria como selecionar os pesos. Em vez de utilizar pesos, pode-se utilizar outra abordagem onde os tempos de execução são normalizados para um computador de referência, dividindo o tempo no computador de referência pelo tempo do computador avaliado, gerando uma razão proporcional ao desempenho (PATTERSON et al., 2014, p. 37). Essa abordagem da razão proporcional é interessante para aplicação nas análises das estratégias de escalabilidade, pois vários programas de *benchmark* serão executados sobre os ambientes em cada estratégia e o resumo do resultado apresentará a variação de ganho de desempenho e capacidade.

2.5 TRABALHOS RELACIONADOS

Entre os trabalhos relacionados pesquisados, destacam-se os seguintes objetivos: avaliação de desempenho, comparativos com base de dados não relacionais e estudos de casos. Os trabalhos de avaliação de desempenho auxiliam no entendimento de como são aplicados os *benchmarks* e como os resultados são apresentados, de acordo com a relevância de cada medição. Já os comparativos com base de dados não relacionais, trazem resultados de comparações com sistemas de paradigmas diferentes. E os estudos de casos agregam informações e resultados da utilização do SGBD em problemas reais.

Dessa forma, os trabalhos selecionados, estão listados no Quadro 2, com o seu título, objetivo e respectiva referência. Eles serão utilizados no desenvolvimento e validação da proposta.

Quadro 2 – Trabalhos relacionados

Título	Objetivo
An Improvement in MySQL Cluster in E-commerce Scenarios (XIANG, 2014)	Analisar o cenário de utilização de sistemas de *e-commerce* e propor um modelo de arquitetura que satisfaça o teorema CAP em diferentes aspectos do negócio de *e-commerce*.
Performance Evaluation of MySQL, Cassandra and HBase for Heavy Write Operation (JOGI et al., 2016)	Avaliar a performance de aplicações, com o perfil de carga de trabalho de escrita, utilizando SGBDs de diferentes paradigmas.

2.6 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados conceitos teóricos sobre princípios de base de dados distribuídas, MySQL, *benchmarking* e análise quantitativa. Esses tópicos são essenciais para o entendimento do problema e para a orientação das próximas etapas do trabalho que envolvem a proposta de solução, aplicação e validação dos objetivos.

3 PROPOSTA DE SOLUÇÃO

A escolha do modelo de escalabilidade é uma etapa importante na definição da arquitetura utilizada para o gerenciamento de dados de uma aplicação. Como cita Schwartz et al. (2012, p. 527), utilizar uma única instância do SGBD, e comparar *hardware* mais potente, conforme a demanda aumenta, não é a solução mais adequada quando se fala em aplicações de larga escala. A tendência dessa abordagem é que em um determinado momento o retorno sobre o investimento não será satisfatório. Sendo assim, a utilização de uma estratégia, como base para a definição da técnica de escalabilidade, poderá orientar o trabalho de DBAs na escolha da técnica mais adequada conforme as características da aplicação.

Entre as características necessárias, para serem utilizadas na estratégia, estão: *throughput*; característica do crescimento do volume de dados, podendo ser exponencial ou linear; padrão da carga de trabalho, considerando a relação de operações de escritas sobre as operações de leituras; e se o processamento é OLTP ou OLAP (*Online Analytical Processing*[18]). Esse conjunto de caraterísticas deve ser de conhecimento de quem pretende utilizar a estratégia.

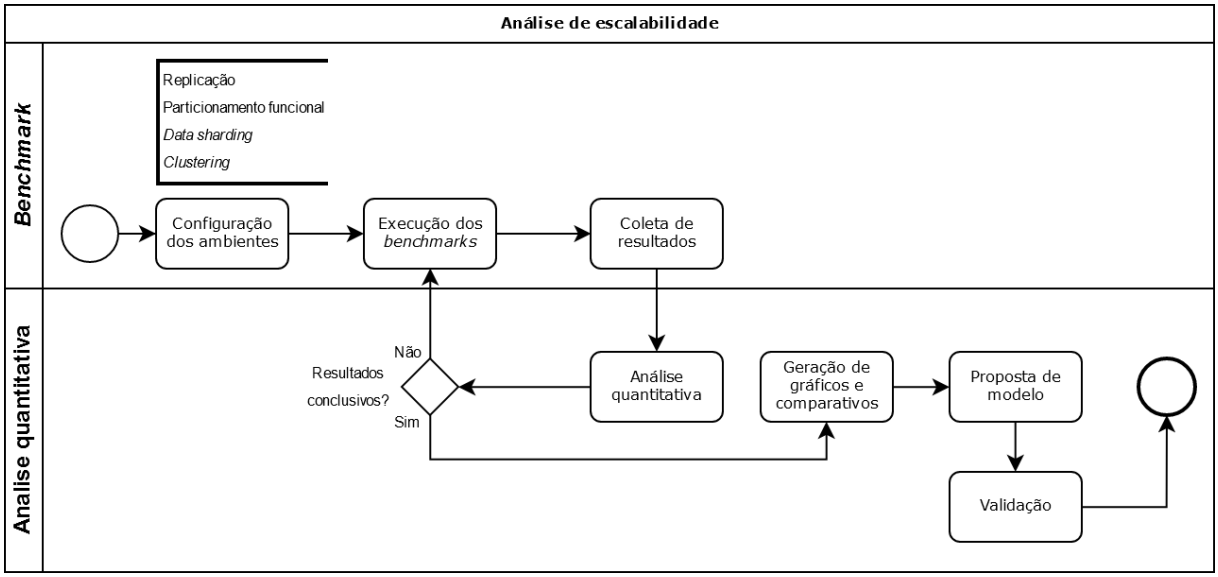
Para propor a estratégia serão utilizadas ferramentas de *benchmarking*, para avaliar o desempenho de cada técnica em determinadas circunstâncias, e a análise quantitativa para orientar a escolha da melhor solução de escalabilidade. As estratégias que serão avaliadas foram apresentadas no tópico de Estratégias de Escalabilidade da seção Referencial Teórico, sendo elas: replicação, particionamento funcional, *data sharding* e *clustering*. De acordo com Schwartz et al. (2012, p. 531), atualmente essas são as principais estratégias consideradas quando se fala em escalabilidade de base de dados.

A Figura 11 apresenta o fluxograma das atividades a serem realizadas no desenvolvimento da proposta. O desenvolvimento será iniciado com a configuração e preparação dos ambientes com cada estratégia de escalabilidade selecionada. Após serão executados os *benchmarks* em cada ambiente e coletados os resultados. Com os resultados será possível iniciar a análise quantitativa e gerar os

18 Cargas de trabalho OLAP estão relacionadas a um perfil de aplicação como da área de inteligência empresarial e inteligência artificial. A principal característica desse perfil é o grande volume de dados que armazenam histórico de informações e normalmente não são voláteis.

conhecimentos necessários para propor a estratégia. Caso não seja possível gerar resultados conclusivos, na análise quantitativa, será necessário executar novamente os *benchmarks* com os parâmetros de execução ajustados. Por fim, será realizada a elaboração da proposta de estratégia e a validação.

Figura 11 – Fluxograma de atividades



Fonte: Autor (2019).

3.1 ATIVIDADES

Entre as atividades a serem executadas na solução estão: configuração dos ambientes para execução dos *benchmarks*, execução dos *benchmarks*, análise quantitativa e por fim a conclusão da análise com a validação da estratégia proposta em estudos de casos reais. A execução de cada passo está no modelo cascata, onde o resultado de cada tarefa será a base para as tarefas posteriores.

3.1.1 Configuração dos ambientes

Para a configuração das estratégias será utilizado um ambiente na nuvem que fornece serviços do tipo IaaS (*Infrastructure as a Service*). Esse tipo de serviço será utilizado, pois atualmente é a base de infraestrutura utilizada em aplicações *web*. Como citado por Patterson et al. (2014, p. 404), além do baixo custo de um

modelo de pagamento por uso para computação de serviços, outro ponto atraente para os usuários de computação em nuvem é que os provedores assumem os riscos do provisionamento em excesso ou insuficiente, isso é um ponto atraente para as empresas, principalmente *start-up*, pois não precisam se preocupar com o gerenciamento e investimento em *hardware*.

Outro ponto positivo desse tipo de tecnologia é a flexibilização e facilidade do gerenciamento de recursos de *hardware* e *software*. Isso facilita a configuração dos ambientes, pois é possível criar uma imagem com uma configuração uniforme do SGBD, que será utilizada como base para todos os ambientes, e a partir dessa base realiza-se as configurações específicas de cada estratégia de escalabilidade.

3.1.2 Execução dos *benchmarks*

Após a configuração de cada ambiente, serão executados os *benchmarks*, com as ferramentas selecionadas, descritas na sessão Ferramentas de *Benchmark* deste capítulo. Os resultados coletados serão a base para o passo posterior de análise quantitativa. Conforme cita Schwartz et al. (2012, p. 41), o primeiro passo para o planejamento de um *benchmark* é identificar o problema e o objetivo, depois deve-se optar por utilizar um *benchmark* padrão ou desenvolver um próprio. Sendo assim, o problema é a limitação de capacidade das estratégias de escalabilidade, conforme o crescimento da demanda da aplicação, e o objetivo é identificar essas limitações.

Entre as limitações de cada estratégia espera-se encontrar problemas relacionados ao planejamento de capacidade. Alguns exemplos são: queda de performance conforme o aumento da carga de trabalho, tempo necessário para o sistema atingir um patamar próximo ao ponto de falha e excesso de provisionamento.

3.1.3 Análise quantitativa

A análise quantitativa será realizada sobre os resultados dos *benchmarks*. Esse passo será baseado nos princípios quantitativos listados na sessão Princípios

Quantitativos. O objetivo desse passo é transformar os resultados dos *benchmarks* em conhecimento, por meio de tabelas e representações gráficas. Esses conhecimentos serão essenciais para a proposta da estratégia, pois é neles que estarão os comportamentos e limitações de cada técnica de escalabilidade, dado um determinado tipo de carga de trabalho.

Na análise quantitativa, poderá ocorrer a situação da geração de resultados não conclusivos sobre a análise dos resultados dos *benchmarks*, nesse caso será necessário realizar adequações nos parâmetros do *benchmark*, ou até melhorias na coleta dos resultados de execução, e realizar uma nova análise. Conforme cita Schwartz et al. (2012, p. 47), geralmente um *benchmark* deve ser executado diversas vezes, de modo a convergir para um padrão, onde pode-se utilizar métodos estatísticos para realizar a média de todos os resultados e obter os intervalos de confiança.

3.1.4 Proposta de estratégia

A atividade de proposta de estratégia possui como objetivo propor um fluxo de atividades que orientará sobre qual a estratégia de escalabilidade é mais adequada, de acordo com as características de uma determinada aplicação. Essa estratégia poderá ser aplicada em *softwares* de diversas áreas, mas englobará somente a análise de escalabilidade do escopo do SGBD.

3.1.5 Validação

A proposta será validada com base em estudos de casos de cargas de trabalho de aplicações reais e de trabalhos relacionados. Nessa validação será analisado a possibilidade de utilizar a estratégia proposta para definir uma técnica de escalabilidade, conforme a necessidade de cada aplicação.

Serão apresentados dois estudos de casos aplicando a estratégia proposta, visando definir a técnica mais adequada para cada aplicação. A validação será realizada por meio da aplicação de procedimentos para certificar se os resultados da

estratégia apresentam coerência, em relação aos cálculos de análise de escalabilidade que serão aplicados.

3.2 PRINCÍPIOS QUANTITATIVOS

Para a orientação da análise quantitativa é essencial definir previamente quais serão os princípios quantitativos utilizados na análise dos resultados da execução dos *benchmarks*. Para propor um modelo de avaliação das estratégias de escalabilidade, os princípios abordados serão paralelismo, replicação de dados, desempenho de *hardware* e capacidade do sistema. Para a análise quantitativa serão utilizadas as seguintes unidades de medidas: *throughput*, tempo de resposta, concorrência e escalabilidade.

O *throughput*, refere-se ao número de transações por unidade de tempo. Essa unidade é amplamente utilizada no mercado de SGBDs para comparar o desempenho de diferentes tecnologias. A unidade de medida mais comum é transações por segundo (TPS) ou transações por minuto (TPM).

No tempo de resposta é medido o tempo total que uma tarefa leva para ser executada. As unidades podem variar nesse tipo de métrica, mas são sempre relacionadas com o tempo, normalmente são apresentadas em milissegundos (ms), segundos (s) ou minutos (min).

A concorrência descreve a quantidade de conexões de clientes, na base de dados, que estão realizando tarefas simultaneamente de leitura e escrita. Diferente das outras métricas já citadas, a concorrência não se refere a um atributo de saída da execução de um *benchmark*, mas sim de uma propriedade definida na própria ferramenta, a qual se encarregará de atingir esse requisito no momento da execução. Nesse caso, em vez de medir a concorrência, deve-se utilizá-la para medir a performance do SGBD, em diferentes níveis de concorrência.

Por fim, a escalabilidade é a medida que define a capacidade do sistema em lidar com carga de trabalho durante um período, mantendo o tempo de resposta aceitável. Essa medida pode evidenciar as fraquezas de cada estratégia de escalabilidade e também as limitações conforme o crescimento da carga de trabalho.

3.3 FERRAMENTAS DE *BENCHMARK*

Para a execução dos *benchmarks* foram selecionadas as ferramentas *sysbench* e *Percona's TPCC-MySQL Tool*. Essas ferramentas são do tipo *single-component*, a escolha desse tipo de ferramenta deve-se ao fato de que a análise das estratégias será realizada somente no contexto de serviço do SGBD. Utilizou-se como base, para a escolha das ferramentas, as características de flexibilidade e implementação de testes do padrão TPC-C.

A ferramenta *sysbench* foi desenvolvida para testar não somente a performance do banco de dados, mas também para verificar o desempenho do *hardware* em atender a demanda do SGBD. Além disso, essa ferramenta é flexível permitindo a criação de execuções específicas por meio de *scripts* da linguagem de programação *Lua* (SCHWARTZ et al., 2012, p. 56).

O *sysbench* possui *benchmarks* prontos para CPU, E/S, cargas de trabalho OLTP (*Online Transaction Processing*[19]) simples, performance de *threads*, *mutex*[20] e escrita sequencial. Dessas possibilidades as mais interessantes para utilização no trabalho são o de carga de trabalho OLTP simples e o de E/S. Com a execução do *benchmark* da carga de trabalho OLTP é possível verificar qual será o comportamento de cada estratégia de escalabilidade sob processamento transacional concorrente. Já o *benchmark* de E/S pode ser executado em conjunto com o OLTP para simular aplicações com alta carga de E/S em determinados períodos de tempos. Um ponto importante do *benchmark* de E/S é que ele emula como o *InnoDB,* mecanismo de armazenamento do MySQL*,* utiliza o disco.

A Figura 12, apresenta um exemplo de inicialização de um *benchmark* OLTP com a ferramenta *sysbench*, onde uma tabela nomeada como *sbtest* é criada, na base de dados, com um milhão de registros. Nesse primeiro passo é realizado somente a preparação da tabela para a execução.

19 Cargas de trabalho OLTP seguem um perfil de aplicações como da área bancária, de comércio eletrônico ou de reservas de hotel. A principal característica desse perfil é a alta demanda em um ambiente concorrente.

20 Mecanismo de sincronização utilizado em programação concorrente, que bloqueia o acesso simultâneo a um recurso compartilhado entre diversas *threads* ou processos.

Figura 12 – Preparação *benchmark* com *sysbench*

```
$ sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test/ --mysql-user=root prepare
sysbench v0.4.8: multithreaded system evaluation benchmark

No DB drivers specified, using mysql
Creating table 'sbtest'...
Creating 1000000 records in table 'sbtest'...
```

Fonte: SCHWARTZ et al. (2012).

A Figura 13 apresenta o resultado da execução do *benchmark*. Os principais pontos, indicados com uma seta na figura, a serem considerados são: a quantidade de transações, a taxa de transações por segundo, as estatísticas de tempo, e como se comportou o escalonador de *threads* durante a execução.

Figura 13 – Resultado *benchmark* OLTP do *sysbench*

```
$ sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test --mysql-user=root/--max-time=60 --oltp-read-
only=on --max-requests=0 --num-threads=8 run
sysbench v0.4.8: multithreaded system evaluation benchmark

No DB drivers specified, using mysql
WARNING: Preparing of "BEGIN" is unsupported, using emulation
(last message repeated 7 times)
Running the test with following options:
Number of threads: 8

Doing OLTP test.
Running mixed OLTP test
Doing read-only test
Using Special distribution (12 iterations, 1 pct of values are returned in 75 pct cases)
Using "BEGIN" for starting transactions
Using auto_inc on the id column
Threads started!
Time limit exceeded, exiting...
(last message repeated 7 times)
Done.
OLTP test statistics:
        queries performed:
            read: 179606
            write: 0                          <-
            other: 25658
            total: 205264
        transactions: 12829 (213.07 per sec.)
        deadlocks: 0 (0.00 per sec.)
        read/write requests: 179606 (2982.92 per sec.)
        other operations: 25658 (426.13 per sec.)
Test execution summary:
        total time: 60.2114s
        total number of events: 12829
        total time taken by event execution: 480.2086   <-
        per-request statistics:
            min: 0.0030s
            avg: 0.0374s
            max: 1.9106s
            approx. 95 percentile: 0.1163s
Threads fairness:
        events (avg/stddev): 1603.6250/70.66      <-
        execution time (avg/stddev): 60.0261/0.06
```

Fonte: SCHWARTZ et al. (2012).

Por outro lado, a ferramenta *Percona's TPCC-MySQL Tool* implementa o padrão de *benchmark* TPC-C do conselho TPC (*Transaction Processing Performance Council*), com uma carga de trabalho OLTP mais complexa. O TCP é uma corporação sem fins lucrativos fundada para definir o processamento de transações e *benchmarks* de banco de dados, com a ideia de disseminar dados de desempenho objetivos e verificáveis.

O *benchmark* TPC-C é baseado em cargas de trabalho do tipo OLTP, sendo uma mistura de transações de leitura e de transações de alteração de dados intensivas, que simulam atividades comuns encontradas em ambientes de aplicações OLTP complexas. Isso é realizado por meio de execução de componentes característicos desse tipo de sistema, caracterizados por: execução simultânea de múltiplos tipos de transações, execução de transação online e deferidas, múltiplas sessões, tempo moderado de execução e retorno para a aplicação, E/S de disco significante, integridade de transações, distribuição não uniforme de acesso aos dados através de chaves primárias e secundárias, bases de dados com tabelas não uniformes, contenção ao acesso de dados e atualização (TPC, 2010, p. 7).

A execução do *benchmark* fornecido pela ferramenta *Percona's TPCC-MySQL Tool* é similar ao apresentado na ferramenta *sysbench,* primeiro deve ser executado um comando para iniciar a base de dados e em seguida executar o teste. A grande diferença está na complexidade da base de dados criada para a execução do *benchmark*, ela simula um ambiente de indústria onde são gerenciados vendas de produtos. O resultado da execução traz um histograma dos tempos de respostas durante a execução do *benchmark*, além das informações básicas que retornam no *sysbench*.

3.4 CONSIDERAÇÕES FINAIS

Neste capítulo foi abordado a proposta de solução para o problema de pesquisa. Essa proposta será utilizada para orientar os próximos passos do trabalho que envolve a aplicação dos *benchmarks*, análise quantitativa, proposta de estratégia e validação.

4 ANÁLISE DE ESCALABILIDADE

O princípio da análise de escalabilidade está na identificação do comportamento de cada estratégia conforme o aumento de carga de trabalho do sistema. Para simular essa situação foram realizadas séries de testes, sobre a mesma estratégia, aumentando gradualmente os parâmetros de volume de dados e nível de concorrência.

A observação dos comportamentos do sistema focam nas propriedades de *throughput* e da sua capacidade de escalabilidade. Esses comportamentos, trazem uma visão geral de como o sistema está operando podendo auxiliar na conclusão de algumas questões práticas como: o tempo que o sistema será capaz de operar dentro do esperado, considerando a taxa de crescimento de carga de uma aplicação; quantos servidores são necessários para um período de carga de trabalho menor; o sistema está próximo a um ponto de falha; a infraestrutura atual está superdimensionada.

Os *benchmarks* possuem um padrão de carga de trabalho OLTP. Esse padrão engloba aplicações que são orientadas a transação, como: sistemas ERPs (*Enterprise Resource Planning*), e-commerce, sistemas financeiros e sistemas de reserva.

As atividades desenvolvidas nessa etapa são: configuração dos ambientes, execução dos *benchmarks* e análise quantitativa. Para configuração dos ambientes e execução dos *benchmarks* foi alocado recursos de *hardware* na AWS (*Amazon Web Services*), que fornece serviços de computação na nuvem.

4.1 CONFIGURAÇÃO DOS AMBIENTES

Para a configuração dos ambientes utilizou-se a ferramenta Kubernetes[21], que funciona como um orquestrador de container para implantação, dimensionamento e gerenciamento de sistemas distribuídos. Essa ferramenta foi inicialmente instalada e configurada em três instâncias básicas da AWS, com baixo recurso computacional para otimização de custos. Possuindo essa configuração funcional nas instâncias

21 https://kubernetes.io

básicas, é fácil ampliar e alocar novos recursos de *hardware* para as etapas de preparação e execução. As especificações das instâncias básicas são apresentadas no Quadro 3.

Quadro 3 – Especificação instâncias configuração

Tipo da instância	Memória	vCPUs	SO
t3a.micro	1GB	2 (AMD EPYC 7571 @ 2.5GHz)	Amazon Linux 2
t2.micro	1GB	1 (Intel® Xeon® CPU E5-2676 v3 2.40GHz)	Amazon Linux 2
t2.micro	1GB	1 (Intel® Xeon® CPU E5-2676 v3 2.40GHz)	Amazon Linux 2

Fonte: Autor (2019)

Cada instância funciona como um nodo que compõem o ambiente distribuído, onde a configuração do que será executado em cada um é realizado diretamente no Kubernetes. O Kubernetes depende de uma solução de virtualização no nível de sistema operacional, instalada nos nodos, para que possa executar os containers. Para isso foi selecionado a ferramenta Docker[22].

O primeiro passo da configuração é a criação das imagens Docker, com as respectivas ferramentas e configurações iniciais. Para os ambientes de replicação, particionamento funcional e *data sharding* é utilizada uma imagem do SGBD MySQL 8, conforme a Figura 14. Para o ambiente de *clustering* é utilizada uma imagem do MySQL *Cluster* 8, conforme a Figura 15. As imagens utilizadas são herdadas das imagens fornecidas pela Oracle, com uma configuração adicional que ajusta o fuso horário para o horário de São Paulo.

Figura 14 – *Dockerfile* do ambiente MySQL 8

```
FROM mysql:8

RUN ln -sf /usr/share/zoneinfo/America/Sao_Paulo /etc/localtime
```

Fonte: Autor (2019).

Figura 15 – *Dockerfile* do ambiente MySQL *Cluster* 8

```
FROM mysql/mysql-cluster:8.0

RUN ln -sf /usr/share/zoneinfo/America/Sao_Paulo /etc/localtime
```

Fonte: Autor (2019).

As imagens Docker são definidas em um arquivo chamado *Dockerfile*, esse arquivo possui instruções que serão utilizadas para a criação de um container

22 https://www.docker.com/

completo e executável, dentro do Docker. Os arquivos de imagens Docker são compostos por múltiplas camadas que fazem toda a parte de configuração e inclusão das bibliotecas de sistema e outras dependências necessárias para a criação de um container.

Com as imagens Docker prontas, deu-se inicio a configuração e instalação das dependências nas instâncias iniciadas na Amazon. A Figura 16, apresenta a lista de comandos executados para a instalação em todas as instâncias. Os principais pontos da instalação são: adição dos repositórios adicionais que fornecem os pacotes dos componentes do Kubernetes e Docker; desativação do SWAP[23], pré-requisito para executar os serviços do Kubernetes; e instalação dos principais componentes, sendo o *Docker CE*, *kubelet*, *kubeadm* e *kubectl*.

Figura 16 – Comandos para instalação de ferramentas nas instâncias

```
$ yum update
$ swapoff -a
$ amazon-linux-extras install docker
$ usermod -a -G docker ec2-user
$ cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/rpm-
package-key.gpg
EOF
$ yum install -y kubelet kubeadm kubectl yum-utils device-mapper-persistent-data lvm2 tc
$ systemctl enable --now kubelet
$ mkdir /etc/docker
$ cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2",
  "storage-opts": [
    "overlay2.override_kernel_check=true"
  ]
}
EOF
$ mkdir -p /etc/systemd/system/docker.service.d
$ systemctl daemon-reload
$ systemctl restart docker
```

Fonte: Autor (2019).

23 SWAP é uma memória virtual, que estende o espaço da memória principal, por meio do armazenamento em disco. O Kubernetes foi projetado para ambientes distribuídos, sendo necessário que os recursos de *hardware* possuam previsibilidade e consistência. Com o SWAP habilitado isso não é possível, pois o tempo de acesso dos dados na memória virtual pode ser maior caso o SWAP seja utilizado.

Cada componente instalado possui uma responsabilidade diferente na execução e gerenciamento dos ambientes. É importante saber qual é o papel de cada componente na execução, pois os próximos passos da configuração envolvem a execução de comandos sobre os mesmos. Dessa forma, os componentes e suas principais características são:

- *Docker*: responsável por executar os containers virtualizados sobre o SO. A execução é comandada por meio do Kubernetes.
- *Kubelet*: *daemon*[24] que é executado em cada nodo do *cluster,* sendo responsável pela comunicação.
- *Kubeadm*: é o serviço executado somente nos nodos administradores, que serve para o gerenciamento geral do *cluster*.
- *Kubectl*: API (*Application Programming Interface*) que fornece os comandos necessários para o gerenciamento, verificação e execução dos serviços disponibilizados pelo Kubernetes.

Para finalizar a configuração do ambiente distribuído por meio do Kubernetes e Docker, deve-se executar os comandos da Figura 17, no servidor *master*. Esses comandos são responsáveis por iniciar o ambiente distribuído e por definir as configurações de chaves de acesso e de comunicação, essenciais para que os nodos *slaves* se conectem no ambiente.

Figura 17 – Comandos para inicialização do *master* do ambiente distribuído

```
$ kubeadm init --apiserver-advertise-address $(hostname -i)
$ mkdir -p $HOME/.kube
$ cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
$ chown $(id -u):$(id -g) $HOME/.kube/config
$ kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 | tr -d '\n')"
```

Fonte: Autor (2019).

Já nos servidores *slaves*, foi executado o comando da Figura 18, que indica para o servidor *master* a intenção do *slave* de ingressar no ambiente distribuído. Se a comunicação e as chaves de acesso estiverem corretas, esse servidor será adicionado ao ambiente.

24 Programa que executa em plano de fundo.

Figura 18 – Comandos para inicialização dos *slaves* do ambiente distribuído

```
$ kubeadm join 172.31.13.30:6443 --token c3kycw.8dp31f66cn64qmzt     --discovery-token-ca-cert-hash
sha256:1b25aa7feea43a4e9487b1c741eb8db025c6680551c3d92c2cf14be7ca1396bc
```

Fonte: Autor (2019).

Por fim, pode-se verificar a situação atual do ambiente executando o comando **kubectl get nodes**. O resultado esperado deve ser a lista dos servidores *master* e *slaves* que compõem o ambiente distribuído, na situação ***Ready*** indicando que o ambiente está pronto para ser utilizado, conforme a Figura 19.

Figura 19 – Situação do ambiente distribuído configurado

```
NAME                          STATUS   ROLES    AGE   VERSION
ip-172-31-1-73.ec2.internal   Ready    <none>   8d    v1.16.0
ip-172-31-13-30.ec2.internal  Ready    master   8d    v1.16.0
ip-172-31-3-234.ec2.internal  Ready    <none>   8d    v1.16.0
```

Fonte: Autor (2019).

Após a instalação e configuração das ferramentas, foi realizada a configuração de execução dos serviços do MySQL sobre o ambiente distribuído. Para isso, foi necessário a criação dos arquivos de configuração do Kubernetes, dos serviços do MySQL e criação dos *scripts* que necessitam ser executados na primeira inicialização dos ambientes.

O Kubernetes possui diferentes categorias de serviços que podem ser configurados, de acordo com a necessidade de cada ambiente. Para o trabalho foram utilizadas os tipos: *deployment*, *pod* e *service*.

O tipo *deployment* foi utilizado para as estratégias que são mais flexíveis quanto a configurações de rede e dos serviços que executam, sendo a replicação, particionamento funcional e *data sharding*. O Apêndice A demonstra um exemplo de configuração de *deployment* do ambiente de replicação. O arquivo **mysql-master-deployment.yaml** descreve todas as configurações de execução do container, informa quais são os arquivos de configuração do MySQL que devem ser utilizados, e define quais *scripts* devem ser executados na primeira inicialização do ambiente. Nesse caso é executado o *script* **init_master.sql** e **init_slave.sql,** que estão no Apêndice B, para a configuração inicial da replicação no MySQL.

Para o ambiente de *cluster* foi utilizado o tipo *pod*, pois ele permite a configuração mais específica da execução de cada serviço. Se fez necessário utilizar esse tipo de configuração, pois no caso do MySQL *Cluster* cada *pod* deve executar um serviço diferente, sendo: gerenciamento (*ndb_mgmd*), dados (*ndbd*) e consulta (*mysqld*). O Apêndice C apresenta as configurações desse ambiente.

Por fim, há os arquivos de configuração do tipo *service* que definem como os serviços serão expostos na rede ao mundo externo. Sem esse tipo de configuração, os serviços só se comunicam na rede interna gerenciada pelo Kubernetes. Dessa forma, as configurações do tipo *service* estão presentes para todos os serviços do MySQL que serão utilizados para a execução dos *benchmarks.* Nesses arquivos estão definidos quais as portas dos *pods* que os serviço utilizam e qual porta deve ser utilizada no *host* para expor o serviço.

No Apêndice D são listados os arquivos de configurações utilizados no MySQL. Cada ambiente utiliza os arquivos de acordo com a estratégia que está sendo configurada, isso é definido nos arquivos de *deployment* ou *pods* do Kubernetes.

4.2 *BENCHMARKS*

Os *benchmarks* foram executados com as ferramentas *sysbench* e *Percona's TPCC-MySQL Tool*, como citado na sessão Ferramentas de *Benchmark*. Da ferramenta *sysbench* foi selecionado o teste **oltp_read_only**, que possui uma execução baseada somente na leitura de dados. A execução é composta de três etapas essenciais: preparação, execução e obtenção dos resultados.

Na etapa de preparação é realizada a criação das bases de dados e de suas estruturas, acompanhado da inserção de um lote de dados iniciais. O volume inicial é configurado por meio de parâmetro das ferramentas de *benchmark.*

A execução é realizada após a etapa de preparação finalizar. Nessa etapa, os comandos programados nos *benchmarks* são executados sobre os ambientes, podendo ser configurado os parâmetros de concorrência e volume de dados adicionais, que simulam um ambiente de execução real de uma aplicação.

Por fim, tem-se a etapa de obtenção dos resultados. Cada ferramenta representa o resultado da execução em um formato diferente. Sendo assim, se faz necessário a normalização dos resultados para um único formato para que possa ser utilizado na análise quantitativa.

4.2.1 Planejamento

Antes de dar início as principais etapas do *benchmark*, foi desenvolvido um plano de execução que possui como objetivo demonstrar o comportamento de aumento na escala de volume de dados e concorrência. Dessa forma, é possível verificar a escalabilidade de cada ambiente testado.

Primeiramente foi definido os tipos de instâncias utilizadas para os *benchmarks.* Optou-se por instâncias do mesmo tipo para os SGBDs e um tipo diferente para a execução dos *benchmarks*, conforme o Quadro 4. O objetivo é possuir mais vCPUs na instância que executa os testes para obter um melhor resultado nas simulações com maior concorrência. Todas as instâncias foram alocadas na mesma região para que a latência de rede fosse a menor possível. A quantidade de instâncias alocadas varia de acordo com a simulação de cada ambiente.

Quadro 4 – Especificação instâncias *benchmark*

Utilização	**Tipo**	**Memória**	**vCPUs**	**SO**	**InnoDB *Buffer***
Benchmark	c4.2xlarge	15GB	8 (Intel Xeon E5-2670 v2 2.50GHz)	Amazon Linux 2	-
SGBD	m3.large	7.5GB	2 (Intel Xeon E5-2670 v2 2.50GHz)	Amazon Linux 2	5GB

Fonte: Autor(2019).

Com os tipos de instâncias definidos, iniciou-se o planejamento do volume de dados e concorrência. Para cada volume é executado uma sequência de testes aumentando gradualmente a quantidade de conexões, como indicado no Quadro 5. Assim, além de testar a escala no aumento de volume de dados, é testada a escala no aumento da concorrência. As escalas de volume de dados foram definidas baseando-se na memória disponível para execução do MySQL (InnoDB *Buffer*). Dessa forma, foi definido que um teste deveria ser realizado com volume de dados

inferior ao total de memória (1GB), outro com o volume no limite disponível de memória (5GB) e por fim com o dobro do que há disponível de memória (10GB).

Quadro 5 – Escala de volume de dados e concorrência

	Volume											
	1GB				5GB				10GB			
Concorrência (Nº conexões)	8	16	32	64	8	16	32	64	8	16	32	64

Fonte: Autor(2019).

Além da escala de volume e concorrência, foi planejado um teste específico, com os ambientes de replicação e *cluster*, para validar se a capacidade do sistema aumenta na medida que são adicionados novos nodos de leitura. Esse teste serve para identificar a capacidade de escalar de cada estratégia. Não foi possível realizar esse mesmo teste nos ambientes de particionamento funcional e *data sharding*, pois essas estratégias possuem o objetivo separar fisicamente os dados cada qual com sua interface de comunicação de E/S separadas.

Com o planejamento finalizado, foi possível dar continuidade nas próximas etapas do *benchmark* que envolvem a preparação dos ambientes e execução dos testes com cada ferramenta. Para a execução, foi definido que o tempo máximo de cada teste é de 5 minutos, gerando um relatório de situação de execução a cada 10 segundos. Assim é possível avaliar possíveis casos de instabilidade, durante a execução, que possam afetar o resultado final.

4.2.2 Preparação

Nesta etapa foi realizado a preparação dos ambientes para execução dos *benchmarks*, possuindo como principal objetivo a alocação dos recursos de *hardware*, inicialização dos ambientes via Kubernetes e criação das bases de dados populadas. Os ambientes foram preparados conforme o planejamento descrito anteriormente, sendo o primeiro passo a inicialização e configuração das instâncias na AWS para utilização do Kubernetes.

Com as instâncias preparadas, foi possível iniciar cada ambiente, utilizando os arquivos de configuração que foram gerados na etapa de Configuração dos

Ambientes. Para iniciar é necessário executar o commando *apply* do Kubernetes que aplica as configurações dos ambientes. É possível verificar a situação de execução do ambiente utilizando o comando *get pods*, conforme a Figura 20, que descreve a situação de cada contâneir e se ele está disponível e operacional no ambiente distribuído, após a aplicação das configurações.

Figura 20 – Comandos para iniciar ambiente de replicação

```
$ kubectl apply -f kube/replica/mysql-master-deployment.yaml
deployment.apps/mysql-master-deploy created
$ kubectl apply -f kube/replica/mysql-slave-deployment.yaml
deployment.apps/mysql-slave-deploy created
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
mysql-master-deploy-6df484448-2tztk 1/1     Running   0          42m
mysql-slave-deploy-59f7dc6c7d-vvk2g 1/1     Running   0          41m
```

Fonte: Autor (2019).

Após aplicar as configurações do ambiente, pode-se dar início a fase de criação das estruturas das bases de dados e inserção dos dados para a execução dos *benchmarks*. Cada ferramenta possui sua própria estrutura de tabelas e indexação. A ferramenta *Percona's TPCC-MySQL Tool* implementa os padrões TPC-C, enquanto a ferramenta *sysbench* cria uma estrutura de tabelas hipotéticas.

Os comandos de preparação de ambas as ferramentas possuem parâmetros similares, exceto no atributo que define o volume de dados. No *sysbench*, o volume pode ser controlado pela quantidade de tabelas e pela quantidade de registros em cada tabela, enquanto no *Percona's TPCC-MySQL Tool*, pode ser definido pela quantidade de armazéns. O Quadro 6 apresenta os valores utilizados para cada preparação conforme o volume planejado. Para o *sysbench* foi definido utilizar por padrão 10 tabelas, definido pelo parâmetro *--tables* e variar a quantidade de registros, definido em *–table-size,* onde cada registro ocupa em média 206 *bytes.* Já para o teste TPC-C a cada unidade de armazém são gerados 100MB de dados, sendo definido pelo parâmetro *-w.*

Quadro 6 – Parâmetros para volume de dados das ferramentas de *benchmark*

	Ferramenta	
Volume	*Sysbench*	*Percona's TPCC-MySQL Tool*
1GB	--tables=10 --table-size=1000000	-w 10
5GB	--tables=10 --table-size=2500000	-w 50
10GB	--tables=10 --table-size=5000000	-w 100

Fonte: Autor(2019).

4.2.2.1 Preparação *Percona's TPCC-MySQL Tool*

O modelo de dados TPC-C não tem intenção de representar um segmento de negócio específico. De acordo com sua especificação, o modelo representa qualquer negócio que possui os requisitos de gerenciar, vender ou distribuir um produto ou serviço (TPC, 2010, p. 10). O modelo foi planejado com o objetivo de reduzir a diversidade das operações encontradas em uma aplicação em produção, enquanto mantém as características essenciais de performance da aplicação, como: nível de utilização do sistema e a complexidade das operações.

O Apêndice E, apresenta o modelo relacional das tabelas desta ferramenta. No total são nove tabelas que simulam uma aplicação contendo funcionalidades de gerenciamento de armazéns, produtos, estoques, vendas, clientes e histórico de compras.

Para executar a preparação do ambiente do teste TPC-C é necessário executar o comando *tpcc_load*, conforme exemplo da Figura 21, com os parâmetros para conexão com a base de dados e a quantidade de armazéns. Como resultado é listado a confirmação dos parâmetros utilizados e a situação do processo de preparação dos dados.

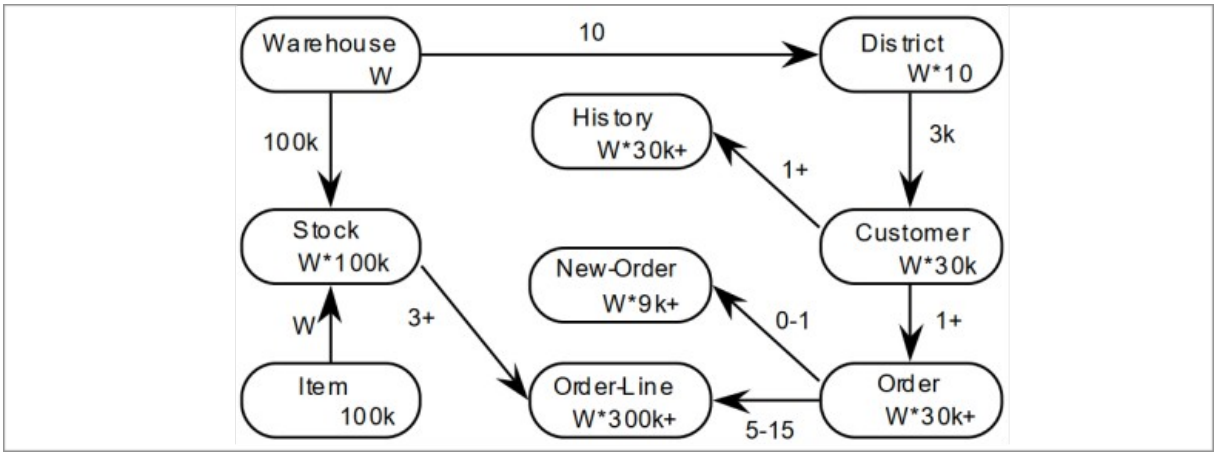
Figura 21 – Comando para preparar o *benchmark* TPC-C

```
$ tpcc_load -h127.0.0.1 -P3036 -dsb -uroot -pbench -w10
************************************
*** TPCC-mysql Data Loader        ***
************************************
option h with value '127.0.0.1'
option P with value '30036'
option d with value 'sb'
option u with value 'root'
option p with value 'bench'
option w with value '10'
<Parameters>
     [server]: 127.0.0.1
     [port]: 30036
     [DBname]: sb
       [user]: root
       [pass]: bench
  [warehouse]: 10
TPCC Data Load Started...
```

Fonte: Autor (2019).

O dimensionamento do volume de dados é controlado pelo número de registros de armazéns informado. A Figura 22 apresenta a relação quantitativa de registros de cada tabela. Os números dos blocos das entidades representam a cardinalidade das tabelas e são múltiplos do número de armazéns (**W**), o símbolo **+** indica que a quantidade de registros pode ter uma pequena variação. Os números das arestas representam a cardinalidade da relação.

Figura 22 – Relação dos registros com o armazém



Fonte: TPC (2010).

4.2.2.2 Preparação *sysbench*

A estrutura de tabelas do *sysbench* é simples e possui o objetivo de estressar ao máximo o SGBD, sem considerar as complexidades adicionais relacionadas a base de dados relacionais. O Apêndice E apresenta um exemplo de estrutura gerada, utilizando como parâmetro a criação de duas tabelas, não há relacionamento entre as mesmas e as colunas representam dados fictícios. Esse é um teste simples que pode ser aplicado em outras tecnologias, como base de dados *NoSQL,* com algumas adaptações. Isso permite realizar comparativos de performance considerando somente as operações básicas de leitura e escrita de cada tecnologia.

Para preparar o ambiente com a ferramenta *sysbench* é necessário executar o comando *sysbench* com o parâmetro *prepare*, conforme a Figura 23. O volume de dados é configurado pelos parâmetros *table-size*, que define a quantidade de registros nas tabelas, e *tables*, que define a quantidade de tabelas criadas. O resultado do comando é a situação da execução da preparação do ambiente.

Figura 23 – Comando para preparar o *benchmark sysbench*

```
$ sysbench –mysql-host=127.0.0.1 --mysql-port=3306 --mysql-user=root --mysql-password=bench \
        --mysql-db=sb  --db-driver="mysql" --threads=4 --tables=10 --table-size=1000000 oltp_read_only prepare
sysbench 1.1.0 (using bundled LuaJIT 2.1.0-beta3)

Initializing worker threads...

Creating table 'sbtest2'...
Creating table 'sbtest1'...
Creating table 'sbtest4'...
Creating table 'sbtest3'...
```

Fonte: Autor (2019).

4.2.3 Execução

Para a execução dos *benchmarks* foi seguindo o planejamento, de modo que após a finalização da preparação do volume de dados no ambiente, iniciou-se a etapa de execução variando a concorrência. Os testes ficaram em execução por um período de cinco minutos, controlados via parâmetro nas ferramentas de *benchmark*. Após a finalização da execução com variação de concorrência, as bases de dados eram excluídas e os SGBD reiniciados, antes de dar início a preparação com um novo volume de dados ou da configuração de um novo ambiente.

A execução dos *benchmarks* foram realizados por meio da instância específica (c4.2xlarge), sendo que a comunicação entre os servidores foi realizada por meio das interfaces de rede interna. Utilizando essa estratégia, além da latência de rede ser menor do que a comunicação externa, é simulado um ambiente de uma aplicação real executada em uma plataforma de infraestrutura na nuvem.

Como cada ambiente possui suas particularidades, foi necessário a utilização de uma ferramenta de *proxy* para que fosse possível fazer o encaminhamento de operações específicas e balanceamento de carga. Para isso foi utilizado o ProxySQL[25], que permite implementar regras de encaminhamento SQL na camada de aplicação. Essas regras também poderiam ser implementadas internamente nas aplicações.

A Figura 24, apresenta um trecho da configuração do ProxySQL utilizada para execução do ambiente de replicação com dois nodos de leitura. Os servidores de leitura pertencem ao mesmo grupo, definido na diretiva **hostgroup** da seção **mysql_servers**, já o servidor de escrita é único no seu grupo. Na seção **mysql_query_rules** é configurado as regras de encaminhamento dos SQLs específicos de cada grupo. As operações de manipulação de dados, consultas com dependência de escrita e gerenciamento de estrutura são encaminhadas para o grupo do servidor de escrita, enquanto todas as outras operações de consultas são encaminhadas para o grupo de servidores de leitura. Esse serviço é executado no servidor de *benchmark*, e recebe conexões na porta **6033**.

25 https://www.proxysql.com/

Figura 24 – Configuração ProxySQL para ambiente de replicação

```
mysql_servers =
(
        { address="172.31.13.162", port=30036, hostgroup=0 },
        { address="172.31.5.69", port=30038, hostgroup=1, weight=1 },
        { address="172.31.13.94", port=30037, hostgroup=1, weight=1 }
)
mysql_query_rules:
(
        {
                rule_id=1
                active=1
                match_pattern="^(CREATE|DROP|ALTER|INSERT|DELETE|UPDATE)"
                destination_hostgroup=0
                apply=1
        },
        {
                rule_id=2
                active=1
                match_pattern="^SELECT .* FOR UPDATE$"
                destination_hostgroup=0
                apply=1
        },
        {
                rule_id=3
                active=1
                match_pattern="^SELECT"
                destination_hostgroup=1
        }
)
```

Fonte: Autor (2019).

Para cada troca de ambiente é necessário atualizar o arquivo de configuração do ProxySQL com os endereços IPs dos servidores que pertencem ao ambiente testado. Após a configuração do ProxySQL é realizado um teste executando uma instrução SQL para retornar o nome do servidor que processou a consulta, conforme a Figura 25. Como resultado, espera-se que cada consulta traga um nome diferente, pois o balanceador de carga distribuirá as consultas de forma equivalente entre os servidores configurados.

Figura 25 – Resultado do teste para validar configuração ProxySQL no ambiente de replicação

```
mysql -u root -pbench -h127.0.0.1 -P 6033 -e "select @@hostname"
+---------------------+
| @@hostname   |
+--------------------- +
| mysql-slave     |
+---------------------- +

mysql -u root -pbench -h127.0.0.1 -P 6033 -e "select @@hostname"
+---------------------+
| @@hostname   |
+---------------------+
| mysql-slave2    |
+---------------------+
```

Fonte: Autor (2019).

A arquitetura de cada ambiente foi configurada utilizando como base o referencial teórico, com a adição do ProxySQL que permite simular uma aplicação que utiliza alguma das estratégias de escalabilidade listada. Cada ambiente possui suas caraterísticas específicas e as estruturas de dados, utilizadas pelas ferramentas, devem se adequar a cada ambiente para que seja possível executar os testes. Nas próximas seções será explicado como cada estratégia foi organizada e quais os detalhes na execução em cada ambiente.

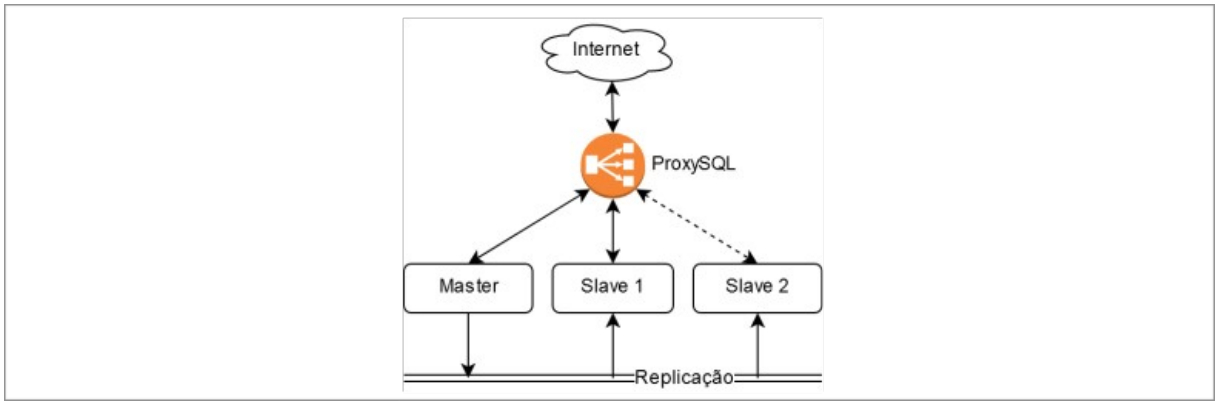
4.2.3.1 Replicação

Para replicação foram realizados duas séries de execuções, uma com a arquitetura utilizando um nodo de leitura, e outra com dois nodos. A ideia foi verificar o aumento da capacidade do sistema em lidar com a mesma carga de trabalho em um ambiente com mais servidores de leitura.

A arquitetura ficou disposta de acordo com a Figura 26, onde as instruções de alterações na estrutura, alterações nos dados, e operações dependentes de leitura para escrita (SELECT ... FOR UPDATE[26]) são direcionadas para o servidor **master**. O restante das consultas são direcionadas para os servidores **slaves**. Na figura o **Slave 2**, possui uma ligação com linha tracejada, pois na primeira série de execução ele não é utilizado, por conta da simulação com apenas um servidor de leitura. Esse servidor é utilizado na segunda série de execução, onde há dois servidores de leitura com a carga balanceada pelo ProxySQL.

26 Realiza o bloqueio contra escrita de todos os registros selecionados até que uma instrução de UPDATE, da mesma transação, seja executada sobre esses registros (MYSQL, 2019).

Figura 26 – Arquitetura do ambiente de replicação



Fonte: Autor (2019).

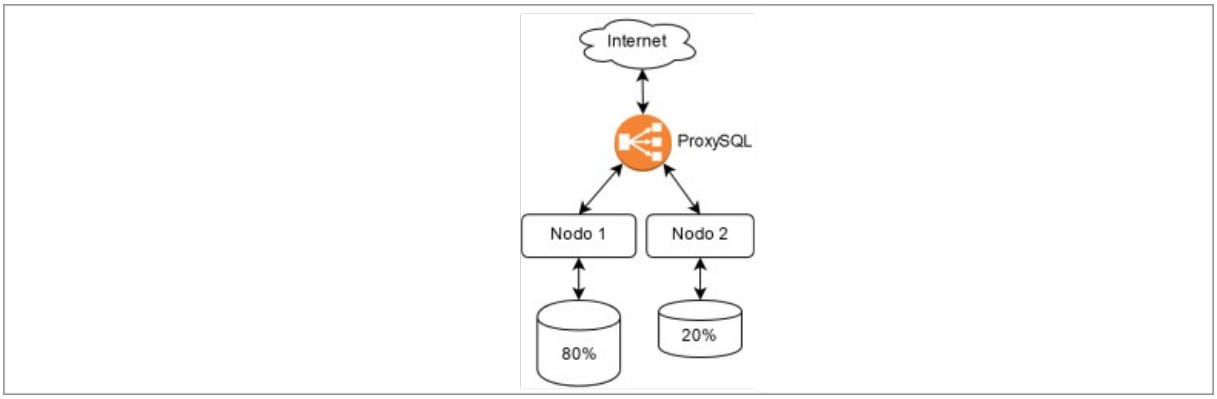
Nessa estratégia de escalabilidade foram executados os *benchmarks* com ambas as ferramentas, pois a estrutura das tabelas e os relacionamentos são adequados para executar no ambiente de replicação.

4.2.3.2 Particionamento Funcional

No particionamento funcional foi realizado somente uma série de execução. Não foram utilizados mais nodos de leitura, pois o objetivo dessa estratégia é realizar a separação da carga de leitura e escrita de uma funcionalidade específica da aplicação. Sendo assim, a intenção do teste foi verificar o desempenho do ambiente simulando a divisão de diferentes cargas de trabalho entre os dois servidores.

A arquitetura ficou disposta de acordo com a Figura 27, de modo que existem dois servidores para leitura e gravação dos dados. No ProxySQL foram configuradas regras de encaminhamento, tanto para consultas como para alterações, para que 80% (8 tabelas) fossem realizadas no **nodo 1** e as outras 20% (2 tabelas) no **nodo 2**. Dessa forma, no momento da execução do *benchmark*, o node 1 receberia um volume maior de carga.

Figura 27 – Arquitetura do ambiente de particionamento funcional



Fonte: Autor (2019).

Para essa estratégia foi executado somente o *benchmark* com a ferramenta *sysbench*, pois a estrutura das tabelas permitem a separação física em servidores distintos. No caso do teste TPC-C, de acordo com a estrutura de dados definida pelo consórcio TPC, não há a possibilidade de divisão física das tabelas, pois elas possuem uma relação forte validada por chaves estrangeiras. Como cita Schwartz et al. (2012, p. 531), o particionamento funcional, considera que cada servidor possua somente os dados de necessidades específicas da aplicação e que normalmente não há a necessidade de junção com os dados dos outros servidores.

Desse modo, para simular uma aplicação executando sobre esse ambiente, foi realizado um teste com *sysbench*, de modo que oito tabelas ficaram em um servidor e as outras duas em outro. Com isso é possível simular um ambiente que há um servidor que recebe mais carga que o outro.

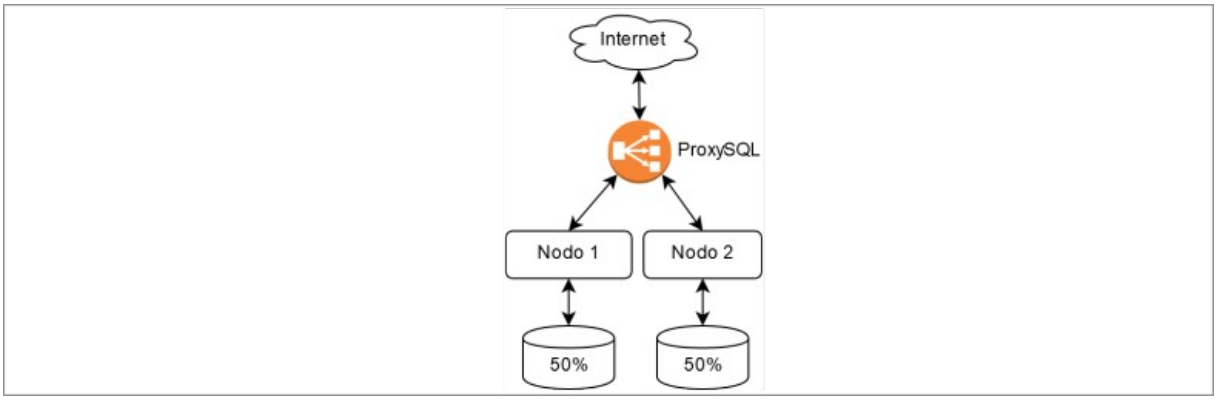
4.2.3.3 *Data Sharding*

Para o ambiente de *data sharding* foi realizado somente uma série de execução, como no particionamento funcional. O objetivo dessa estratégia segue o mesmo conceito base do particionamento funcional, com a diferença de que a divisão dos dados de uma mesma tabela possa ser realizada em blocos menores, distribuídos em diferentes servidores.

A arquitetura ficou disposta de acordo com a Figura 28, similar ao particionamento funcional, com a diferença na distribuição do volume de dados.

Nessa estratégia cada servidor ficou com 50% do volume de dados (5 tabelas), simulando a divisão de carga de uma aplicação.

Figura 28 – Arquitetura do ambiente de *data sharding*



Fonte: Autor (2019).

Para o ambiente de *data sharding* foi executado o *benchmark* com a ferramenta *sysbench*, pelo mesmo motivo da estratégia de particionamento funcional. Dessa forma, para simular o ambiente de *data sharding* foi realizado a divisão balanceada das tabelas criadas pela ferramenta, onde cada servidor ficou com cinco tabelas.

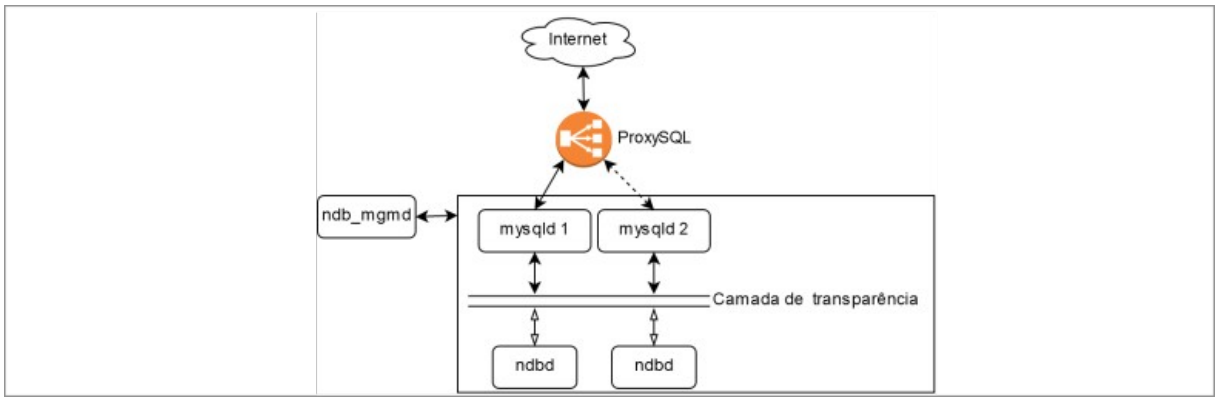
4.2.3.4 *Clustering*

No ambiente de *clustering* foi realizado a execução de duas séries, uma com um e outra com dois nodos de aplicação, seguindo a mesma ideia da replicação. A diferença desse ambiente é que os servidores de aplicação tendem a aumentar a capacidade de conexão com o SGBD, no entanto, a arquitetura depende da performance dos servidores de dados para responder as requisições.

A arquitetura de *clustering* é composta de três diferentes tipos de serviços independentes, como mostra a Figura 29. O serviço **mysqld**, ou serviço de aplicação, é responsável por receber as conexões e instruções SQL e processar as requisições se comunicando com a camada de transparência que é implementada pelo SGBD. Na figura a ligação tracejada entre o ProxySQL e o **mysqld 2**, representa a execução da segunda série, de modo que na primeira não é utilizado o **mysqld 2**. Os serviços **ndbd**, ou serviço de dados, são responsáveis pelo

armazenamento e gerenciamento dos dados, é possível configurar diferentes abordagens de redundância e divisão dos dados. Para o teste foi utilizado uma abordagem simples de divisão dos dados entre os dois nodos, sem redundância. A heurística de divisão é definida pelo SGBD, sendo que o padrão é dividir os dados aplicando uma função sobre o identificador primário dos registros. Por fim, tem-se o serviço **ndb_mgmd**, ou serviço de gerenciamento, que é responsável por controlar toda a estrutura do *cluster* incluindo as configurações do ambiente que deverão ser aplicadas na camada de transparência.

Figura 29 – Arquitetura do ambiente de *clustering*



Fonte: Autor (2019).

Para esse ambiente utilizou-se os *benchmarks* de ambas as ferramentas, pois o SGBD se encarrega de implementar todos os níveis de transparência e realizar o gerenciamento dos dados distribuídos nos diferentes servidores. Sendo assim, ambas estruturas de dados das ferramentas podem ser aplicadas no ambiente.

4.2.4 Resultados

Nesta etapa foram realizadas as análises quantitativas dos resultados das execuções dos *benchmarks*. A performance foi medida em relação a quantidade de transações por segundo (tps). Os gráficos apresentam o comparativo da performance de cada estratégia sob determinado volume de dados para cada ferramenta de *benchmark*. Os resultados quantitativos, utilizados para geração dos gráficos, estão listados no Apêndice F. A primeira relação de gráficos traz os

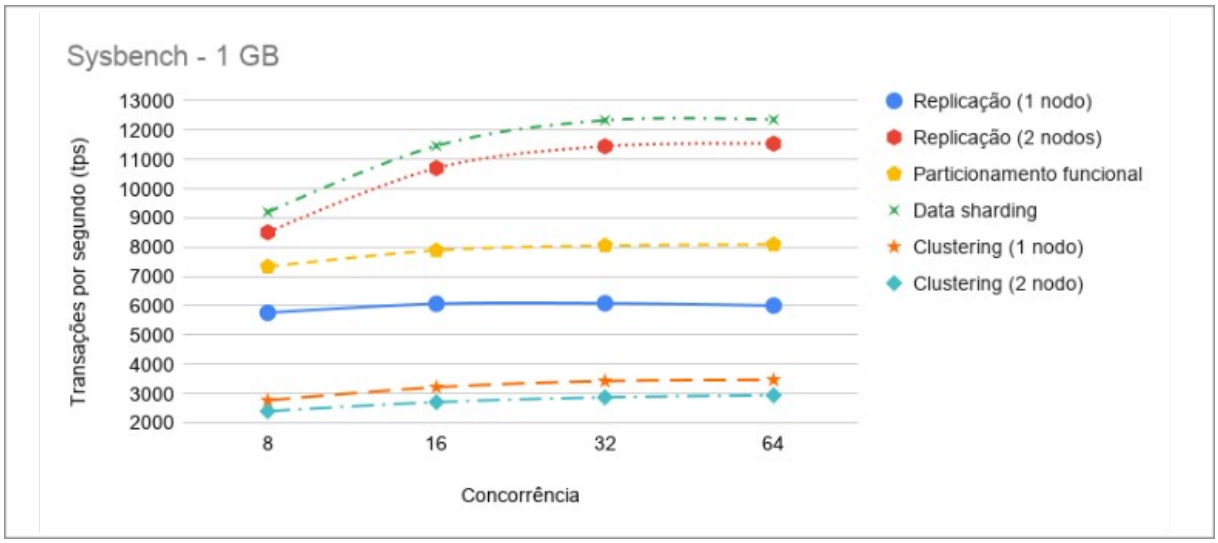
resultado da execução do teste **oltp_read_only**, da ferramenta *sysbench*. O padrão de carga de trabalho desse teste é de somente leitura.

O gráfico da Figura 30, apresenta os resultados da execução do *sysbench* com o volume de dados de 1 GB. Pode-se observar que a estratégia que possui melhor desempenho no teste realizado é o *data sharding*. Isso se explica pelo fato desse ambiente estar configurado para realizar a divisão física dos dados pela metade, isso faz com que o volume de dados seja distribuído o que permite uma melhor indexação das consultas.

Outro ponto relevante é o ganho de capacidade gerado com a adição de um nodo de leitura na estrutura de replicação. A performance se aproximou da estratégia de *data sharding*. No entanto, como na replicação o mesmo volume de dados está presente em todos os servidores, a indexação das consultas é afetada.

A estratégia de *clustering* apresentou um desempenho abaixo das outras. Isso pode ser explicado pelo fato da complexidade de gerenciamento dos registros distribuídos na rede, além da latência de comunicação.

Figura 30 – Execução *sysbench* volume 1GB

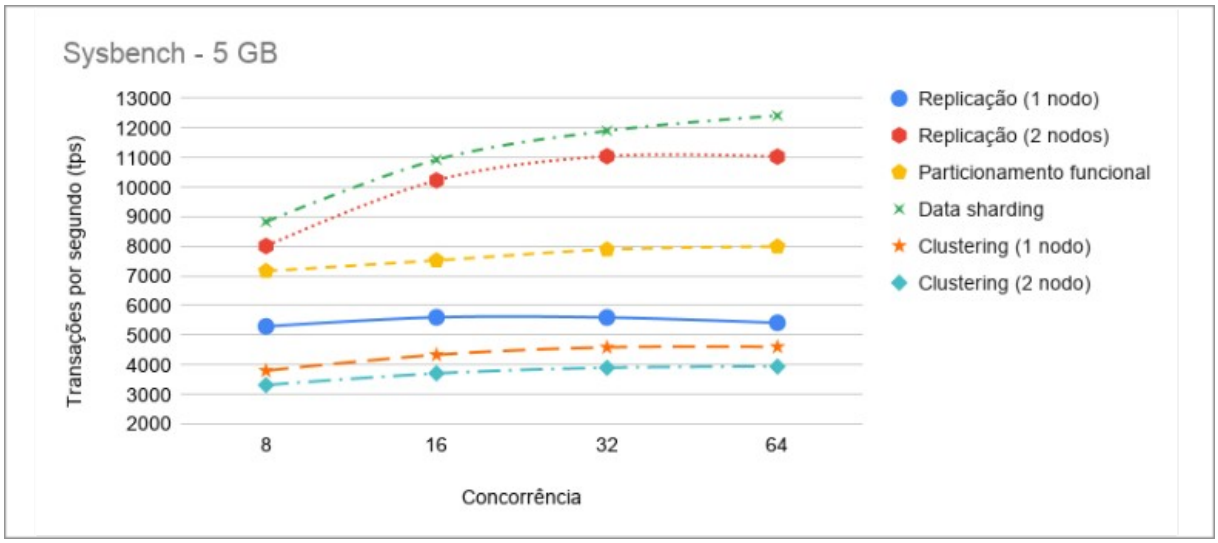


Fonte: Autor (2019).

No gráfico da Figura 31, com o aumento do volume de dados para 5 GB pode-se notar uma leve queda na performance de todas as estratégias exceto na estratégia de *clustering*. De acordo com a documentação do MySQL *Cluster*, a heurística padrão de divisão utiliza uma função *hash*, sobre o identificador primário

dos registros, para distribui-los uniformemente entre os nodos de dados podendo ser aprimorada utilizando um sub identificador (ORACLE, 2016, p. 9). Sendo assim, com um volume pequeno, os dados podem ficar dispersos entre os dois nodos fazendo com que seja necessário uma maior utilização de comunicação de rede para processar as consultas.

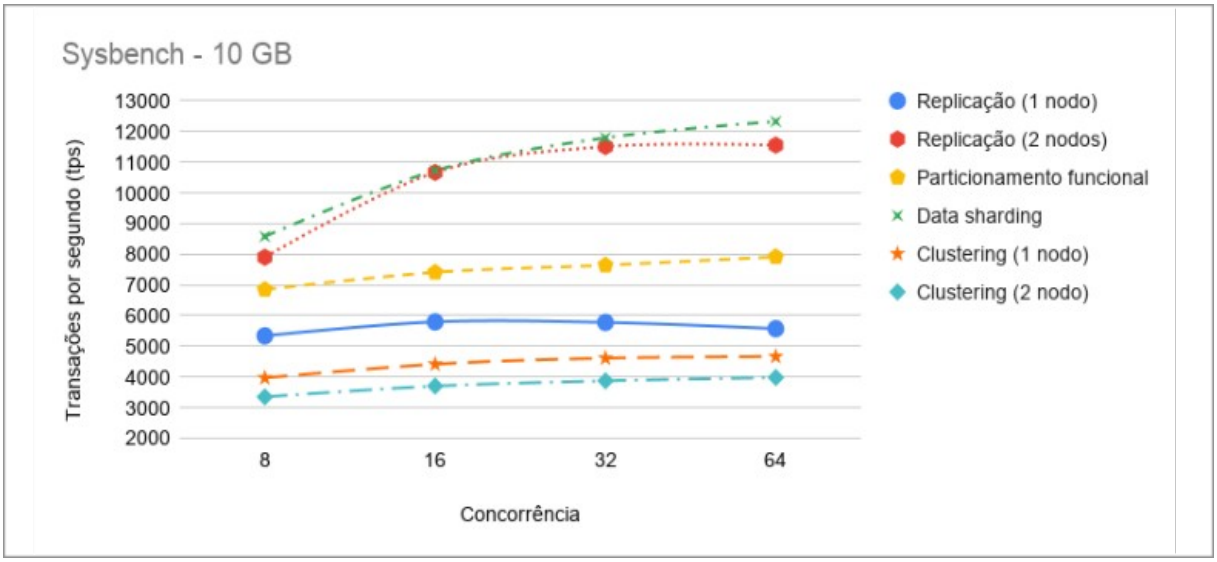
Figura 31 – Execução *sysbench* volume 5GB



Fonte: Autor (2019).

Finalizando a série de execução com volume de 10 GB, utilizando o *sysbench*, os ambientes apresentam uma queda de performance, conforme a Figura 32. Alguns ambientes começam a apresentar um comportamento de parábola na sua função de performance conforme o aumento dos parâmetros de concorrência, seguindo o comportamento proposto na Lei de Escalabilidade Universal. Pode-se observar que com 16 e 32 conexões a performance dos ambientes de replicação, com dois nodos, e *data sharding* se aproximam. Isso ocorre, pois o volume de dados no ambiente de *data sharding* atinge o limite de memória disponível interferindo na indexação das consultas.

Figura 32 – Execução *sysbench* volume 10GB

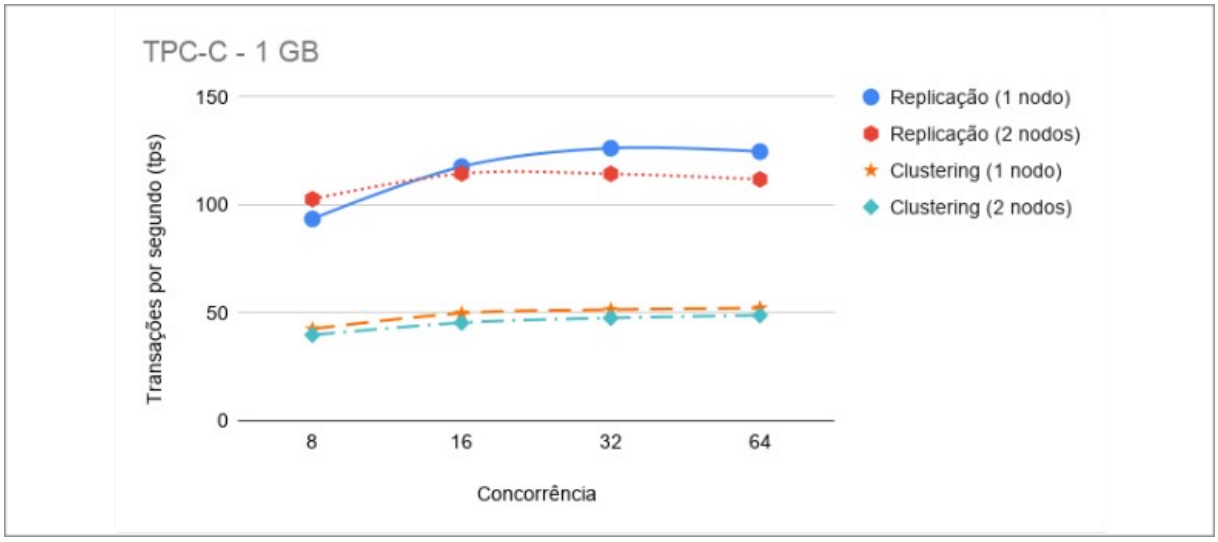


Fonte: Autor (2019).

Na próxima relação de gráficos são apresentados os resultados da execução do teste TPC-C com a ferramenta *Percona's TPCC-MySQL Tool*. O TPC-C implementa um padrão de carga de trabalho mais complexo que envolve leitura, escrita e leitura para escrita.

A primeira série de execução foi realizada com o volume de 1GB. Conforme o gráfico da Figura 33, pode-se verificar que as estratégias utilizando replicação tiveram melhor desempenho comparado a estratégia de *clustering*. Isso se explica pelo fato de que, além da latência de rede, o SGBD deve garantir a consistência das escritas dos dados em todos os nodos do ambiente distribuído. Comparando o comportamento das estratégias de replicação deste teste, com o teste de somente leitura, pode-se observar uma degradação de performance mais acentuada, na replicação com dois nodos, na medida que a concorrência aumenta. Este comportamento é gerado por conta da sobrecarga da replicação em um ambiente que possui carga de trabalho de escrita.

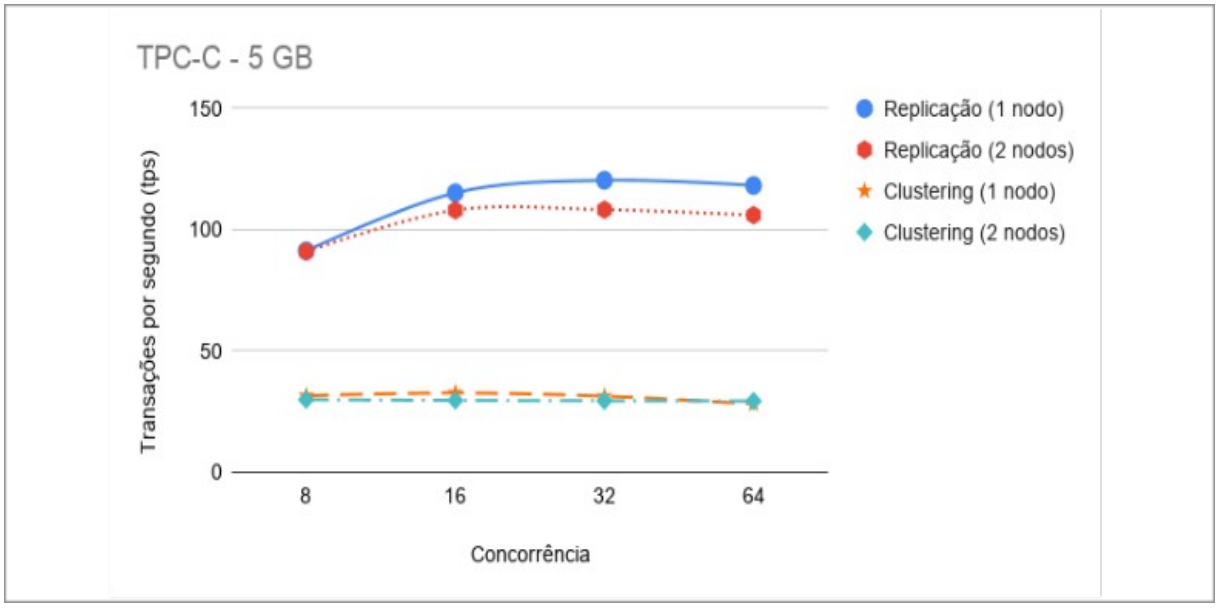
Figura 33 – Execução TPC-C volume 1GB



Fonte: Autor (2019).

Na execução com o volume de 5GB, como mostra a Figura 34, é possível observar uma leve degradação de performance na estratégia de replicação. Desse modo, a replicação com dois nodos iguala seu desempenho com a replicação de um nodo na execução com oito processos, a qual levou vantagem com o volume de 1GB. O ambiente de *clustering* teve uma maior degradação de desempenho relacionado ao ambiente de replicação.

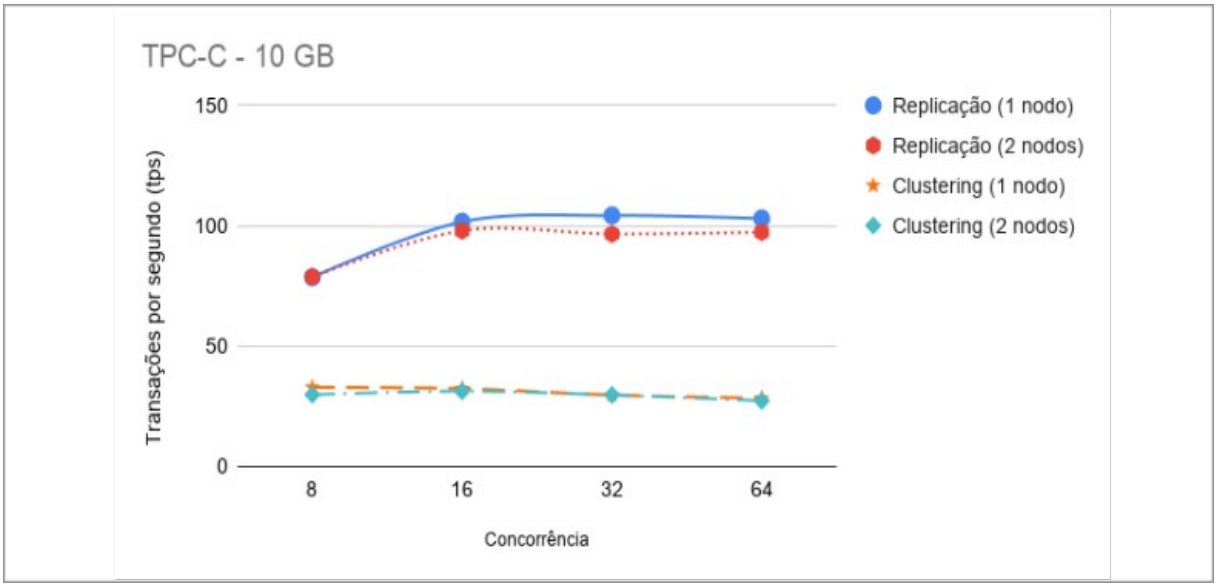
Figura 34 – Execução TPC-C volume 5GB



Fonte: Autor (2019).

Por fim, o resultado da execução do teste TPC-C com o volume de 10GB, conforme a Figura 35, apresenta uma maior degradação no ambiente de replicação. Com o volume de dados maior do que a memória disponível a indexação da base é afetada. O problema é amplificado pelas operações de escrita e sobre a carga adicional necessária para a replicação. O ambiente de *clustering* se comporta de forma mais estável comparado com sua execução em 5GB. Isso se explica pelo fato da distribuição física dos dados em dois nodos permitirem um fator maior de escalabilidade.

Figura 35 – Execução TPC-C volume 10GB



Fonte: Autor (2019).

Com base nos resultados de execução do *sysbench,* utilizando o teste de somente leitura, pode-se concluir que as estratégias que fornecem maior capacidade de processamento para leitura são *data sharding* e replicação com dois nodos. O ambiente de *cluster* possui um desempenho inferior, comparado as outras estratégias, mas aparenta ter um comportamento mais estável conforme o aumento do volume de dados e dos níveis de concorrência. Outro ponto relevante, do ambiente de *clustering*, é que a utilização de dois nodos de aplicação aumentou a concorrência de acesso aos dados fazendo com que a capacidade de leitura do sistema diminuísse.

Na execução dos testes TPC-C, pode-se observar uma queda acentuada na escala de transações por segundo, enquanto o *sysbench* teve sua escala máxima próxima a treze mil, o TPC-C teve a escala próxima a 150. O perfil de carga dos *benchmarks*, fez com que ocorresse essa queda, pois o TPC-C possui o fator de bloqueio das transações por conta das operações de escrita.

No geral, pode-se observar que replicação e *data sharding*, são estratégias que possuem melhor desempenho para escalar operações de leitura. Para operações de escrita a replicação possui melhor desempenho comparado ao ambiente de *clustering*. No entanto, para decidir qual estratégia mais adequada para uma aplicação, o desempenho não é o único fator a ser considerado. Além do desempenho, deve-se considerar outras questões como custo de implementação, capacidade de escalar e a complexidade de manutenção. No próximo capítulo esses fatores serão abordados na proposta de estratégia, que visa ponderar essas questões em uma orientação para a seleção da estratégia e evolução da arquitetura.

5 PROPOSTA DE ESTRATÉGIA

A partir da execução dos *benchmarks* foi possível identificar algumas características de cada estratégia de escalabilidade que são essenciais para a definição de uma arquitetura adequada para uma aplicação. No entanto, o planejamento de capacidade, do SGBD, não é uma atividade simples e determinística. Cada aplicação possui suas próprias regras de negócio, implementadas de forma específica que não seguem um padrão universal. Desse modo a proposta de estratégia visa definir e orientar a organização de um fluxo de atividades contínuas que possam ser utilizadas como orientação para tomada de decisão.

Para isso, serão utilizados como base os referenciais teóricos a fim de organizar a estratégia de acordo com os diferentes métodos e conceitos relacionados a escalabilidade de sistemas. Dessa forma, será possível utilizar a estratégia para entender, avaliar, comparar e melhorar a escalabilidade de sistemas.

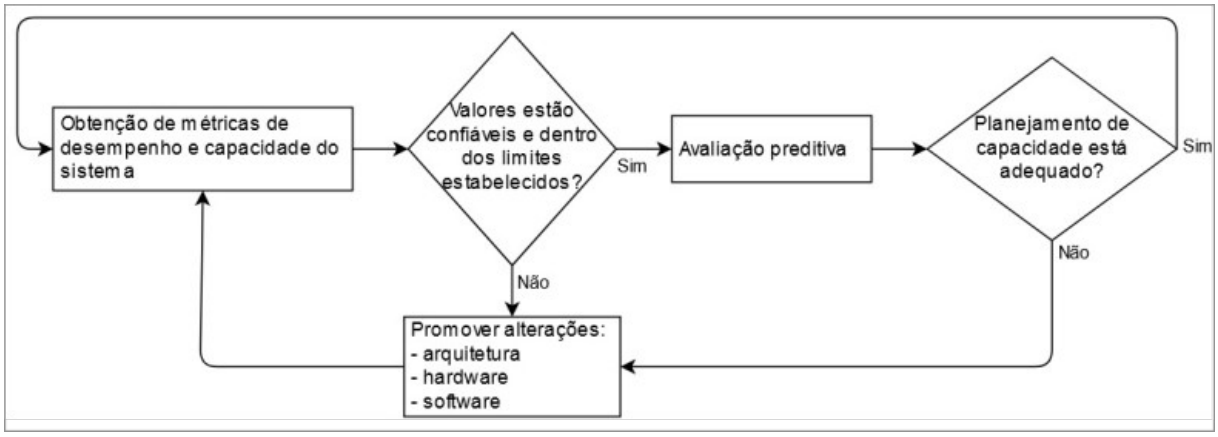
Primeiramente, deve-se definir alguns requisitos de capacidade e carga total por meio de métricas específicas que possam ser mesuradas na arquitetura em execução. Algumas métricas importantes são: tempo de resposta, capacidade consumível e taxa de processamento esperada em momentos de pico.

Com os requisitos definidos é possível avançar para o próximo passo que é responder as seguintes questões, para criar um processo contínuo de planejamento de capacidade (ALLSPAW, 2008, p. 2):

1. Como a infraestrutura atual está funcionando?
2. O que é necessário no futuro para manter um desempenho aceitável do sistema?
3. Como ocorre a instalação e configuração dos recursos após definir o que se quer?
4. Repetir questionamentos periodicamente.

Dentro destas questões, é necessário a tomada de algumas ações para que seja possível respondê-las. O fluxograma da Figura 36 apresenta a estratégia a ser seguida para responder estas questões.

Figura 36 – Fluxograma da estratégia de análise de escalabilidade



Fonte: Autor (2019).

Para a primeira questão é essencial a existência de coletores de métricas de como o sistema se comporta, baseado nos requisitos de capacidade definidos. A configuração de ferramentas automatizadas de monitoramento e alerta, sobre essas métricas, são importantes para notificar os administradores de sistema que algum evento inesperado está acontecendo. No caso do sistema ainda não estar em produção, *benchmarks* específicos são úteis para obter essas informações.

Com as informações coletadas é possível levantar um conjunto base de características importantes sobre a aplicação que deverão ser consideradas no processo. Entre estas características estão: função de crescimento do volume de dados (ex: linear, exponencial), *throughput* e padrão da carga de trabalho, que pode ser obtida pela relação das operações de leitura e escrita.

A segunda questão aborda a predição da capacidade futura buscando determinar quanto recurso será necessário ser alocado futuramente e qual o momento mais adequado para fazer essas alterações. Para isso, pode-se realizar análises quantitativas sobre as informações históricas coletadas sobre o sistema e utilizar a Lei de Escalabilidade Universal, que auxilia a prever o desempenho de um sistema sobre uma carga de trabalho hipotética.

A terceira questão trata do nível de complexidade para ampliar os recursos e fornecer a manutenção do sistema. Cada estratégia possui sua própria característica, de modo que algumas adicionam um nível de complexidade na aplicação, enquanto outras buscam ser mais transparentes de modo que não seja necessário nenhuma intervenção na aplicação.

O Quadro 7, apresenta uma comparação entre as estratégias sobre os aspectos perfil de carga, complexidade de implantação, transparência para aplicação, tipo de replicação, escalabilidade horizontal e número mínimo de servidores. Esses aspectos foram verificados na configuração dos ambientes e na execução dos *benchmarks* na análise de escalabilidade.

Quadro 7 – Comparação de aspectos das estratégias de escalabilidade

	Replicação	**Particionamento funcional**	***Data sharding***	***Clustering***
Perfil de carga	Leitura	Leitura/Escrita	Leitura/Escrita	Leitura/Escrita
Complexidade de implantação	Baixa	Média	Alta	Baixa
Transparência para aplicação	-	-	-	Sim
Tipo de replicação	Assíncrona ou semi síncrona	-	-	Síncrona
Escalabilidade horizontal	Leitura	Leitura e escrita, somente sobre a funcionalidade	Leitura e escrita	Leitura e escrita
Número mínimo de servidores	2	2	2	4

Fonte: Autor (2019).

Por fim a quarta questão se refere a repetir as questões periodicamente com o objetivo de manter a evolução de capacidade do sistema, de acordo com o aumento da demanda, aprimorando o plano de capacidade durante o tempo. Essa ideia de processo contínuo é essencial para acompanhar a evolução da aplicação e da real necessidade de capacidade do SGBD. A periodicidade desse processo pode ser definida pelo administrador do sistema considerando a adição de novas funcionalidades na aplicação, um evento de crescimento de uso inesperado ou então um monitoramento que tenha excedido o limite predefinido.

5.1 VALIDAÇÃO

Para validação da estratégia foram selecionados dois artigos de estudos de casos, com objetivo de aplica-lá verificando se é adaptável as propostas. O primeiro artigo selecionado, *An Improvement in MySQL Cluster in E-commerce Scenarios* (XIANG, 2014), trata sobre uma proposta de arquitetura que pretende satisfazer o teorema CAP em diferentes aspectos. Os autores utilizaram o *benchmark* TPC-W,

definido pelo consórcio TPC, que define uma carga de trabalho de uma loja virtual para venda de livros.

De acordo com a estratégia proposta por este trabalho é necessário, primeiramente, obter as métricas de desempenho e capacidade do sistema, para posteriormente extrair as características da aplicação. O artigo cita que o foco principal está nas operações de busca, de modo que o desempenho das operações de leituras são mais relevantes do que as de escritas. Dessa forma, pode-se extrair as seguintes características da aplicação:

- Crescimento do volume de dados: Linear.
- *Throughput*: N/A.
- Padrão da carga de trabalho: Leitura.

No próximo passo é necessário verificar se a arquitetura atual possui métricas de desempenho confiáveis e dentro dos limites estabelecidos. No artigo os autores não expõem os valores obtidos nos testes, mas informam que a métrica utilizada foi o tempo de resposta das requisições. Nas suas conclusões, informam que a arquitetura proposta gerou gargalos em tabelas que possuem relações *many-to-many*, e que ocorre degradação da performance na medida que o volume de dados aumenta nessas tabelas. Por outro lado, os autores afirmam que a arquitetura proposta possui um bom desempenho nas outras operações e que é fácil de escalar.

Dessa forma, de acordo com a estratégia proposta, há dois caminhos possíveis para essa situação. Como os autores identificaram um gargalo nas relações *many-to-many*, o processo de avaliação preditiva deverá focar nesse ponto fraco, para identificar até que momento essa arquitetura funcionará para determinada aplicação. Assim que os limites definidos forem atingidos será necessário iniciar o processo de alteração de arquitetura. Seguindo o comparativo do Quadro 7, pode-se sugerir a utilização da estratégia de replicação simples, a curto prazo, por conta simplicidade de implantação e por conseguir escalar carga de leitura; e a longo prazo pode-se utiliza uma estratégia de *Clustering*, como indica o artigo.

O segundo artigo selecionado, *Performance Evaluation of MySQL, Cassandra and HBase for Heavy Write Operation* (JOGI et al., 2016), possui o objetivo de

realizar um comparativo entre o desempenho de base de dados NoSQL com o MySQL utilizando um *benchmark* com padrão de carga de escrita. Dessa forma aplicando o primeiro passo da estratégia proposta tem-se as seguintes características:

- Crescimento do volume de dados: Exponencial.
- *Throughput*: 71,9 tps.
- Padrão da carga de trabalho: Escrita.

Entre o comparativo realizado pelos autores os SGBDs, Cassandra e HBase, tiveram uma performance superior na execução do *benchmark.* Com base no comparativo do artigo, é possível identificar que essas outras tecnologias possuem um desempenho melhor do que SGBD relacional tradicional. A estratégia de análise de escalabilidade pode ser utilizada para avaliação de arquiteturas nessas outras tecnologias, no entanto, é necessário que seja realizada uma avaliação prévia da performance dessas técnicas de escalabilidade. Neste trabalho foi realizado uma avaliação das estratégias do SGDB MySQL, desta forma não há como realizar a comparação entre estratégias no estudo de caso específico.

6 CONSIDERAÇÕES FINAIS

O trabalho apresentou uma análise quantitativa sobre técnicas de escalabilidade para base de dados MySQL. O objetivo foi propor uma estratégia para ser utilizada como base para a definição da técnica mais adequada, de acordo com a necessidade de cada aplicação. A partir da análise quantitativa e do referencial teórico foi possível propor a estratégia para planejamento da capacidade de sistemas que utilizam esse SGBD.

Dentro do processo de desenvolvimento da solução foram realizadas as configurações dos ambientes de cada técnica, a execução dos *benchmarks* e a proposta da estratégia. Para isso, foi utilizado o Kubernetes, um orquestrador para ambientes distribuídos, que permitiu uma maior flexibilidade no gerenciamento das configurações e na execução dos *benchmarks*. Para a execução, foi necessário a utilização de uma ferramenta de *proxy*, o ProxySQL, que permitiu simular a execução dos testes nos ambientes distribuídos com distribuição de carga e encaminhamento de operações específicas.

A partir dos resultados apresentados, foi possível extrair alguns aspectos de cada estratégia que posteriormente foram utilizados na elaboração da proposta de estratégia. Essa define orientações para uma avaliação contínua da capacidade do sistema SGBD sob determinada arquitetura.

Os estudos de casos e as avaliações realizadas mostraram que não é possível utilizar um modelo de classificação para a escolha da estratégia de escalabilidade. Isso se deve ao fato de que a análise de capacidade de um sistema é complexo e influenciado diretamente pelas especificidades de negócio implementadas nas aplicações. Por esse motivo, foi elaborado uma estratégia com atividades contínuas, pois assim é possível mensurar e evoluir a arquitetura na medida que surgem novas necessidades por parte da aplicação.

Por fim, conclui-se que o objetivo do trabalho foi atingido parcialmente, pois a validação com estudo de casos sobre situações sintéticas, não comprovam a eficiência da estratégia. Por outro lado, a utilização da mesma, nos estudos de casos, garante uma avaliação inicial das técnicas de escalabilidade disponíveis e permite a avaliação contínua da arquitetura do SGBD.

6.1 TRABALHOS FUTUROS

Para trabalhos futuros, pode-se validar a utilização da estratégia proposta com aplicações que estão em produção. Com isso será possível identificar a eficiência em atingir o objetivo de servir como base para definição da estratégia de escalabilidade mais adequada para as aplicações.

Outra proposta futura, é a ampliação do escopo de avaliação das estratégias de escalabilidade para novas tecnologias e a adição de atributos qualitativos nas avaliações. Como analisado nos trabalhos relacionados, outras tecnologias, como base de dados NoSQL, trazem novas possibilidades e desafios que podem ser avaliados e incrementados nas análises. Assim é possível aumentar as possibilidades da estratégia proposta para que os administradores de sistema possam selecionar a tecnologia e técnica mais adequada a necessidade de cada aplicação.

REFERÊNCIAS

ALLSPAW, John. **The Art Of Capacity Planning**. Sebastopol, CA: O'Reilly Media, 2008.

AMDAHL, Gene M. **Validity of the single processor approach to achieving large scale computing capabilities**. *In*: Proceedings of the April 18-20, 1967, spring joint computer conference (AFIPS '67 (Spring)). ACM, New York, NY, USA, 483-485.

BREWER, Eric. **CAP twelve years later***:* How the "rules" have changed. Computer, vol. 45, no. 2, p. 23-29. Berkeley: University of California, 2012.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson, 2011.

GUNTHER, Neil. **Guerrilla Capacity Planning**: A Tatical Approach to Planning for Highly Scalable Applications and Services. Berlin: Springer-Verlag, 2007.

MYSQL. **MySQL Documentation**. [S. l.: s. n.]: 2019. Disponível em: https://dev.mysql.com/doc/. Acesso em: 13 abr. 2019.

ORACLE. **Guide to scaling web databases with MySQL Cluster**: accelerating innovation on the web and in the cloud. [S. l.: s. n.]: 2016.

ORACLE. **Optimizing MySQL Cluster Performance.** [S. l.: s. n.]: 2016.

ÖZSU, M. Tamer.; VALDURIEZ, Patrick. **Principles of distributed database systems**. 3. ed. Springer Science & Business Media, 2011.

JOGI, Vishal D.; SINHA, Ashay. **Performance evaluation of MySQL, Cassandra and HBase for heavy write operation**. *In:* 2016 3rd International Conference on Recent Advances in Information Technology (RAIT), Dhanbad, 2016, 586-590.

PATTERSON, David A; HENNESSY, John L. **Arquitetura de computadores uma abordagem quantitativa**. 5. ed. Rio de Janeiro: Elsevier, 2014.

SCHWARTZ, Baron; ZAITSEV, Peter; TKACHENKO, Vadim. **High performance MySQL**: optimization, backups, and replication. 3. ed. Sebastopol, CA: O'Reilly Media, 2012.

TPC. **TPC Benchmark C**: Standard Specification. [S. l.: s. n.]: 2010. Disponível em: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf. Acesso em: 8 jun. 2019.

XIANG, Kai. **An improvement in MySQL cluster in e-commerce scenarios**. *In*: Proceedings of 2nd International Conference on Information Technology and Electronic Commerce, Dalian, 2014, 286-289.

APÊNDICE A – CONFIGURAÇÕES DO AMBIENTE DE REPLICAÇÃO PARA KUBERNETES

Arquivo **mysql-master-deployment.yaml**, utilizado para execução do serviço *master* do ambiente de replicação.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: mysql-master
  name: mysql-master-deploy
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql-master
  strategy:
    type: Recreate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: mysql-master
    spec:
      containers:
      - env:
        - name: MYSQL_ROOT_PASSWORD
          value: bench
        image: ambiente_mysql
        imagePullPolicy: IfNotPresent
        name: mysql-master
        ports:
        - containerPort: 3306
        resources: {}
        volumeMounts:
        - mountPath: /etc/mysql/conf.d/
          name: mysql-master-confs
        - mountPath: /docker-entrypoint-initdb.d/
          name: mysql-master-init
      volumes:
        - name: mysql-master-confs
          configMap:
            name: mysql
            items:
            - key: default.cnf
              path: default.cnf
            - key: replica_master.cnf
              path: replica_master.cnf
        - name: mysql-master-init
          configMap:
            name: mysql-init-scripts
            items:
            - key: init_master.sql
              path: init_master.sql
      nodeSelector:
        id: master
```

Arquivo **mysql-master-service.yaml**, utilizado para a configuração da exposição do serviço *master*, do ambiente de replicação, na rede. Nesse caso o serviço é exposto na porta 30036.

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-master
spec:
  selector:
    app: mysql-master
  type: NodePort
  ports:
  - name: "mysql"
    port: 3306
    targetPort: 3306
    NodePort: 30036
```

Além desses dois arquivos há os arquivos **mysql-slave-deployment.yaml** e o **mysql-slave-service.yaml**, que complementam a configuração do ambiente de replicação. Esses dois arquivos seguem o mesmo modelo do arquivo **mysql-master-deployment.yaml** e **mysql-slave-service.yaml**, com diferença nas configurações específicas do *script* de inicialização do ambiente, arquivo de configuração do MySQL e na porta de exposição do serviço, que o caso do *slave* é na porta 30037.

APÊNDICE B – *SCRIPTS* PARA INICIALIZAÇÃO DO AMBIENTE DE REPLICAÇÃO

Os arquivos de *scripts* de inicialização possuem a responsabilidade de inicializar configurações que necessitam ser realizadas pelo cliente SGBD. Para isso o sistema deve estar em execução. Esses *scripts* são executados somente na primeira vez que o ambiente é iniciado.

Arquivo **init_master.sql**, cria um usuário para a função de replicação e habilita as permissões necessárias.

```
CREATE USER 'replica'@'%' IDENTIFIED BY 'bench';
GRANT REPLICATION SLAVE ON *.* TO 'replica'@'%';
```

Arquivo **init_slave.sql**, informa ao SGBD qual será o servidor *master* e qual usuários deverá utilizar para a replicação.

```
CHANGE MASTER TO MASTER_HOST='mysql-master', MASTER_USER='replica', MASTER_PASSWORD='bench';
```

APÊNDICE C – CONFIGURAÇÕES DO AMBIENTE MYSQL *CLUSTER* PARA KUBERNETES

Arquivo **mysql-cluster-mgm-pod.yaml**, utilizado para configuração do gerenciador do *cluster* MySQL.

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: mysql-cluster-mgm
  name: mysql-cluster-mgm
spec:
  containers:
  - env:
    - name: MYSQL_ROOT_PASSWORD
      value: bench
    image: ambiente_mysql_cluster
    imagePullPolicy: IfNotPresent
    name: mysql-cluster-mgm
    command: ["/entrypoint.sh"]
    args: ["ndb_mgmd"]
    ports:
    - containerPort: 1186
    volumeMounts:
    - mountPath: /etc/my.cnf
      subPath: my.cnf
      name: mysql-cluster-mgm-my-confs
    - mountPath: /etc/mysql-cluster.cnf
      subPath: mysql-cluster.cnf
      name: mysql-cluster-mgm-cluster-confs
  volumes:
    - name: mysql-cluster-mgm-my-confs
      configMap:
        name: mysql
        items:
        - key: cluster_my.cnf
          path: my.cnf
    - name: mysql-cluster-mgm-cluster-confs
      configMap:
        name: mysql
        items:
        - key: cluster.cnf
          path: mysql-cluster.cnf
  nodeSelector:
    id: master
```

Arquivo **mysql-cluster-sv1-service.yaml**, utilizado para expor um serviço de consulta de dados do *cluster* na porta 30036.

```
apiVersion: v1
kind: Service
metadata:
  name: mysql-cluster-sv1
spec:
  selector:
    app: mysql-cluster-sv1
  type: NodePort
  ports:
  - name: "mysqld"
    port: 3306
    targetPort: 3306
    nodePort: 30036
```

Arquivo **mysql-cluster-dn1-pod.yaml**, utilizado para configurar um dos serviços de dados do *cluster.*

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: mysql-cluster-dn1
  name: mysql-cluster-dn1
spec:
  containers:
  - env:
    - name: MYSQL_ROOT_PASSWORD
      value: bench
    image: ambiente_mysql_cluster
    imagePullPolicy: IfNotPresent
    name: mysql-cluster-dn1
    command: ["/entrypoint.sh"]
    args: ["ndbd"]
    ports:
    - containerPort: 2202
    volumeMounts:
    - mountPath: /etc/my.cnf
      subPath: my.cnf
      name: mysql-cluster-dn1-my-confs
  volumes:
    - name: mysql-cluster-dn1-my-confs
      configMap:
        name: mysql
        items:
        - key: cluster_my.cnf
          path: my.cnf
  nodeSelector:
    clst: dn1
```

Arquivo **mysql-cluster-sv1-pod.yaml**, utilizado para configurar um dos serviços de consultas do *cluster.*

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    app: mysql-cluster-sv1
  name: mysql-cluster-sv1
spec:
  containers:
  - env:
    - name: MYSQL_ROOT_PASSWORD
      value: bench
    image: ambiente_mysql_cluster
    imagePullPolicy: IfNotPresent
    name: mysql-cluster-sv1
    command: ["/entrypoint.sh"]
    args: ["mysqld"]
    ports:
    - containerPort: 3306
    volumeMounts:
    - mountPath: /etc/my.cnf
      subPath: my.cnf
      name: mysql-cluster-sv1-my-confs
  volumes:
    - name: mysql-cluster-sv1-my-confs
      configMap:
        name: mysql
        items:
        - key: cluster_my.cnf
          path: my.cnf
  nodeSelector:
    clst: sv1
```

APÊNDICE D – CONFIGURAÇÕES MYSQL

Os arquivos de configuração do MySQL definem uma série de parâmetros que alteram a forma de como o SGBD se comporta quando está em execução. Cada ambiente possui uma necessidade de configuração diferente, por esse motivo foram criados arquivos específicos para cada configuração. O único arquivo compartilhado é o **default.cnf** que é utilizado para os ambientes de replicação, particionamento funcional e *data sharding*.

Arquivo **default.cnf**, define as configurações padrões para alguns ambientes. Entre as configurações estão o formato da replicação dos dados, localização do diretório de dados, quantidade máxima de memória utilizada pelo SGBD e definição do fuso horário a ser utilizado nos arquivos de *logs*.

```
[mysqld]

default-authentication-plugin = mysql_native_password
log_timestamps = SYSTEM
datadir = /var/lib/mysql
sync_binlog = 0
binlog_expire_logs_seconds = 3
slow_query_log = 1
long_query_time = 5
binlog_format = ROW

innodb_flush_log_at_trx_commit = 0
innodb_buffer_pool_size = 5G
max_connections = 1000
```

Arquivo **replica_master.cnf**, define o identificador do servidor *master*:

```
[mysqld]
server_id = 1
```

Arquivo **replica_slave.cnf**, define o identificador do servidor *slave.*

```
[mysqld]
server_id = 2
```

Arquivo **cluster.cnf**, define a estrutura do *cluster*, identificando cada serviço e qual o seu endereço na rede, e as configurações gerais do ambiente distribuído. Para adicionar novos componentes, por exemplo, um novo serviço de dados, é necessário alterar esse arquivo para manter as configurações persistentes em caso de reinicialização da estrutura. O serviço de gerenciamento do *cluster* fornece um ambiente administrativo para realizar essas alterações em execução, sem necessidade de parada do SGBD.

```
[ndbd default]
NoOfReplicas = 1
DataMemory = 5GB
datadir = /var/lib/mysql
LockPagesInMainMemory=1
TimeBetweenLocalCheckpoints=6
NoOfFragmentLogFiles=500

[ndb_mgmd]
NodeId = 1
hostname = 10.44.0.1

[ndbd]
NodeId = 2
hostname = 10.44.0.2

[ndbd]
NodeId = 3
hostname = 10.47.0.1

[mysqld]
NodeId = 4
hostname = 10.36.0.1

[mysqld]
NodeId = 5
hostname = 10.36.0.2
```

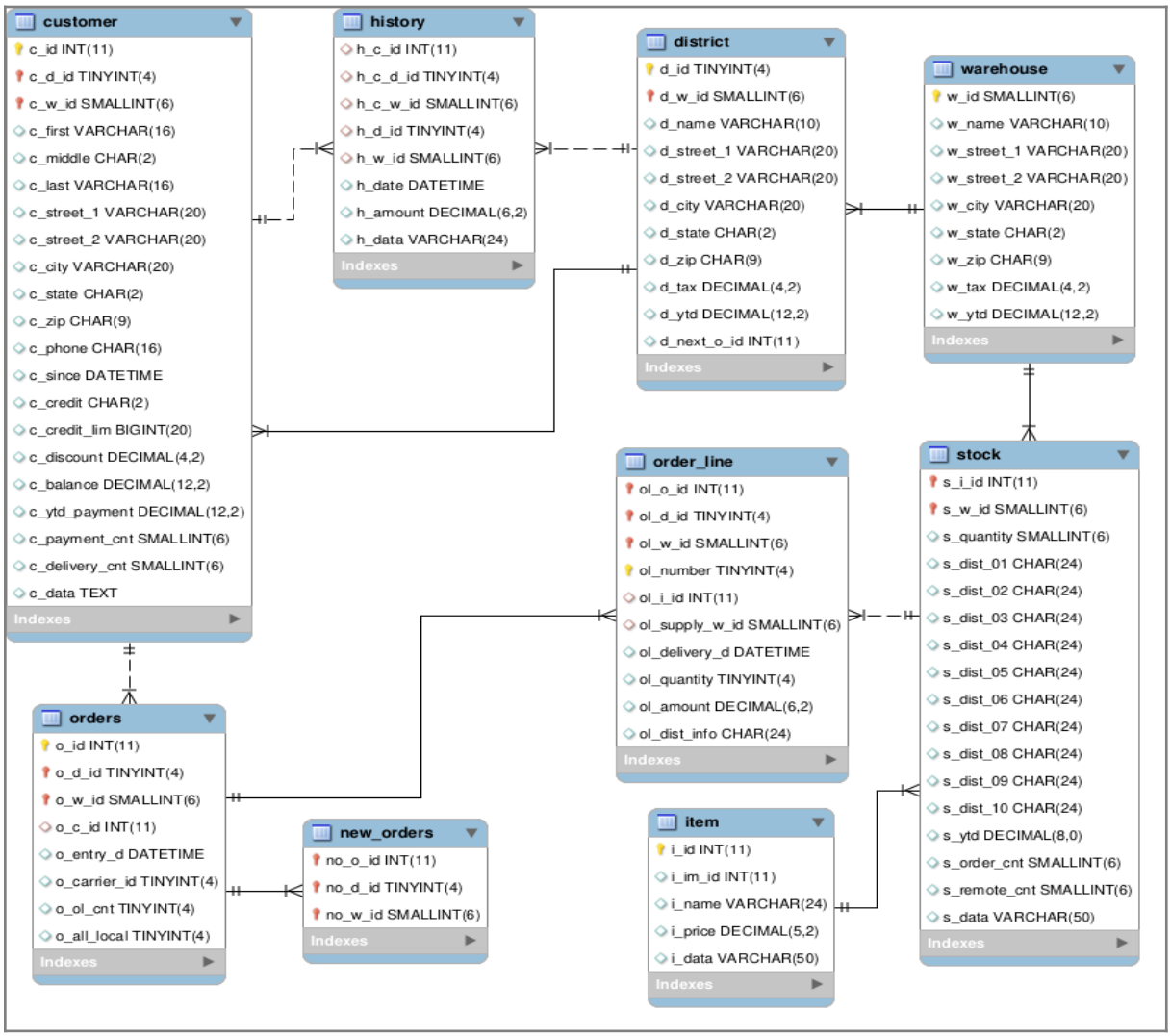
Arquivo **cluster_my.cnf**, possui as configurações padrões para os serviços do *cluster*, identificando que os serviços devem executar no ambiente distribuído.

```
[mysqld]

default-authentication-plugin = mysql_native_password
log_timestamps = SYSTEM
datadir = /var/lib/mysql
default_storage_engine=ndbcluster
max_connections = 1000

ndbcluster
ndb-connectstring = 10.44.0.1
user = root

[mysql_cluster]
ndb-connectstring = 10.44.0.1
```

APÊNDICE E – MODELO RELACIONAL DAS TABELAS GERADAS PELAS FERRAMENTAS

Os modelos relacionais foram gerados com a ferramenta *MySQL Workbench*.

Figura 37 – Modelo relacional da ferramenta *Percona's TPCC-MySQL Tool*



Fonte: Autor (2019).

Figura 38 – Modelo relacional da ferramenta *sysbench*

Fonte: Autor (2019).

APÊNDICE F – RESULTADOS DAS EXECUÇÕES DO *BENCHMARKING*

Tabela 1 – Resultados execução *benchmark* com *sysbench*

1GB	**Replicação (1 nodo)**	**Replicação (2 nodos)**	**Particionamento funcional**	***Data sharding***	***Clustering* (1 nodo)**	***Clustering* (2 nodo)**
8	5760,63	8507,13	7334,39	9195,28	2767,15	2397,74
16	6066,86	10699,76	7890,85	11461,63	3216,61	2707,02
32	6084,51	11445,69	8055,07	12334,53	3428,61	2871,4
64	6005,15	11537,15	8091,42	12355,23	3466,61	2941,95
5GB	**Replicação (1 nodo)**	**Replicação (2 nodos)**	**Particionamento funcional**	***Data sharding***	***Clustering* (1 nodo)**	***Clustering* (2 nodo)**
8	5296,69	8012,93	7167,83	8831,13	3800,09	3315,13
16	5604,75	10230,33	7526,4	10928,35	4337,94	3709,21
32	5598,74	11046,9	7895,17	11904,47	4589,02	3899,98
64	5418,93	11031,5	7995,55	12416,47	4604,69	3942,16
10GB	**Replicação (1 nodo)**	**Replicação (2 nodos)**	**Particionamento funcional**	***Data sharding***	***Clustering* (1 nodo)**	***Clustering* (2 nodo)**
8	5342,12	7902,61	6849,61	8579,07	3969,87	3350,89
16	5796,6	10666,82	7411,61	10727,16	4417,81	3703,64
32	5778,47	11510,09	7643,69	11794,7	4613,33	3875,24
64	5574,29	11557,08	7917,12	12326,24	4666,23	3982,08

Fonte: Autor (2019).

Tabela 2 – Resultados execução *benchmark* TPC-C

1GB	**Replicação (1 nodo)**	**Replicação (2 nodos)**	***Clustering* (1 nodo)**	***Clustering* (2 nodos)**
8	93,55	102,72	42,62	39,81
16	117,76	114,56	49,90	45,48
32	126,30	114,43	51,59	47,70
64	124,71	111,86	52,26	48,90
5GB	**Replicação (1 nodo)**	**Replicação (2 nodos)**	***Clustering* (1 nodo)**	***Clustering* (2 nodos)**
8	91,30	91,08	31,61	29,99
16	115,26	108,07	32,76	29,69
32	120,41	108,29	31,41	29,54
64	118,27	106,04	28,24	29,43
10GB	**Replicação (1 nodo)**	**Replicação (2 nodos)**	***Clustering* (1 nodo)**	***Clustering* (2 nodos)**
8	78,64	78,82	32,86	29,89
16	101,68	97,98	32,37	31,33
32	104,31	96,62	29,75	29,76
64	103,02	97,38	28,43	27,31

Fonte: Autor (2019).