CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA
PAULA SOUZA

FACULDADE DE TECNOLOGIA DE LINS PROF. ANTONIO SEABRA
CURSO SUPERIOR DE TECNOLOGIA EM BANCO DE DADOS

ANA RENATA CARDOSO DE ALMEIDA

UM ESTUDO SOBRE APLICAÇÃO DE BENCHMARK EMSISTEMA DE BANCO DE DADOS DISTRIBUÍDO HOMOGÊNEO BASEADO EM POSTGRESQL

LINS/SP
1º SEMESTRE/2016

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA

FACULDADE DE TECNOLOGIA DE LINS PROF. ANTONIO SEABRA
CURSO SUPERIOR DE TECNOLOGIA EM BANCO DE DADOS

ANA RENATA CARDOSO DE ALMEIDA

UM ESTUDO SOBRE APLICAÇÃO DE BENCHMARK EM SISTEMA DE BANCO DE DADOS DISTRIBUÍDO HOMOGÊNEO BASEADO EM POSTGRESQL

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Lins para obtenção do Título de Tecnólogo (a) em Banco de Dados.

Orientador: Prof. Dr. Mário Henrique de Souza Pardo

LINS/SP
1º SEMESTRE/2016

ANA RENATA CARDOSO DE ALMEIDA

UM ESTUDO SOBRE APLICAÇÃO DE BENCHMARK EM SISTEMA DE BANCO DE DADOS DISTRIBUÍDO HOMOGÊNEO BASEADO EM POSTGRESQL

Trabalho de Conclusão de Curso apresentado à Faculdade de Tecnologia de Lins, como parte dos requisitos necessários para a obtenção do título de Tecnólogo (a) em Banco de Dados sob orientação do Prof. Dr. Mário Henrique de Souza Pardo

Data de Aprovação: 28/06/2016

_____

Orientador: Prof. Dr. Mário Henrique de Souza Pardo

_____

Prof. Me. Felipe Maciel Rodrigues

_____

Prof. Me. Adriano Bezerra

Dedicatória

Aos meus pais, Marcos Cardoso de Almeida e Sonia Tobias Prado e a minha avó Maria Lila Tobias.

ANA RENATA CARDOSO DE ALMEIDA

AGRADECIMENTOS

Gostaria de agradecer primeiramente a todos que fizeram parte direta ou indiretamente da realização deste trabalho.

A todos os professores e funcionários da FATEC que tive oportunidade de conviver e por concederem conhecimento, atenção e disposição sempre que necessário. Todos foram extremamente importantes nesta jornada e lhes sou grata por cada detalhe que aprendi com cada um.

Manifesto também minha imensa gratidão ao meu orientador, Prof. Dr. Mário Henrique de Souza Pardo pela amizade e por ter me auxiliado, apoiado e incentivado durante esta monografia.

Por fim, agradeço especialmente aos meus pais Marcos Cardoso de Almeida e Sonia Tobias Prado, meu irmão João Marcos Cardoso de Almeida e minha avó, Maria Lila Tobias, *in memorian* pelo encorajamento e por estarem sempre comigo.

ANA RENATA CARDOSO DE ALMEIDA

RESUMO

As arquiteturas de sistemas de banco de dados evoluíram com o passar do tempo. Antes ineficientes e limitados, hoje gradativamente mais adaptados para o que é imprescindível diante da demanda de seus usuários e o complexo contexto de uso massivo de aplicações Por meio de rede de computadores. Para atender a essas necessidades, é fundamental que não haja falhas que possam comprometer o funcionamento computacional, uma solução que vem ganhando espaço no mercado de grandes sistemas de informação é a implementação de arquiteturas de banco de dados distribuídos. Diante disto, é importante a realização de avaliações de desempenho para garantir qualidade de processamento e as capacidades de um sistema desta magnitude. Este trabalho visa aferir o desempenho de um banco de dados distribuído por meio de um *benchmark*, para que seja possível analisar seu comportamento diante de um ambiente que demanda grandes números de requisições, de usuários e de outros fatores considerados cruciais para um cenário próximo de um ambiente real de produção. Além disso, é apresentado o planejamento de experimentos e sistemática de avaliação de resultados a serem seguidos. Para implementação do cenário de teste foram utilizados como tecnologias: o sistema de gerenciamento de banco de dados PostgreSQL 9.4 e uma ferramenta de *benchmark* que utiliza o protocolo TPC-B, desenvolvido pela corporação sem fins lucrativos TPC, utilizado para medir a taxa de transferência e quantas transações por segundo um sistema pode executar.

Palavras-chave: Banco de dados distribuído, benchmarking, TPC-B, PgBench, avaliação de desempenho, PostgreSQL.

ABSTRACT

The database system architectures have evolved over the time. Once inefficient and limited, today, gradually more adapted to what is essential given the demands of its users and the complex context of massive use of applications through computer network. To meet these needs, it is essential that there are no flaws that could compromise the computer operation, a solution that is becoming more popular in the large information systems market is the implementation of distributed database architectures. Knowing this, it is important to conduct performance evaluations to ensure processing quality and the capabilities of such system intensity. This work aims to assess the performance of a distributed database through a benchmark, so it´s behavior can be analyzed in an environment that demands large numbers of requests, users and other factors considered crucial for setting near a real production environment. Furthermore, the systematic planning experiments and results of evaluation is shown to be followed. To implement the test scenario were used as technologies: the database management system PostgreSQL 9.4 and a benchmarking tool that uses the TPC-B protocol, developed by non-profit corporation TPC, used to measure the throughput of how many transactions per second a system can run.

Keywords: Distributed database, benchmarking, TPC-B, PgBench, performance evaluation, PostgreSQL.

LISTA DE ILUSTRAÇÕES

Figura 1.1 - Sistemas de banco de dados 17
Figura 1.2 - Sistema distribuído organizado como Middleware 18
Figura 1.3 - Banco de dados distribuídos heterogêneos 21
Figura 1.4 - Banco de dados distribuídos homogêneos 22
Figura 1.5 – Protocolos TPC e suas destinações 23
Figura 1.6 – Perfil das transações TPC-B 28
Figura 1.7 – Diagrama banco de dados TPC-B 28
Figura 2.1 – Ambiente de banco de dados distribuído 30
Figura 2.2 – Funções das camadas de um BDD. 31
Figura 2.3 – Replicações síncronas e assíncronas. 35
Figura 2.4 – Instâncias e réplicas 36
Figura 2.5 – Arquitetura do Pgpool 37
Figura 2.6 – Descrição e detalhes da execução 38
Figura 3.1 – Arquivo pool_hba.conf 40
Figura 3.2 – Disposição do ambiente de testes 41
Figura 3.3 – Inicialização Pgbench 43
Figura 3.4 – Planejamento de experimentos 44
Figura 4.1 – Variáveis TPS 100 transações. 46
Figura 4.2 – Variáveis TPS 1000 transações. 48
Figura A.1 – Primeiros passos Debian 57
Figura A.2 – Configuração de rede 58
Figura A.3 – Configuração de usuários e senhas 58
Figura A.4 – Opções de particionamento de discos 59
Figura A.5 – Seleção de disco 59
Figura A.6 – Esquema de particionamento 60
Figura A.7 – Seleção de software 61
Figura A.8 – Carregador de inicialização GRUB 61
Figura A.9 – Finalização da instalação. 62
Figura A.10 – Prompt de comando Debian. 62
Figura B.1 – *Setup*PostgreSQL 63
Figura B.2 – InstalaçãoPostgreSQL 63

Figura B.3 – Instalador de pacotes adicionais ao PostgreSQL 64
Figura B.4 – Pacotes adicionais 64
Figura B.5 – Selecionando componentes 65
Figura B.6 – Selecionando o diretório 66
Figura B.7 – Definição de *password* 66
Figura B.9 – Seleção de localização cluster banco de dados 67

LISTA DE TABELAS

Tabela 3.1 – Distribuição de endereços IP..... 42

Tabela 3.2 – Tabela inicial Pgbench 44

LISTA DE ABREVIATURA E SIGLAS

BDD - Banco de dados distribuídos

BSD –*Berkeley Software Distribution*

OLTP –*Online transaction processing*

SGBD - Sistema de gerenciamento de banco de dados

SQL –*Structured Query Language*

TPC - Transaction Processing Performance Council

VM – Virtual Machine

TPS – Transações por segundo

CPU – *Central Processing Unit*

IO – *Input*/*Output*

SUMÁRIO

INTRODUÇÃO 14

1 FUNDAMENTAÇÃO TEÓRICA 16

1.1 BANCO DE DADOS 16

1.2 SISTEMAS DISTRIBUÍDOS 17

1.3 BANCO DE DADOS DISTRIBUÍDOS 19

1.3.1 Banco de dados heterogêneos 20

1.3.2 Banco de dados homogêneos 21

1.4 BENCHMARKING PARA BANCO DE DADOS 22

1.5 POSTGRESQL 25

1.6 PGBENCH 27

1.6.1 Entidades e relacionamentos 28

2 BANCO DE DADOS DISTRIBUÍDOS 30

2.1 AMBIENTE DISTRIBUÍDO 30

2.2 CLUSTERS 32

2.2.1 Arquitetura cluster em banco de dados 33

2.3 REPLICAÇÃO 34

2.3.1 Tipos de replicação 34

2.3.2 Fragmentação e replicação 35

2.4 PGPOOL 36

3 DESENVOLVIMENTO DO TRABALHO 39

3.1 AMBIENTE DE TESTES 39

3.2 METODOLOGIA 42

3.3 RESULTADOS ESPERADOS 44

4 RESULTADOS E DISCUSSÃO 46

4.1 RESULTADOS EXPERIMENTAIS 46

CONCLUSÃO 51
TRABALHOS FUTUROS 53
REFERÊNCIAS BIBLIOGRÁFICAS 54
ANEXO A – INSTALAÇÃO DEBIAN 57
ANEXO B – INSTALAÇÃO POSTGRESQL 63
ANEXO C – INSTALAÇÃO PGPOOL 69

INTRODUÇÃO

Os sistemas de banco de dados são componentes essenciais para as organizações, já que asseguram maior controle de dados vitais para empresas, além de permitir o armazenamento, gerenciamento e disposição de informações que fazem parte de nosso cotidiano. Os bancos de dados devem priorizar informações em tempo real e principalmente oferecer disponibilidade de acesso a qualquer momento.

De acordo com Rainer Jr, Cegielski (2011), o uso de banco de dados eliminou problemas que anteriormente eram causados pela maneira de armazenamento de acesso aos dados. Os softwares de banco de dados minimizam redundância, isolamento e incoerência de dados e maximiza a segurança, integridade e independência de cada informação contida nos bancos de dados.

A centralização dos dados, ou seja, todos os dados guardados em um só banco de dados, não permite o real objetivo de tornar os dados mais facilmente disponíveis ao usuário final em aplicações geograficamente dispersas. A relação entre os custos de processamento e de comunicações alteraram-se com a queda acentuada do custo de processadores, mas não do custo de transmissão de dados. Ou seja, a estratégia de trazer os dados a um processador central pode ser mais cara do que trazer capacidade computacional ao local de geração ou uso dos dados. Finalmente, sistemas centralizados apresentam vulnerabilidade maior a falhas e nem sempre permitem um crescimento gradativo da capacidade computacional instalada de forma simples e adequada. (CASANOVA e MOURA, 1999).

Sendo assim, os bancos de dados distribuídos que são definidos como uma “coleção de sistemas de banco de dados parcialmente independentes que compartilham um esquema comum e coordenam o processamento de transações que acessam dados não locais.” (SILBERSCHATZ, et al. 2012, p. 500), ganharam notoriedade e importância para as organizações.

Mannino(2008) afirma que existem diversas vantagens para utilização de banco de dados distribuídos, como controle local dos dados, melhor desempenho, já que os dados oferecem maior disponibilidade, menor custo de comunicação e maior confiabilidade. Além disso, os dados podem ser replicados para posteriormente serem disponibilizados para mais de um local.

Diversos trabalhos já abordaram variadas questões de bancos de dados distribuídos e se fazem necessários para que a performance seja mais satisfatória. Existem fatores que consideram avaliação de seu desempenho de acordo com tráfego de dados de rede, visando avaliar a influência de número de *hosts* para não dificultar a circulação de pacotes. Do mesmo modo, se faz necessário avaliar este mesmo elemento considerando o ambiente de aplicação web, para que o número de requisições e tempo médio de resposta entre um servidor web e um servidor de banco de dados seja mensurável. Há também o foco em seleção de dados em banco de dados distribuídos, que visa determinar coeficientes influentes, tais como quantidade de requisições solicitadas, estrutura interna e número total de tabelas.

Este estudo justifica-se pela necessidade crescente de melhor desempenho e disponibilidade de um banco de dados distribuído, que deve sempre retornar e realizar consultas no menor tempo possível, além da importância da distribuição de um banco de dados homogêneo, que permite menos chances a falhas, já que, se um computador em rede falhar, ainda haverão outros para darem continuidade ao trabalho através da replicação dos dados.

Mediante o que foi descrito, o objetivo desta monografia é avaliar o desempenho do banco de dados PostgreSQL em um ambiente distribuído, por meio de ferramentas disponíveis para medir consultas e realizar testes que utilizam todo o poder de processamento do sistema. Por meio deste estudo, será possível saber o tempo e disponibilidade dos dados que foram requisitados.

Este trabalho está organizado em quatro capítulos. O primeiro capítulo contém as tecnologias necessárias para a elaboração e desta monografia, com explanações detalhadas de cada uma. O segundo capítulo aborda conceitos aprofundados a cerca dos bancos de dados distribuídos,replicação de dados e a ferramenta PgPool. O terceiro capítulo apresenta os detalhes sobre o desenvolvimento do projeto. Os resultados obtidos serão apresentados no quarto capítulo. Por fim, as conclusões sobre o trabalho.

1 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão apresentadas as tecnologias utilizadas para a realização desta monografia.

1.1 BANCO DE DADOS

Banco de dados pode ser conceituado, de acordo com Elmasri e Navathe (2011)como uma coleção de dados relacionados entre si, denominados de fatos conhecidos. Um fato conhecido pode ser registrado em um banco de dados e tem um significado implícito, comumente conhecido como aspectos do mundo real, que podem ser nomes, endereços, telefones e outras informações relevantes. Um banco de dados é planejado e constituído de dados que possuem uma finalidade específica, significando que, uma variedade aleatória de informações não podem ser confundidas com banco de dados.

Conforme Ramakrishnan e Gehrke (2008), para gerenciar todas essas informações, é fundamental a utilização de um *software* conhecido como Sistema de Gerenciamento de Banco de Dados (SGBD), projetado essencialmente para ajudar na manutenção e utilização dos dados. Provedor de diversas vantagens, um SGBD consiste em manter a independência dos dados, ou seja, oferecer uma visão abstrata dessas informações, ocultando determinados detalhes como meio de armazenamento e manutenção desses dados. Também é possível armazenar e recuperar dados eficientemente; garantir a segurança dos dados por meio de controles de acesso, definindo a visibilidade de determinadas informações; proteger usuários dos efeitos de falhas de sistema, e vários outros aspectos que garantem a vantagem de utilização de um SGBD.

> Um sistema de banco de dados é projetado para armazenar grandes volumes de informações. O gerenciamento de informações implica a definição das estruturas de armazenamento destas informações e o fornecimento de mecanismos para sua manipulação. Além disso, o sistema de banco de dados precisa proporcionar segurança ao armazenamento de informações, diante de falhas do sistema ou acesso não autorizado. Se os dados são compartilhados entre diversos usuários, o sistema precisa evitar possíveis resultados anômalos. (SILBERSCHATZ; KORTH; SUDARSHAN, 2012, p. 17)

Na figura 1.1, é exemplificado como um sistema de banco de dados e um sistema de gerenciamento de banco de dados são utilizados.

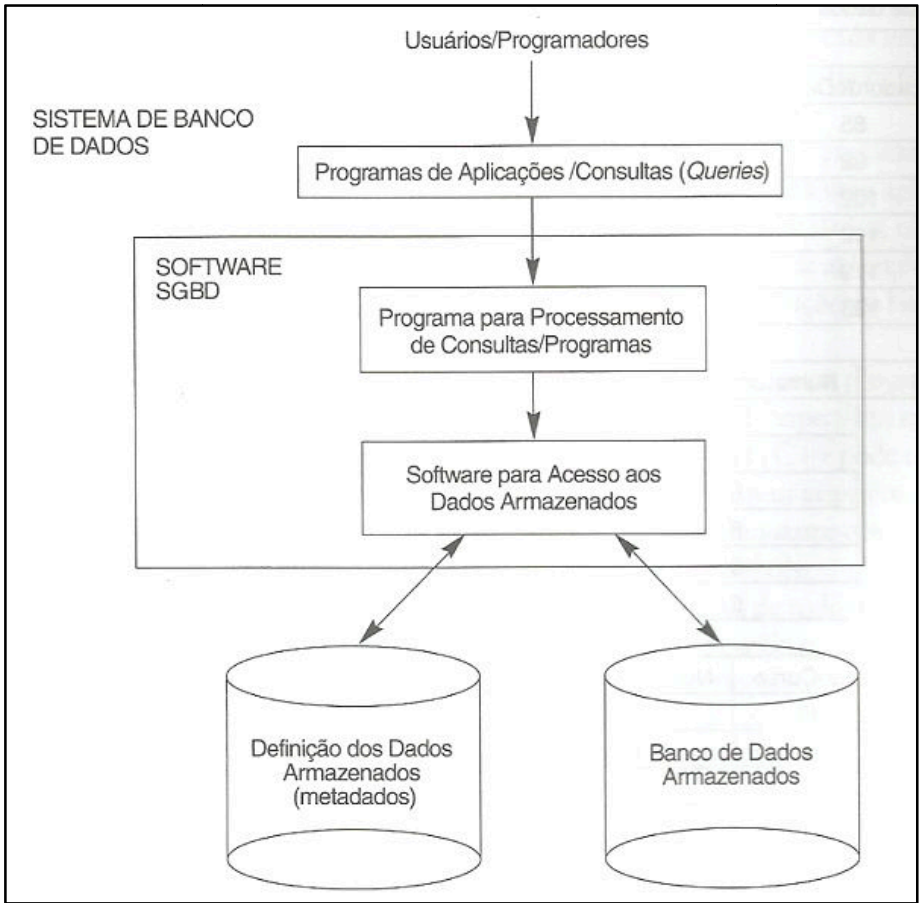

Figura 1.1 - Sistemas de banco de dados
Fonte:Elmasri e Navathe, 2011, p. 5

1.2 SISTEMAS DISTRIBUÍDOS

De acordo com Tanenbaum e Van Steen (2007), sistemas distribuídos são conceituados como computadores que operam em conjunto, porém são independentes entre si, apresentando-se aos seus usuários como um sistema único e proporcionando meios de comunicação simultânea entre componentes de uma única aplicação ou aplicações divergentes. Conjuntamente, oculta de seus usuários as diferenças em *hardware* e os sistemas operacionais para cada serviço.Os computadores e outros dispositivos que utilizam a rede para propósitos de comunicação são referidos como *hosts*. O termo nó é usado para especificar qualquer computador ou dispositivo de comunicação ligado à rede. Todos os *hosts* se comunicam entre si através da internet.

> Para suportar computadores e redes heterogêneos e, simultaneamente, oferecer uma visão de sistema único, os sistemas distribuídos costumam ser organizados por meio de uma camada de software – que é situada logicamente entre uma camada de nível mais alto, composta de usuários e aplicações, e uma camada subjacente, que consiste em sistemas operacionais e facilidades básicas de comunicação. Tal sistema distribuído, ás vezes é denominado *middleware*. (TANENBAUM e VAN STEEN, 2007, p. 2)

Conforme Coulouris, Dollimore e Kindbeg (2007), um *middleware* é um *software* com a finalidade de ocultar a heterogeneidade, como a variedade de redes, sistemas operacionais, *hardware* e linguagens de programação diferentes e oferecer um modelo de programação conveniente para seus utilizadores, através da disponibilização de blocos básicos de construção para montagem de componentes de *software* que possam trabalhar paralelamente em sistemas distribuídos, simplificando atividades de comunicação de programas aplicativos. Na figura 1.2 é demonstrado como um sistema distribuído é projetado, detalhando a localidade do *middleware*.

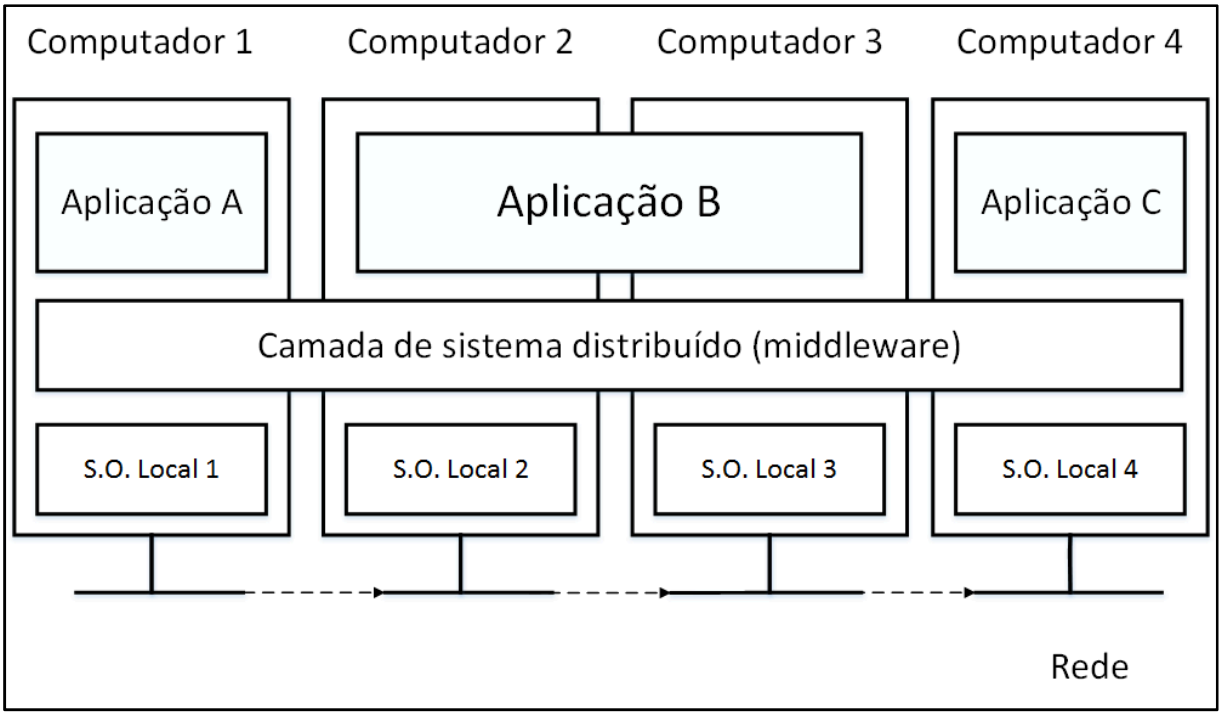

Figura 1.2 - Sistema distribuído organizado como Middleware
Fonte: Adaptado de Tanenbaum e Van Steen, 2007, p. 2

Os autores ainda ressaltam que, sistemas distribuídos devem possuir determinadas características, como:

- Heterogeneidade: para garantir que todos os componentes heterogêneos possuam a capacidade de operar entre si.
- Abertura: para que as interfaces permitam que componentes sejam adicionados ou substituídos.
- Segurança: para certificar que o sistema será utilizado para seu devido fim.
- Escalabilidade: em razão do sistema funcionar eficientemente, com o número de usuários crescendo continuamente.
- Tratamento ou Tolerância a falhas: se um componente falhar, os outros não devem ser descontinuados igualmente, levando a inutilidade do sistema como um todo.
- Transparência: atestando que todos os processos e distribuições não devem ser percebidos pelo seu usuário. A base de dados está fisicamente distribuída, mas não tem necessidade de ser perceptível, dando a impressão de que é totalmente integrada.

1.3 BANCO DE DADOS DISTRIBUÍDOS

Os bancos de dados distribuídos surgiram da associação de duas tecnologias computacionais: tecnologia de banco de dados e tecnologia de rede e comunicação de dados. Os primeiros bancos de dados eram totalmente centralizados, ou seja, continham dados em um só lugar, criando sobrecarga. Para solucionar isso, a descentralização surgiu, elaborando protótipos de pesquisas para tratar questões de distribuições de dados e gerenciamento de metadados em banco de dados distribuídos para gerar fontes de dados heterogêneas.(ELMASRI; NAVATHE,2011)

> Banco de dados distribuídos é definido como uma coleção de múltiplos bancos de dados, logicamente inter-relacionados, distribuídos por uma rede de computadores. Um sistema de gerenciamento de banco de dados distribuídos, é definido como o sistema de software que permite a gestão do banco de dados distribuído, tornando a distribuição transparente para os usuários. Os dois termos importantes para o esclarecimento do que é um banco de dados distribuídos são "logicamente inter-relacionados" e "distribuído por uma rede de computadores". (OZSÜ e VALDURIEZ, 2011, p. 3)

A vulnerabilidade a falhas fica maior quando um único computador, no qual todos os programas executados, acessam um banco. Os dados dificilmente ficam disponíveis ao usuário final, quando se utiliza aplicações de amplitude geográfica. Os banco de dados distribuídos são uma escolha atrativa, pois oferecem maior autonomia e responsabilidade local ao usuário. Com o crescimento gradativo das redes de comunicação de dados, a interligação de vários processadores independentes foi permitido, gerando custos de comunicação menores, já que a maior parte dos acessos gerados em um nó podem ser resolvidos localmente. A projeção deste sistema pode ser feito para permitir o crescimento da aplicação, apenas acrescentando novos processadores e módulos do banco ao sistema e melhorar a disponibilidade de confiabilidade, através da replicação de dados. (CASANOVA; MOURA, 1999, p.2)

Os banco de dados distribuídos (BDD) podem sem classificados em dois grupos: Heterogêneos e Homogêneos. Estes tipos de BDD serão conceituados e melhor explanados nas duas próximas seções do trabalho.

1.3.1 Banco de dados heterogêneos

De acordo com Ramakrishnan e Gehrke (2008), banco de dados distribuídos heterogêneos estão inter-conectados para permitir o acesso aos dados a partir de vários locais, porém, controlados a partir de diferentes sistemas gerenciadores de banco de dados de forma autônoma. Para a construção bem sucedida de um sistema heterogêneo, é fundamental um protocolo de *gateway*. Através do acesso aos servidores de banco de dados por meio deste protocolo, as diferenças de *software* e servidores são disfarçadas satisfatoriamente.

A heterogeneidade em banco de dados distribuídos pode se resumir em diferença de *hardware* e protocolos de rede para variações em gestão de dados. Também está presente em modelos de dados, linguagens de consulta e protocolos de gerenciamento de transações. Heterogeneidade em linguagens de consulta, envolve o uso de diferentes paradigmas de acesso a dados em modelos de dados diferentes. Da mesma forma, abrange diferentes idiomas, mesmo quando sistemas individuais usam o mesmo modelo de dados. Embora o *Structured Query Language* (SQL) seja uma linguagem de consulta relacional padrão, ainda existem muitas implementações diferentes e cada fornecedor possui uma linguagem de significado

diferente. (OZSÜ; VALDURIEZ, 2011, p. 27). Na Figura 1.3, é ilustrado um banco de dados heterogêneo *Oracle* e um outro SGBD, com o acesso permitido através do protocolo de *Gateway*.

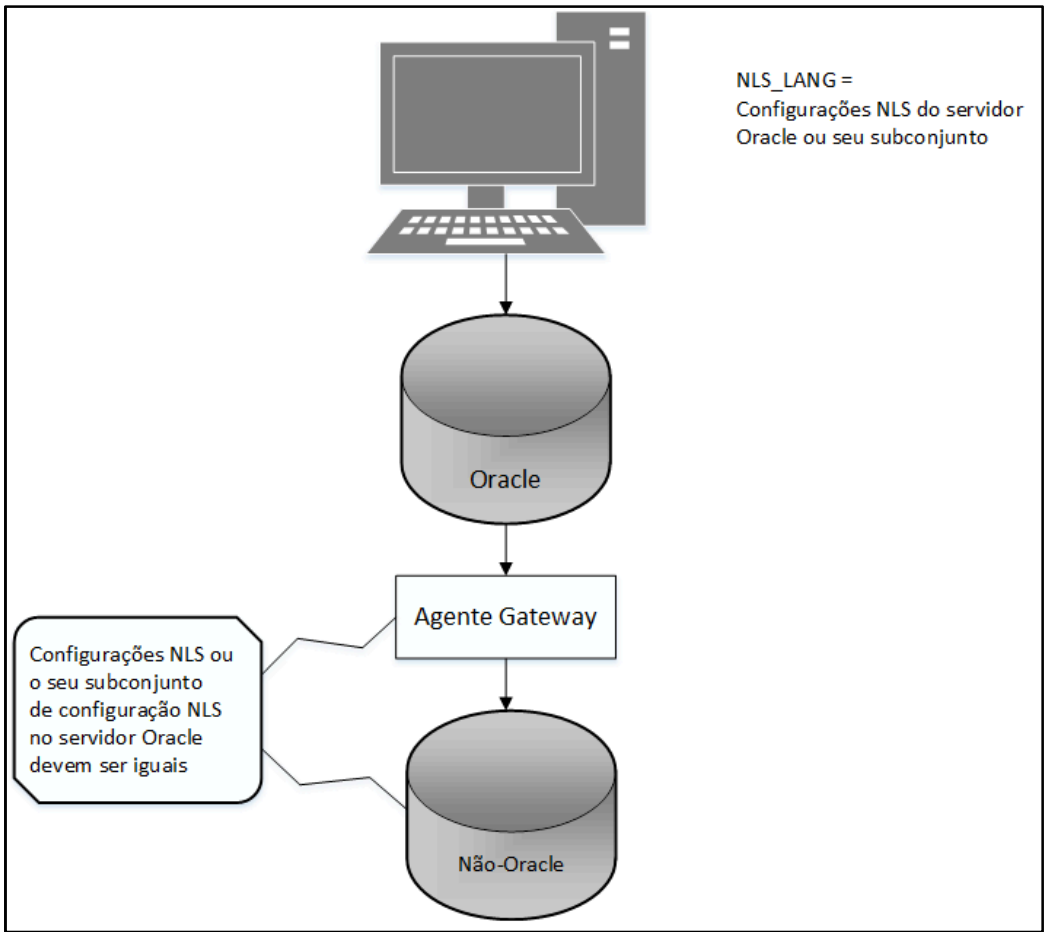

Figura 1.3 - Banco de dados distribuídos heterogêneos
Fonte: Adaptado de Oracle, 2014.

1.3.2 Banco de dados homogêneos

De acordo com Casanova e Moura (1999), um banco de dados distribuído é homogêneo quando seus sistemas de gerenciamento de banco de dados locais que utilizam o mesmo *software* são iguais, oferecendo interfaces idênticas e mesmos serviços aos usuários, em nós diferentes. É utilizado com mais frequência quando a aplicação destinada nunca existiu anteriormente. Na Figura 1.4, é exemplificado um banco de dados homogêneo, inter-conectado com banco de dados Oracle.

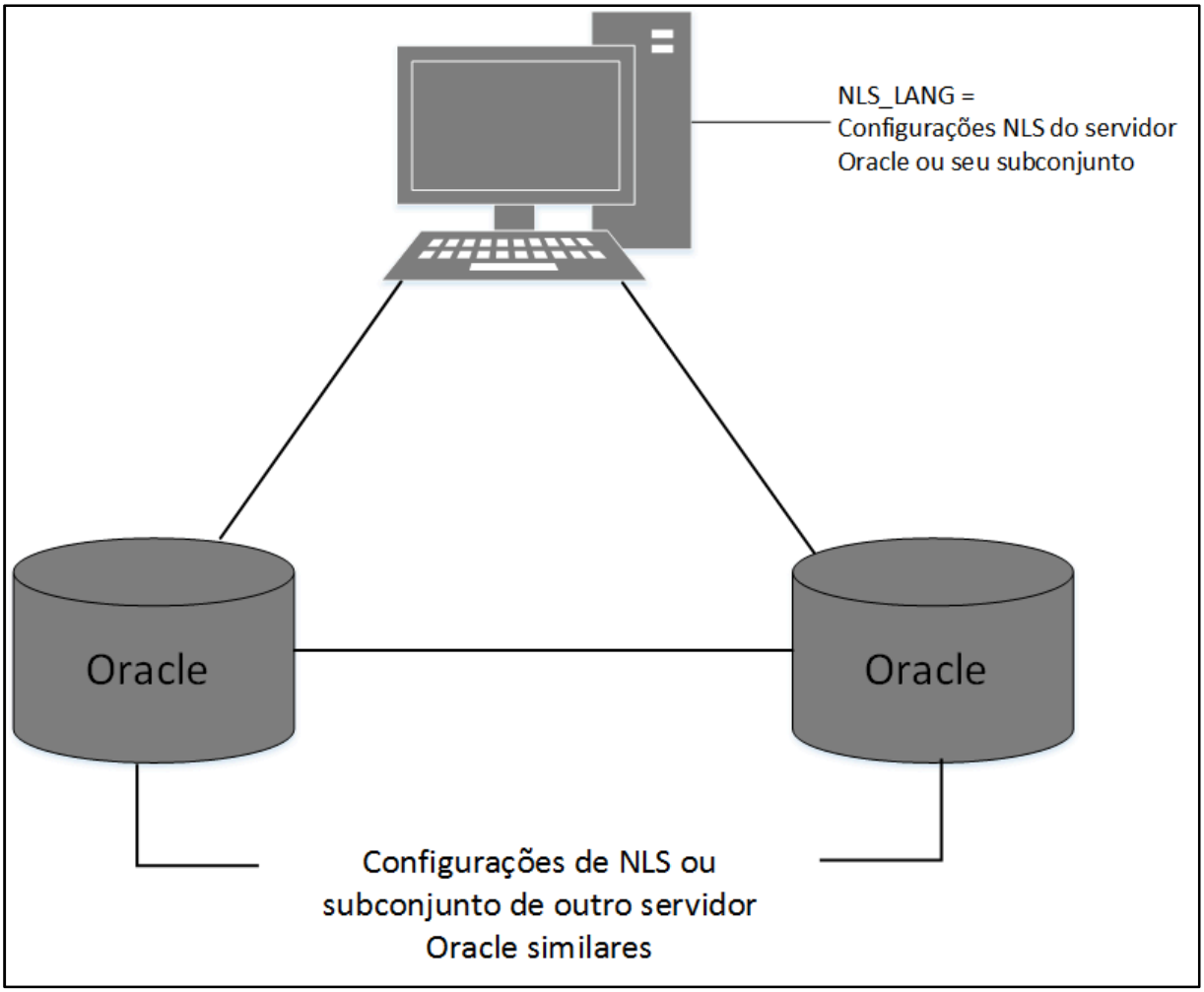

Figura 1.4 - Banco de dados distribuídos homogêneos
Fonte: Adaptado de Oracle, 2014.

Segundo Silberschatz, Korth e Sudarshan (2012), neste tipo de sistema, os sítios locais entregam uma parte de sua autonomia em termos do seu direito de mudar esquemas ou *software* de sistema de gerenciamento de banco de dados. Este *software* também precisa cooperar com outros sítios na troca de informações sobre transações, para tornar o processamento da transação possível entre vários sites.

1.4 BENCHMARKING PARA BANCO DE DADOS

Segundo o site da Oracle, as organizações e os usuários devem ser capazes de utilizar ferramentas da tecnologia da informação de acordo com suas necessidades de negócios. Para isso, a indústria da informática criou um conjunto de *benchmarks*, que ajudam a realizar várias funções como comparar o desempenho

dos sistemas de diferentes fornecedores, determinar o desempenho e o tempo de resposta do sistema para cargas de trabalho específicas, obtenção de requisitos de *data centers* para determinado aplicativo de destino ou padrão de uso, medir e informar sobre melhorias de desempenho do sistema ao longo do tempo.

Para medir o desempenho de banco de dados, uma organização sem fins lucrativos chamada *Transaction Processing Performance Council* (TPC) criou vários *benchmarks* específicos para cada tipo de funcionalidade: *Data warehouse*, *Online Transaction Processing* (OLTP) e banco de dados distribuídos, detalhado na Figura 1.5. De acordo com a TPC (2014), a finalidade principal da corporação é definir o processamento de transações e valores de referência de banco de dados, para troca de informações objetivas. A TPC produz *benchmarks* que medem o processamento de transações e desempenho de banco de dados em termos de quantas transações de um determinado sistema e banco de dados pode executar por unidade de tempo, transações por segundo ou transações por minuto.

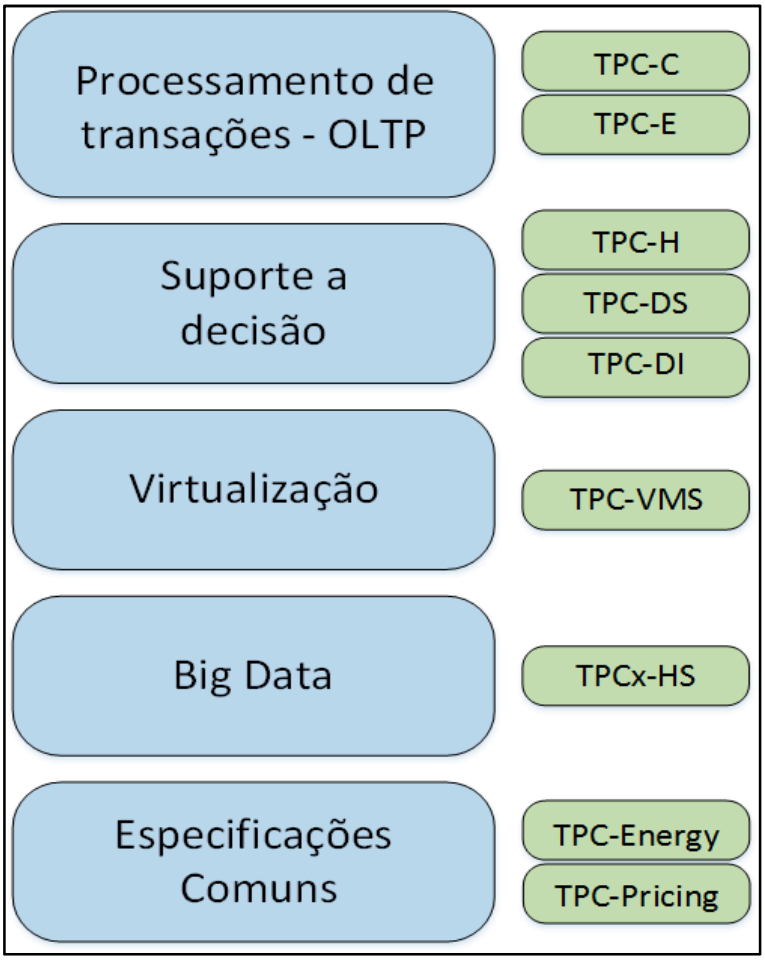

Figura 1.5 – Protocolos TPC e suas destinações
Fonte: Adaptado de TPC, 2015.

O *benchmark* criado pela TPC considerado padrão pela corporação é o TPC-C, que simula um ambiente completo de computação, no qual os usuários executam transações em um banco de dados. O valor de referência utilizado está de acordo com as principais atividades de um ambiente de entrada, que podem incluir entrega de encomendas, registro de pagamentos, verificação de *status* de pedidos, acompanhamento de nível de estoque em armazéns. O TPC-C não se limita somente a uma atividade ou segmento em particular, enquadrando-se em qualquer indústria que precise gerir, vender ou distribuir um produto ou serviço, conforme TPC (2014).

TPC-DS é um *benchmark* de apoio à decisão de sistemas modernos, que manipulam grandes volumes de dados e utilizam respostas para questões de negócios do mundo real. O TPC-DS executa consultas com requisitos operacionais de alta complexidade, elevando a carga de *Central Processing Unit* (CPU) e *Input*/*Output* (IO).

TPC-E, simula cargas de trabalho OLTP de uma corretora com os clientes gerando transações relacionadas a negócios, consultas de conta e pesquisa de mercado. Sua métrica é dada em transações por segundo (TPS), que refere-se ao número de transações, descrevendo o número de transações que um servidor pode sustentar em um longo período de tempo.

TPC-H também é um *benchmark* de apoio a decisão para computadores conectados entre si. Composto por um conjunto de consultas de negócios orientados e modificações de dados simultâneas, é de grande relevância no setor a ser utilizado. Este ilustra sistemas de apoio à decisão que analisam grandes volumes de dados, executam *queries*, que são operações de banco de dados nas quais recuperam informações de uma ou mais tabelas (ORACLE, 2016), sendo complexas e fornecem respostas a questões críticas de negócios.

TPC-VMS reporta o desempenho de banco de dados virtualizados. Sua intenção é representar um ambiente de virtualização de cargas de trabalho de banco de dados consolidados em um servidor. A partir disto, escolhe-se um dos quatro *benchmarks* (TPC-C, TPC-E, TPC-H ou TPC-DS) e executa-o em cada uma das máquinas virtuais no sistema. As três máquinas virtuais devem compartilhar os mesmos atributos. A métrica de desempenho deste benchmark é o valor mínimodos três *benchmarks* executados nos ambientes de virtualização.

TPC-A utiliza uma única transação de atualização intensiva para carregar o sistema testado, projetado para exercer os principais componentes de um sistema OLTP. Este *benchmark* mede quantas operações por segundo um sistema pode realizar quando operam com múltiplos terminais.

TPC-R é similar ao TPC-H, porém, este permite realizar otimizações adicionais com base em conhecimentos avançados de consultas. Sua medida de desempenho reflete-se na capacidade que um sistema possui para processar consultas, incluindo tamanho do banco de dados selecionado contra cada consulta executada. Também mede seu poder de processamento de consulta, taxa de transferência de consultas quando apresentadas por vários usuários simultâneos.

O protocolo utilizado neste trabalho definido por padrão pelo programa escolhido para comparação de resultados é o TPC-B, que mede a quantidade de dados transferidos em termos de quantas transações por segundo um sistema pode realizar. Foi projetado especialmente para realizar testes de estresse sobre a parte principal de um sistema de banco de dados. Ele "estressa" a CPU, memória e os dispositivos I/O (dispositivos de entrada/saída).

1.5 POSTGRESQL

Segundo Drake e Worsley (2002), o PostgreSQL é um Sistema Gerenciador de Banco de Dados (SGBD), ou seja, é composto por programas de gerenciamento, armazenamento e acesso aos dados para a manipulação das informações tornar-se mais fácil e prática. Foi desenvolvido através da internet e idealizado a partir do projeto Postgres, na Universidade da Califórnia, em Berkeley. É um banco de dados com características, desempenho e confiabilidade comparáveis aos bancos de dados comerciais. Também apoia várias interfaces de programação, como a linguagem de programação Java e possui transações, *views*, procedimentos armazenados e *constraints* de integridade referencial. A incorporação da linguagem de consulta estruturada (SQL), linguagem usada como padrão nos banco de dados, foi adicionada em 1995 por dois estudantes da Universidade de *Berkeley*, e em 1996, foi disponibilizado via internet com o nome de PostgreSQL.

Segundo Milani (2008), O PostgreSQL utiliza a licença *Berkeley Software Distribution* (BSD), que possui poucas restrições comparado a outras licenças, fazendo com que o código seja acessível para utilização da ferramenta até para fins

comerciais. Não possui um limite de tamanho para seus bancos de dados, sendo que sua única limitação é o *hardware* do computador que está armazenado. É um banco de dados estável, projetado para executar no método 24/7 (24 horas por dia, sete dias na semana).

São características principais, a possibilidade de configurá-lo para que atue como um *cluster* de informações, que envolve o uso de dois ou mais computadores que são interligados e sincronizados entre si, para atender as requisições dos usuários; gerenciamento de várias conexões com o banco de dados uma única vez através de *multithreads*, ou seja, mais de uma pessoa pode acessar a mesma informação sem ocasionar atrasos ou fazer com que os dados sejam corrompidos e também suporta grandes tamanhos de informações em suas tabelas. (MILANI, 2008)

Sua capacidade de armazenamento quanto ao seu tamanho máximo permitido é: banco de dados, ilimitado; Tabelas, 32 *terabytes*; Linhas, 1,6 *terabytes*; Campo, um *gigabyte*; Linhas por tabela, ilimitadas; Colunas por tabela, de 250 a 1.600 dependendo dos tipos de dados utilizados; Índices por tabela, ilimitados; (MILANI, 2008, p. 25)

O PostgreSQL permite várias linguagens de compilação e interpretação para realizar a interface com o banco de dados. Algumas interfaces existentes suportadas são Java (JDBC), ODBC, Perl, Python, Ruby, C, C++, PHP, Lisp, Scheme e Qt dentre outras. *Stored Procedures,* ou seja, componentes PL/SQL que são utilizados para construir diversas aplicações diferentes (ORACLE, 2016), e *Triggers,* procedimentos armazenados dentro do banco de dados, sendo executados quando determinada instrução ocorre (ORACLE, 2016), que podem ser escritas em linguagem C e carregadas na base de dados como uma biblioteca, permitindo estender suas capacidades. Além disso, o PostgreSQL permite que seus desenvolvedores determinem e criem seus próprios tipos de dados personalizados, juntamente de funções de suporte e operadores para definir seu comportamento através de um *framework*. (POSTGRESQL, 2015).

Sua implementação SQL está de acordo com o padrão ANSI-SQL:2008. Possui um catálogo de sistema relacional que suporta múltiplos esquemas por base de dados, sendo também acessível através do esquema de informações (*Information Schema*) definido pelo padrão SQL. Recursos de integridade de dados incluem chaves primárias e estrangeiras com restrição, atualizações e exclusões em cascata, verificação de *constraints*, *constraints* exclusivas e não nulas. Dispõe de várias

extensões e recursos avançados, tais como: colunas de incremento automático através de sequências e *LIMIT*/*OFFSET* que permitem o retorno de resultados parciais. Suporta também índices compostos, originais, parciais e funcionais. (POSTGRESQL, 2015).

1.6 PGBENCH

Trata-se de um programa para executar testes de *benchmark* no banco de dados PostgreSQL, através de uma sequência de comandos SQL em múltiplas sessões de banco de dados. O *benchmark* é baseado no protocolo TPC-B, possuindo três comandos de inicialização, dezesseis opções para realização dos testes, contando com suporte para realizar avaliações de desempenho personalizados através de scripts de SQL. (POSTGRESQL, 2016).

De acordo com TPC, com o objetivo de fornecer dados de desempenho relevantes para seus usuários, o TPC-B ocupa os componentes de banco de dados necessários para execução de tarefas associadas ao ambiente de processamento de transações, dando ênfase em serviços que possuem atualização intensiva. A carga de trabalho destina-se a refletir os aspectos de banco de dados de uma aplicação através de uma atualização intensiva de transações que proporcionam uma unidade simples e repetitiva que carregam o sistema em teste.

O *benchmark* simula um banco hipotético. Cada banco possui um ramo (*branches*), cada ramo possui contadores (*tellers*). O banco possui muitos clientes e cada um possui uma conta. O banco de dados apresenta o saldo do caixa de cada entidade, sendo elas o ramo, caixa e conta, gerando um histórico de operações recentes executados pelo banco. Cada transação representa o trabalho feito quando um cliente realiza um depósito ou uma retirada em sua conta. A operação é realizada por um contador em seu ramo (TPC, 1994). Na Figura 1.6 é demonstrado o perfil das transações.

```
BEGIN TRANSACTION
   Update Account where Account_ID = Aid:
      Read Account_Balance from Account
      Set Account_Balance = Account_Balance + Delta
      Write Account_Balance to Account
   Write to History:
      Aid, Tid, Bid, Delta, Time_stamp
   Update Teller where Teller_ID = Tid:
      Set Teller_Balance = Teller_Balance + Delta
      Write Teller_Balance to Teller
   Update Branch where Branch_ID = Bid:
      Set Branch_Balance = Branch_Balance + Delta
      Write Branch_Balance to Branch
COMMIT TRANSACTION
Return Account_Balance to driver
```

Figura 1.6 – Perfil das transações TPC-B
Fonte: TPC, 2016, p. 6.

1.6.1 Entidades e relacionamentos

Os componentes da base de dados consistem em quatro tabelas. *Account* (conta), *branch* (ramo), *teller* (contadores) e *history* (história). As relações entre essas tabelas são definidos no diagrama de descrição lógica, sem implicações para implementação física, na Figura 1.7. Os resultados do *benchmark* dependem do volume de trabalho, requisitos específicos de aplicação, tipo do sistema e sua implementação. O desempenho do sistema varia através destes fatores.

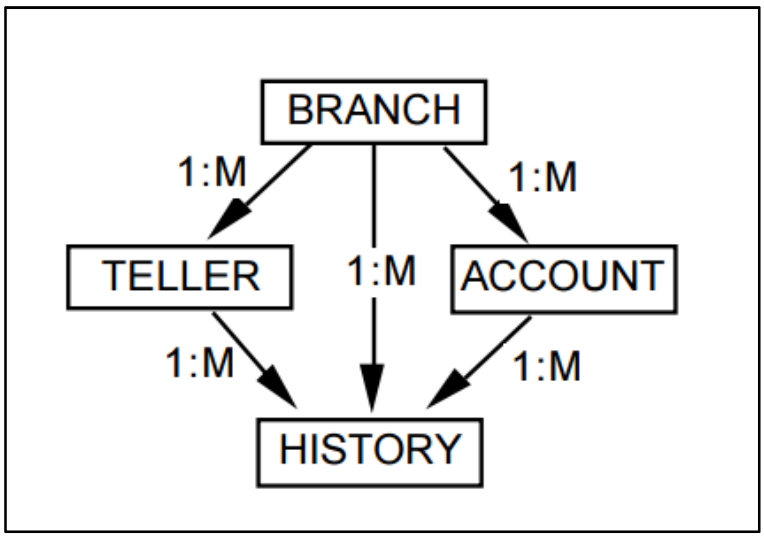

Figura 1.7 – Diagrama banco de dados TPC-B
Fonte: TPC, 2016, p. 13

As tecnologias descritas foram minuciosamente estudadas e apresentam características imprescindíveis para a realização desta monografia, sendo possível seu desenvolvimento. No próximo capítulo, será apresentado o desenvolvimento desta monografia.

2 BANCO DE DADOS DISTRIBUÍDOS

Este capítulo descreve alguns conceitos e tecnologias a cerca do funcionamento interno de bancos de dados distribuídos e conceitos aprofundados sobre as ferramentas utilizadas neste trabalho.

2.1 AMBIENTE DISTRIBUÍDO

De acordo com Özsu e Valduriez (2011), o banco de dados é administrado de modo centralizado por um computador e todas as suas requisições são encaminhadas para um sítio (site). Para alocar dados, são utilizados dois tipos de tecnologia: particionada (ou não replicada) e replicada. No esquema particionado, o banco de dados é dividido em um número de partições desmembradas nas quais cada uma pode ser colocada em um local diferente. Replicação pode ser totalmente replicada, na qual o banco de dados inteiro é armazenado em cada site; ou parcialmente replicada, cada partição do banco de dados é armazenado em mais de um ponto, mas não em todos eles. A figura 2.1 representa a configuração de um ambiente distribuído e seus dados dispostos dentre os pontos.

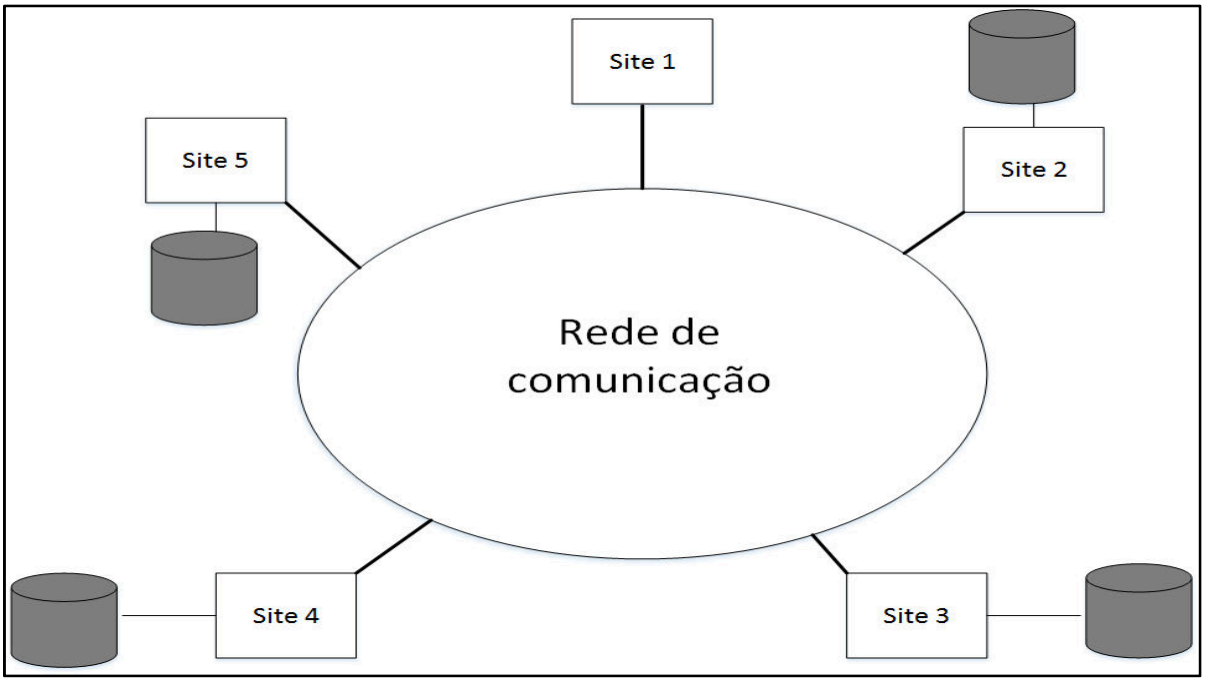

Figura 2.1 – Ambiente de banco de dados distribuído
Fonte: Adaptado de Özsu; Valduriez, 2011, p. 5.

A concepção de um sistema de computação distribuído envolve tomar decisões nas atribuições de dados e programas através de ambientes da rede de computador, assim como a concepção da própria rede. No caso de banco de dados distribuídos, a distribuição das aplicações envolve dois temas: a distribuição dos *softwares* de banco de dados distribuídos e a distribuição do programa de aplicação que é executado nele. (ÖSZU; VALDURIEZ, 2011).

As funções realizadas por um banco de dados distribuído são as camadas indicadas na figura 2.2 nas quais cada flecha indica a direção dos dados e o controle do fluxo. As camadas são interface, controle, compilação, execução, acesso de dados e administração da consistência. Abaixo são descritos detalhadamente cada função. (ÖSZU; VALDURIEZ, 2011).

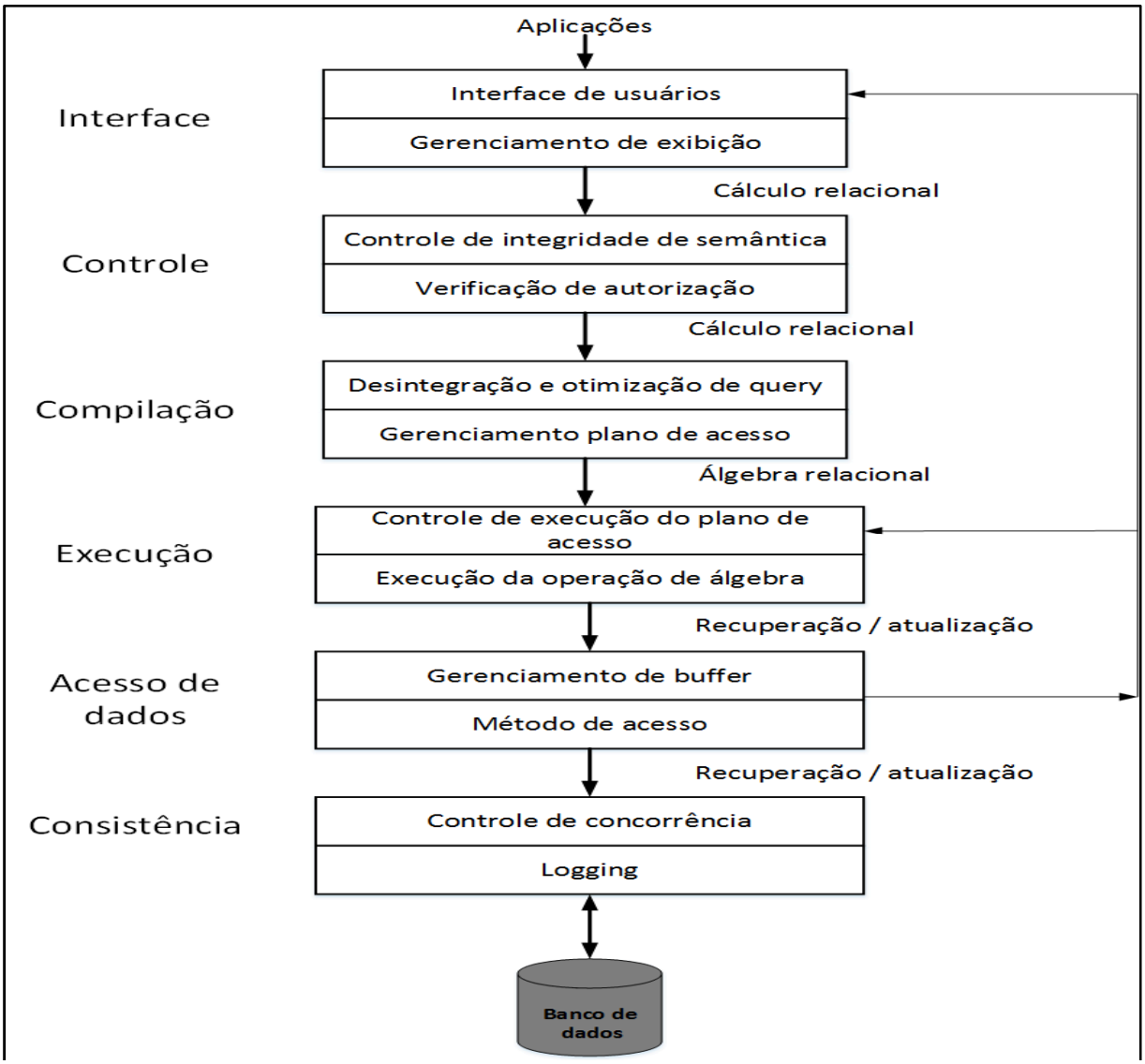

Figura 2.2 – Funções das camadas de um BDD.
Fonte: Adaptado de Öszu; Valduriez, 2011, p. 24.

- Camada de interface: consiste em traduzir a *query* do usuário a partir dos dados externos para os dados conceituais. A visão em um banco de dados relacional é uma relação virtual derivada das relações de base através das operações de álgebra relacional.
- Camada de controle: controla as consultas, adicionando predicados de integridade semântica e autorização de predicados. *Constraints* de Integridade semântica e autorizações são geralmente especificadas em uma linguagem declarativa. A saída de dados desta camada é uma consulta em linguagem de alto nível aceita pela interface.
- Camada de compilação ou processamento de consultas: é a camada que mapeia a consulta em uma sequência otimizada de operações de nível inferior (operações de álgebra), visando melhor performance. O resultado fica armazenado em um plano de acesso.
- Camada de execução: direciona a efetuação do plano de acesso gerado pela camada anterior, incluindo nela a administração de transações (*commit* e *restart*) e sincronização das operações de álgebra. Chama a camada de acessos por meio da recuperação e atualização de pedidos
- Camada de acesso de dados: administra a estrutura dos dados que implementa as filas e índices. Também administra os *buffers* armazenando em cachê os dados mais acessados frequentemente. Esta camada minimiza o acesso ao disco para obter ou escrever dados.
- Camada de consistência: permite transações, recuperação de sistema e mídia após falhas. Gerencia controle de concorrência e de registro para solicitações de atualização.

2.2 CLUSTERS

Clusters são um conjunto de nós de servidores independentes, interligados para compartilhar recursos e formar um único sistema. São modelos de computação paralela que oferecem uma alternativa mais em conta para supercomputadores ou multiprocessadores fortemente acoplados. Todos os nós de um *cluster* são homogêneos, geograficamente concentrados como um nó único, de leitura intensiva e tem uso efetivo em informática científica, recuperação de informação na web

(mecanismo de pesquisa do Google) e *data warehousing*. Os recursos compartilhados podem ser de *hardware* como em disco, ou *software*, como serviços de gerenciamento de dados. (ÖSZU; VALDURIEZ, 2011).

Segundo Böszörmeny e Scönig (2013), o *cluster* oferece serviços e recursos para o usuário. Desde que cada nó esteja executando uma instância do administrador da camada de *cluster*, qualquer serviço pode ser executado em qualquer nó. Os recursos podem ser de *standalone,cloned* ou *master-slave*. Apenas uma instância do recurso *standalone* pode ser executada a qualquer momento através do *cluster*. O recurso *cloned* funciona em sua maioria como o *standalone*, porém mais de uma instância pode ser executada através do cluster e eles funcionam independentes entre si. Recurso *master-slave* são geralmente relacionados ou conectados a cada um e eles são dependentes entre si. Neste tipo de serviço de *cluster*, é comum ocorrer o *fail-over*, quando um serviço ou um computador mostra comportamento irregular.

2.2.1 Arquitetura cluster em banco de dados

De acordo com Öszu e Valduriez (2011), a administração de dados paralela é feita por um SGBD independente, regidos por um *middleware* replicado em cada nó. Para melhorar seu desempenho e disponibilidade, os dados podem ser replicados em nós diferentes usando o SGBD local. Aplicações de clientes interagem com o *middleware*para submeter transações de banco de dados, *ad-hocqueries*, transações ou chamadas para *stored procedures*.

O *middleware* de um *cluster* de banco de dados possui várias camadas de *software*. O primeiro é o balanceador de carga de transação que aciona a execução de transações com o melhor nó. O melhor nó é definido como o nó que possui a carga mais leve. O administrador de replicação controla o acesso para dados replicados e assegura forte consistência de maneira que as transações que atualizam os dados replicados são executadas na mesma ordem em cada nó. Por fim, o processador de instruções controla a execução da consulta, a composição do resultado final e o balanceamento de carga. (ÖSZU; VALDURIEZ, 2011).

2.3 REPLICAÇÃO

Bancos de dados distribuídos são replicados, com o propósito de garantir disponibilidade do sistema, já que os itens de dados estarão acessíveis em múltiplos sites. Se algum falhar, outros estarão acessíveis a partir de outros locais. Também é necessária para garantir melhor desempenho, a replicação permite localizar os dados mais próximos de seu ponto de acesso, contribuindo para a redução do tempo de resposta. A replicação também assegura escalabilidade, já que os sistemas crescem geograficamente em termos de número de sítios. (ÖSZU; VALDURIEZ, 2011).

Segundo Öszu e Valduriez (2011), uma base de dados distribuída deve ser totalmente ou parcialmente replicada. Quando são parcialmente replicadas, o número de dados físicos para cada dado lógico pode variar, além da possiblidade da não replicação de alguns dados. Transações que acessam dados replicados precisam ser executados em múltiplos locais, sendo assim, transações globais.

2.3.1 Tipos de replicação

As replicações são classificados em dois tipos: síncronas ou assíncronas. No caso da replicação assíncrona, os dados podem ser replicados depois das transações serem enviadas para o mestre. O escravo nunca está a frente do mestre. No caso de escrita de dados, geralmente ficará um pouco atrás do mestre, cujo atraso é chamado de *lag*. Emreplicações síncronas, todo o sistema precisa garantir que os dados escritos por uma transação estarão no mínimo em dois servidores no mesmo tempo que a transação é enviada. O escravo não possui um *lag* atrás do mestre. (BÖSZÖRMENY; SCHÖNIG,2013). A figura 2.3 exemplifica como as replicações assíncronas e síncronas funcionam eretrata suas diferenças referidas no texto. De acordo com Böszörmeny e Scönig (2013), na replicação assíncrona, há uma janela (*lag*) durante cada dado que pode ser perdido. O tamanho deste *lag* pode ser curto, como alguns milissegundos ou pode ser grande, durando minutos, horas ou dias. Se for imprescindível não perder dados, a melhor opção de replicação a ser escolhida será a síncrona.

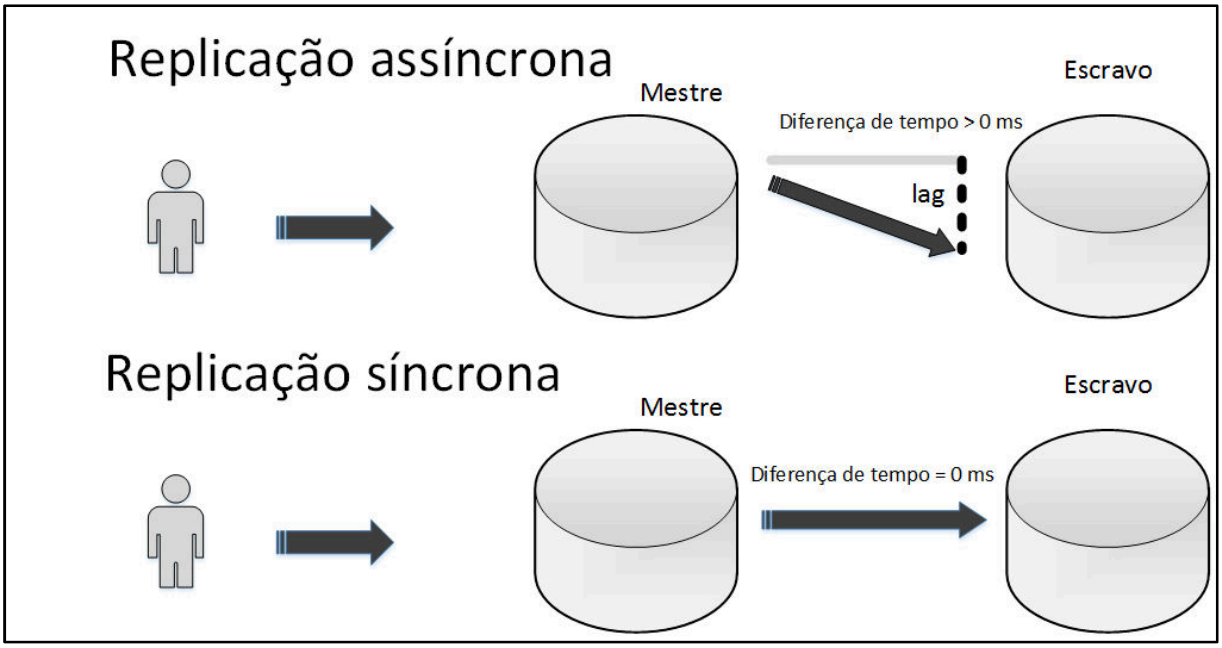

Figura 2.3 – Replicações síncronas e assíncronas.
Fonte: Adaptado de Böszörmeny; Scönig, 2013, p. 11.

2.3.2 Fragmentação e replicação

É preciso fazer com que todo o sistema seja mais confiável e a prova de falhas quando os dados de cada servidor ou partição são segmentados em pedaços úteis. Quanto mais *servers* (servidores) em rede, maior a chance de um deles falhar e ficar indisponível por alguma razão. Para garantir máximo rendimento e disponibilidade, é possível transformar esta arquitetura em uma redundância. Um servidor nunca é o bastante para prover alta disponibilidade, todos os sistemas precisam de um sistema de *backup*, que deve assumir em caso de emergência. Mas apenas dividir um conjunto de dados não garante disponibilidade já que mais servidores poderão falhar. Para solucionar o problema, é possível adicionar réplicas a cada uma das partições. Cada partição é uma instância de banco de dados PostgreSQL separado e cada uma de suas instâncias tem sua própria réplica (BÖSZÖRMENY; SCHÖNIG, 2013). Na Figura 2.4 é representada a arquitetura citada acima.

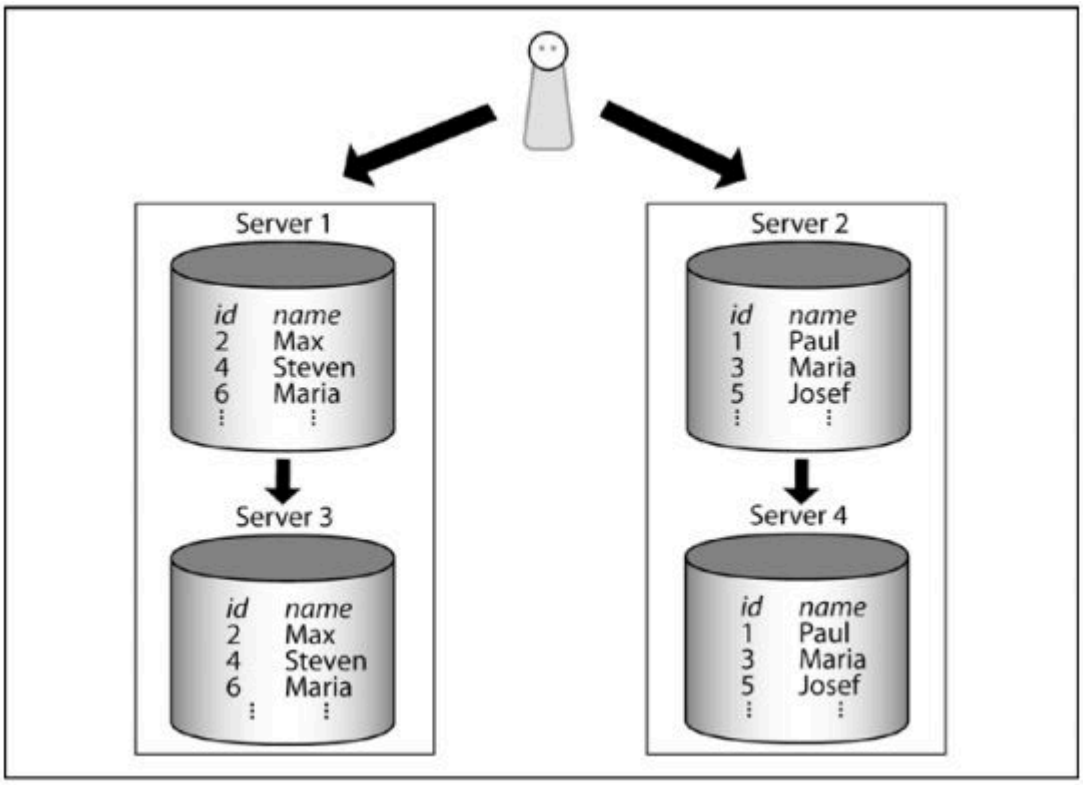

Figura 2.4 – Instâncias e réplicas
Fonte: Böszörmeny; Scönig, 2013, p. 23

2.4 PGPOOL

Trata-se de um *middleware* que trabalha entre os servidores e o cliente de banco de dados PostgreSQL. Esta ferramenta salva conexões para os servidores e as usa sempre que uma nova conexão com as mesmas propriedades chegam, reduzindo a sobrecarga de conexão e melhorando o desempenho do sistema em geral. Também gerencia múltiplos servidores do PostgreSQL, usando a função de replicação que permite a criação de um *backup* de dois ou mais discos físicos, que continuam o serviço sem parar a execução dos servidores, caso o disco falhe. Garante o balanceamento de carga ao reduzi-la em cada servidor do banco de dados através da distribuição de consultas *SELECT* entre vários servidores, melhorando o rendimento geral do sistema. Esta função opera melhor quando diversos usuários executam variadas consultas ao mesmo tempo. (PGPOOL, 2015). Sua arquitetura é ilustrada na Figura 2.5.

Conforme Smith (2010), seu principal objetivo não é somente o *pool* de conexão, ele fornece o balanceamento de carga e a replicação. Suporta algumas configurações de consulta paralela a qual cada uma pode ser fragmentada e

espalhada pelos nós e cada nó possui uma cópia da informação que está sendo requisitada. O *pool* no nome Pgpool é referido porque lida com múltiplos servidores, sendo que o programa serve como um servidor *proxy* entre os clientes e um determinado número de banco de dados.

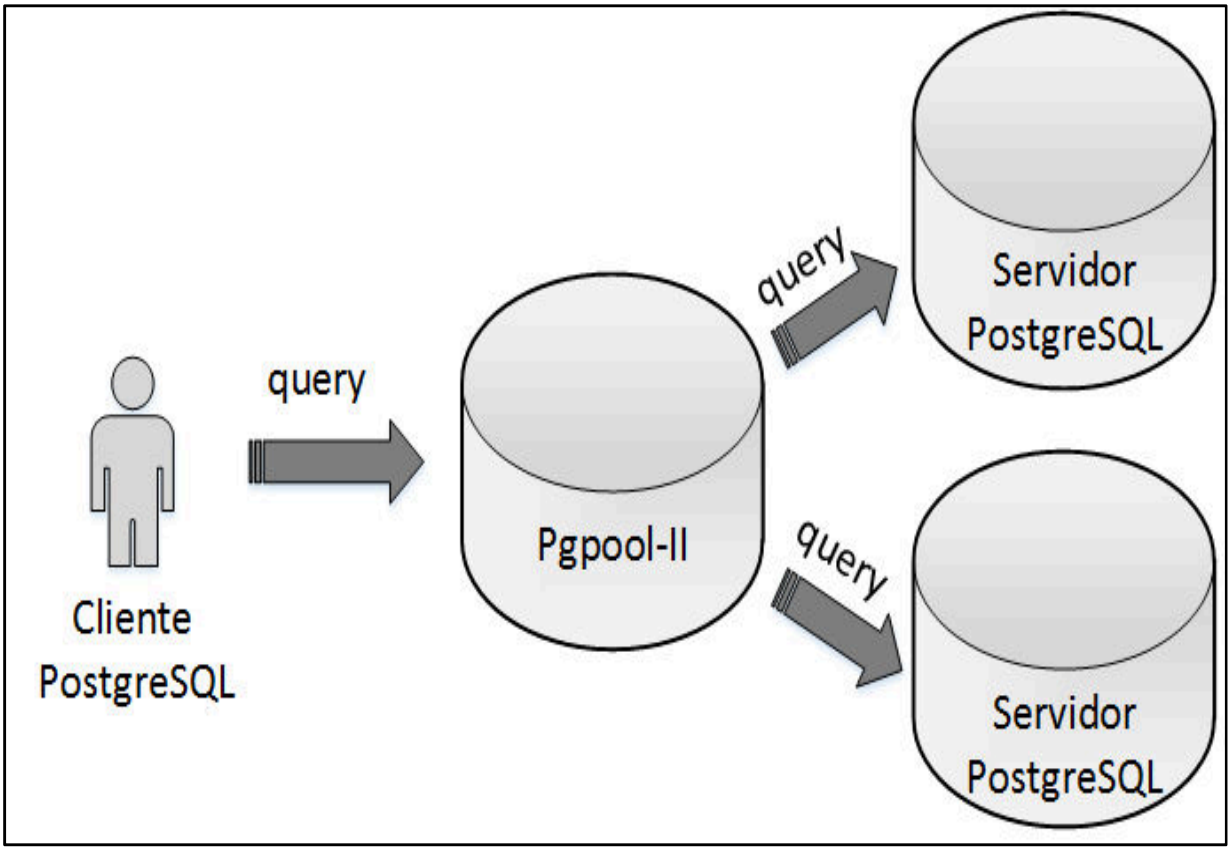

Figura 2.5 – Arquitetura do Pgpool
Fonte: Adaptado de Böszörmeny; Scönig, 2013, p. 148

Quando o Pgpool é iniciado, um processo chamado *parent* (pai) é iniciado. Este processo se dividirá e criará os processos *child* (filho), responsáveis por atender as requisições para os usuários finais e interagir com cada um dos nós dos bancos de dados. Cada processo *child* lidará com um grupo de conexões de *pool*, estratégia que reduz o número de requisições de autenticação para o PostgreSQL. A infraestrutura PCP é necessária para lidar com a configuração e administração. É necessário um grupo de nós de banco de dados PostgreSQL como armazenamento de *backend*. Usuários finais não irão se conectar a esses nós diretamente, somente através do Pgpool (BÖSZÖRMENY; SCHÖNIG, 2013). A Figura 2.6 detalha o processo descrito acima.

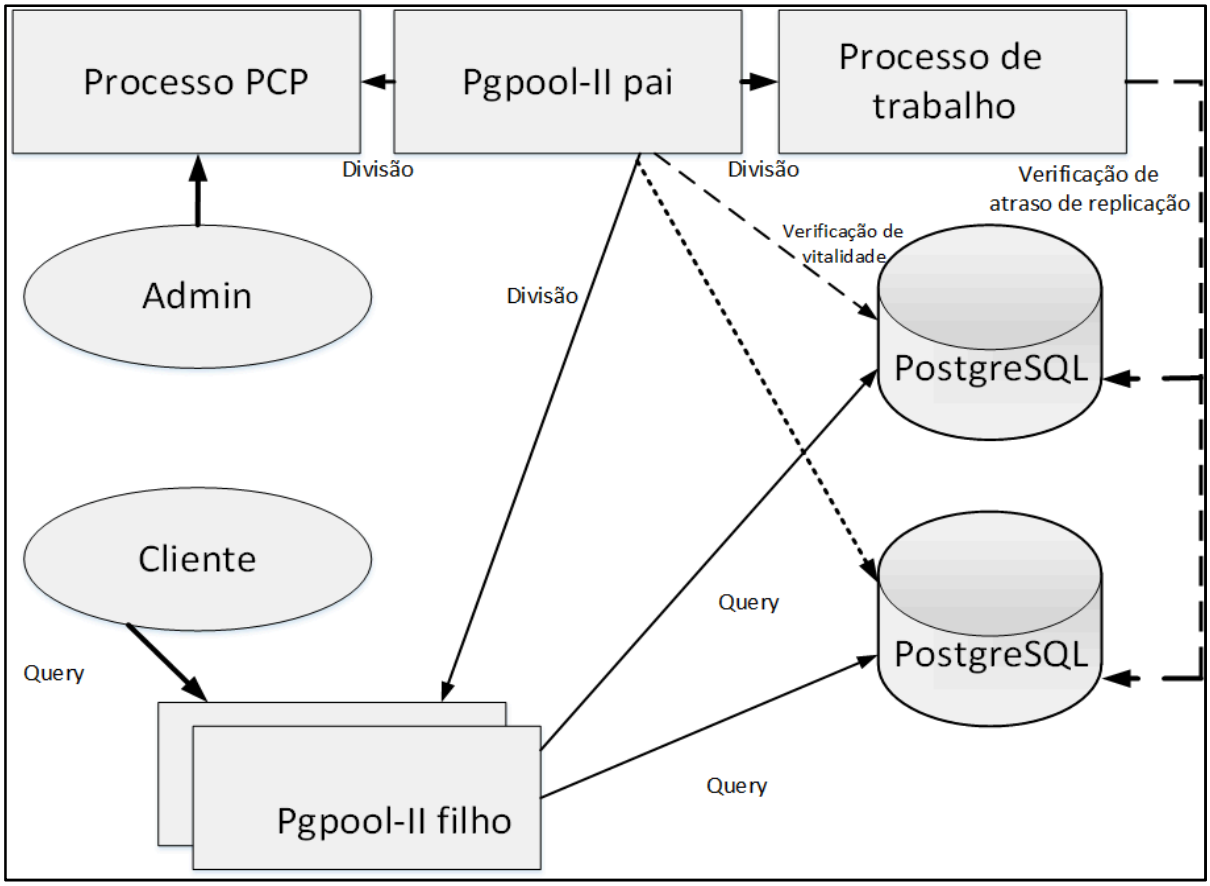

Figura 2.6 – Descrição e detalhes da execução
Fonte: Adaptado de Böszörmeny; Scönig, 2013, p. 148

O Pgpool fornece um conjunto de ferramentas úteis de recursos para banco de dados em escala e permite seu funcionamento de vários modos que incluem conexão de *pool*, replicação, *master*/*slave* e consulta paralela. Alguns modos podem ser combinados para ficarem ativos ao mesmo tempo.

Conforme visto, este capítulo teve o propósito de aprofundar os conhecimentos sobre banco de dados distribuídos, suas funcionalidades, as possibilidades de um *cluster* e a importância da replicação nas bases de dados distribuídas. Também explanou detalhadamente a principal ferramenta que torna estas configurações possíveis, o Pgpool, levando em consideração como é feita sua execução e sua arquitetura. No próximo capítulo será abordado os detalhes sobre o desenvolvimento do trabalho para a realização dos testes de *benchmark*.

3 DESENVOLVIMENTO DO TRABALHO

Este capítulo tem por objetivo especificar como as tecnologias citadas anteriormente contribuíram para o desenvolvimento deste trabalho. Será detalhado as principais configurações do ambiente de testes, os aspectos da metodologia e os resultados finais esperados.

3.1 AMBIENTE DE TESTES

O ambiente de testes desta monografia foi projetado de acordo com a especificação padrão da ferramenta *Pgbench* que é baseada no protocolo TPC-B, visando a mediçãodo número total de operações simultâneas que o sistema pode suportar, considerando o valor máximo de desempenho e taxas de transferência a serem estudadas. Cada transação aguarda sua conclusão e em seguida envia outra. As transações são enviadas pelos programas que estão sendo executados ao mesmo tempo, de acordo com TPC (2015).

Os ambientes utilizados neste projeto foram divididos entre ambiente físico e virtual, através da virtualização, conceituando-se como uma técnica que permite a execução de múltiplos sistemas operacionais e aplicações sobre uma máquina física, por meio de um ambiente próprio, disponibilizando um sistema operacional, aplicativos e serviços de rede (CARISSIMI, 2008). O uso da virtualização foi preterido também para garantir confiabilidade, a fim de não perder dados e configurações entre os locais citados anteriormente.

Como parte do ambiente físico, foi utilizado um *notebook* da marca Lenovo, modelo G400s – 80AC0001BR-2099. Processador Intel Core i5 – 3230M, CPU de 2.60 GHz. Memória RAM de 4.0 GB e HD de 1 TB, operando com sistema operacional Windows 8.1, de 64 bits. Neste *notebook*, foi instalado o *software*de banco de dados *pgAdmin* III versão 1.18.1 para administrar o ambiente de banco de dados distribuídos virtualizados e que foram dispostos através de seus endereços de IP através de permissões autorizadas no arquivo pool_hba.conf conforme Figura 3.1.

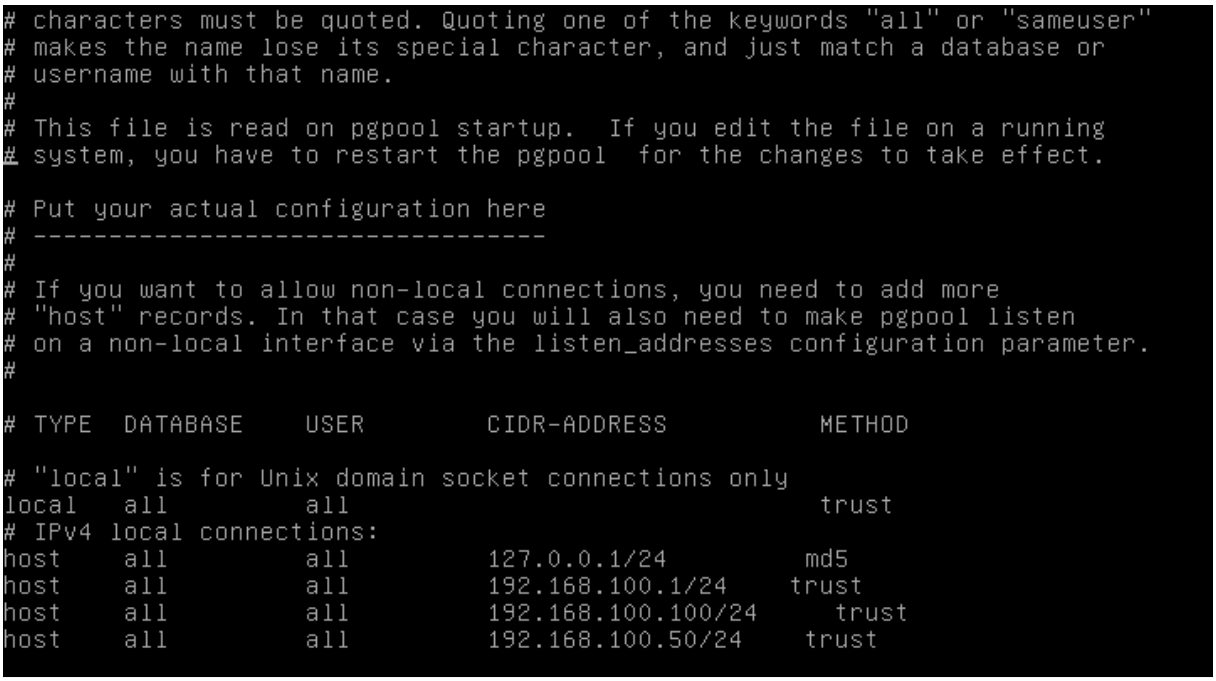

Figura 3.1 – Arquivo pool_hba.conf
Fonte: Elaborado pela autora, 2016.

O arquivo especificado na figura 3.1 controla quais *hosts* possuem permissão para realizar a conexão com o banco de dados, como a autenticação dos clientes é feita, nome dos usuários permitidos e os bancos de dados que podem acessá-lo. Para a realização dos testes, foi imprescindível que neste arquivo constasse a especificação de todos os *hosts*, banco de dados e usuários selecionados para realizar o acesso, além dedefinir a faixa de IP utilizada para cada máquina que possui permissão para o controle local e remoto. É importante destacar também que a conexão de servidores e a replicação destes foi permitida pela ferramenta Pgpool.

A partir destes requisitos, o ambiente virtual foi criado e composto por três máquinas virtuais, todas utilizando o sistema operacional Debian, de 32 *bits*, em sua versão 7.6.0. Para cada uma, foi disposto 512 MB de memória principal, e para conexão de rede, as placas foram configuradas em modo b*ridge*. O modo *bridge* permite que a máquina virtual conecte-se diretamente à rede da máquina física, através de uma ponte com a interface do *host*. Para a placa em modo *bridge*, a configuração é Intel(R) 82579LM *GigabitNetworkConnection*, seu tipo de placa é Intel PRO/1000 MT *Desktop* (82540EM) com endereço MAC 0800276244F7. Dois adaptadores de rede foram utilizados a partir de suas especificações, ambos com a mesma configuração, adaptador 1 e 2 Intel PRO/1000 MT Desktop (Placa em modo *Bridge*, Intel(R) 82579LM *Gigabit Network Connection*.

O ambiente físico foi composto por três máquinas utilizando o sistema operacional *Windows*7*Professional* 2009 com *ServicePack* 1, modelo HP Compaq 8200 Elite SFF PC64 *bits* e arquitetura ACPI x64-*based* PC. Quando ao processador, Intel(R) *Core* (TM) i5-2400 CPU @ 3.10GHz 3.10 GHz e possui memória instalada de (RAM) 4,00 GB - utilizável 3,89 GB. Em cada uma dessas máquinas foi instalado o *software* Virtual Box em sua versão 5.0.6. Para conectar os *hosts* e a habilitar a comunicação das máquinas físicas e virtuais, foi utilizado o *switch* da marca Compex, modelo PS 2216, que não possui configuração gerenciável; possui 16 portas no qual 4 foram utilizadas e largura de banda 100 *megabits*.

O principal *host* foi chamado de *Pooler* no qual foi instalado o programa *PgPool*,*software* intermediário que realiza a conexão entre o PostgreSQL e o servidor. Os demais foram denominados *Node*1 e *Node*2, conforme a Figura 3.2, instalado em ambos o banco de dados PostgreSQL.

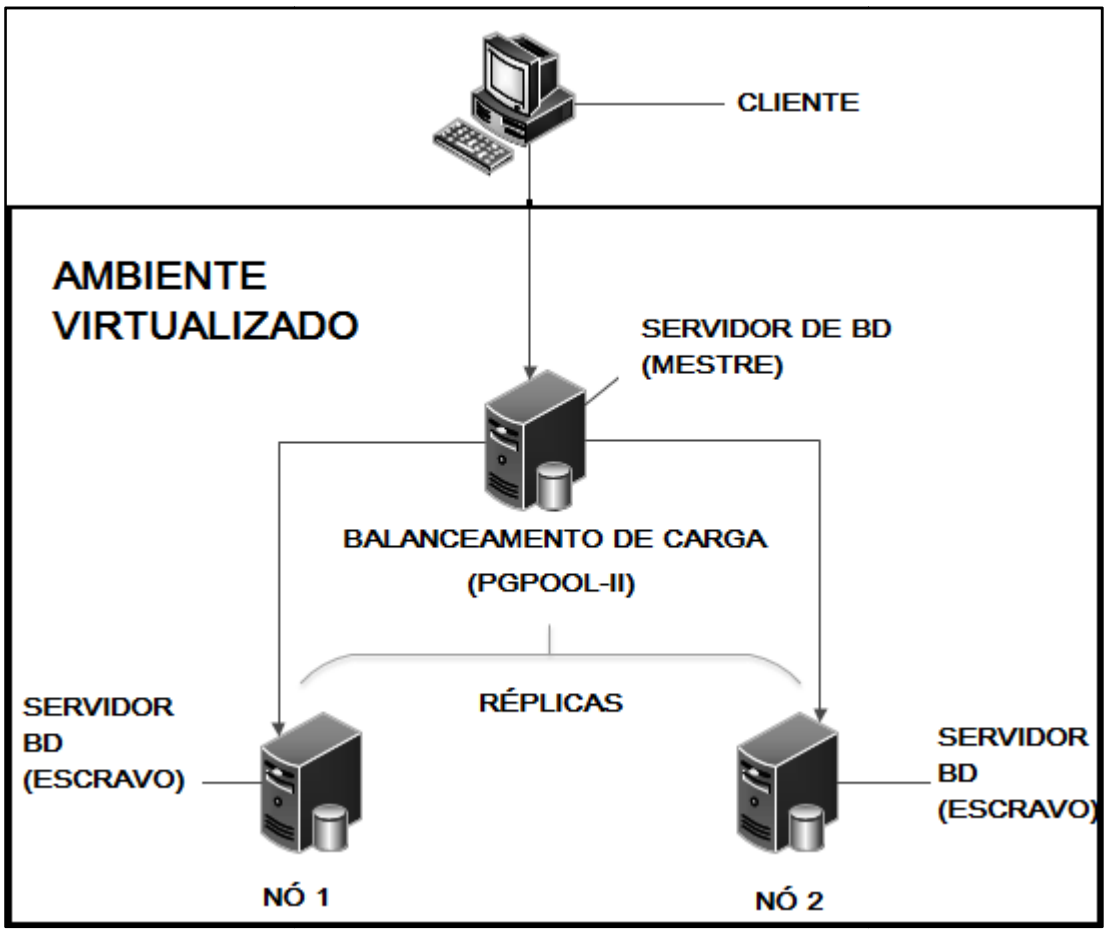

Figura 3.2 – Disposição do ambiente de testes
Fonte: Elaborado pela autora, 2016.

O ambiente de testes foi configurado para que o cliente possa realizar requisições que são processadas pelo *Pooler*, que posteriormente replicará essas informações para o *Node* 1 e o *Node 2*, respectivamente.

Para o endereçamento de IP, foi utilizada a classe 192.168.100.0/24. A tabela 3.1 indica a distribuição de endereços IP das máquinas virtuais e físicas.

Tabela 3.1 – Distribuição de endereços IP.

EQUIPAMENTO	**IP**
POOLER	192.168.100.1/24
NODE 1	192.168.100.2/24
NODE 2	192.168.100.3/24
MÁQUINA FÍSICA 1	192.168.100.50
MÁQUINA FÍSICA 2	192.158.100.51
MÁQUINA FÍSICA 3	192.168.100.52

Fonte: Elaborado pela autora, 2015

3.2 METODOLOGIA

O principal objetivo deste teste é gerar consultas com grandes volumes de dados através do *benchmark Pgbench*, que reproduz atividade intensiva do cliente para o servidor de banco de dados do sistema virtualizado e calcula a média das transações feitas pelo sistema. O resultado final de cada execução de testes deste*benchmark*são dados em seis linhas, cada uma especificando o número de transações completadas e as últimas duas linhas mostram o número de transações por segundo, a qual possui duas opções: incluindo e excluindo o tempo que é necessário para iniciar as sessões de banco de dados conforme a Figura 3.3 que ilustra o método de inicialização do *Pgbench* –i e juntamente a atribuição o nome do usuário do banco de dados que foi criado para concretização dos testes, *postgres*.

Para a execução do *benchmark* mencionado, várias opções de comandos são disponibilizadas através do manual do *Pgbench* as quais devem ser escolhidas conforme a necessidade de cada teste e suas finalidades.

```
C:\Program Files\PostgreSQL\9.4\bin>pgbench.exe -h 192.168.100.1 -i -U postgres

NOTA:  tabela "pgbench_history" n+úo existe, ignorando
NOTA:  tabela "pgbench_tellers" n+úo existe, ignorando
NOTA:  tabela "pgbench_accounts" n+úo existe, ignorando
NOTA:  tabela "pgbench_branches" n+úo existe, ignorando
creating tables...
100000 of 100000 tuples (100%) done (elapsed 3.47 s, remaining 0.00 s).
vacuum...
set primary keys...
done.
```

Figura 3.3 – Inicialização Pgbench
Fonte: Elaborado pela autora, 2016.

Foi utilizado para este trabalho, a opção –c, que especifica o número de clientes simulados no banco de dados. A opção –h, que é usado para classificar o *hostname*, ou seja, o endereço de IP que se encontra o *host* que irá disparar e replicar as requisições para os outros nós. Posteriormente, é fundamental definir as variáveis dos testes. Para cada teste são dispostas as opções que irão depender do resultado que o usuário definiu anteriormente. Foram usadas as variáveis –t que classifica o número de transações que cada cliente executa, –j são as *threads*, que foram definidas de acordo com o mesmo número de clientes utilizados em cada teste e para finalizar, o parâmetro –r que relatou a latência média por tempo de execução de cada teste de *benchmark.*

O *Pgbench* realiza os testes de modo a popular as tabelas referidas na Tabela 3.2 do banco de dados, preenchendo-as com seus valores pré determinados, através de opções disponibilizadas em sua documentação. O modelo de dados utilizado é disposto pelo próprio *benchmark*, que inclui um conjunto de agências bancárias, cada um dos quais tem um número de caixas e contas nos quais o *benchmark* executa uma sequencia de comandos que envolvem os três comandos de transações de um banco de dados, sendo eles o *select*, *update* e o *insert.* O comando *insert*, insere linhas dentro de uma tabela, o *select* recupera linhas e valores das tabelas e por último, o comando *update*, que altera os valores das colunas especificadas nas linhas que constam a condição detalhada na instrução dos comandos SQL.

Tabela 3.2 – Tabela inicial Pgbench

TABELAS	# DE TUPLAS
Branches	1
Tellers	10
Accounts	100000
History	0

Fonte: PostgreSQL, 2015.

Quanto aos experimentos deste trabalho, foram definidos através de dois fatores principais: número de clientes e número de transações, conforme lista a figura 3.4

Planejamento de Experimentos									
Fator	Nível 1	Nível 2	Nível 3	Nível 4	Nível 5	Nível 6	Nível 7	Nível 8	Nível 9
Núm. Clientes Virtuais	50	55	60	65	70	75	80	85	90
Núm. De Transações	100	1000							

Figura 3.4 – Planejamento de experimentos
Fonte: Elaborado pela autora, 2016.

Será executado um planejamento de experimentos do tipo fatorial simples com 9 níveis de testes e a variável de resposta obtida para cada repetição será o tempo de retorno de cada requisição feita ao banco de dados distribuído mensurado em milisegundos. Os experimentos serão realizados 10 vezes de modo a obter os valores médios de todos os níveis especificados. As repetições auxiliarão na análise do comportamento do sistema de banco de dados distribuído quanto à variância no tempo de resposta.

3.3 RESULTADOS ESPERADOS

Espera-se que a carga de trabalho aplicada ao ambiente virtualizado de banco de dados distribuídos seja replicada e processada aos demais banco de dados de modo a estimular a variável do tempo de resposta. Também será analisado o comportamento do sistema distribuído quanto à perda de pacotes referentes aos vários tipos de consultas enviadas ao sistema pelo software de *benchmark*. Este trabalho, por meio da ferramenta Pgbench espera ocasionar

situações de picos de demanda, visando a simulação de situações de uso do sistema sobre condições extremas.

No próximo capítulo, serão apresentados os resultados dos testes realizados através de gráficos e tabelas, além de demonstrar como foram as variações dos fatores expostos neste capítulo.

4 RESULTADOS E DISCUSSÃO

Este presente capítulo tem por intuito demonstrar os valores obtidos após a realização dos testes descritos no capítulo três, através da disposição de gráficos, suas respectivas analises e resultados dos valores retornados após a execução do *benchmark*. Deste modo, serão apresentados os principais detalhes e discussão acerca do principal objetivo desta monografia.

4.1 RESULTADOS EXPERIMENTAIS

Conforme análise do gráfico de valores médios da variável de resposta de Transações por Segundo (TPS) ilustraado na Figura 4.1 é possível perceber a sinuosidade dos valores obtidos como resultado. Esse comportamento pode ser explicado por meio de algumas verificações que foram feitas durante a execução dos experimentos, embora as informações acerca de alguns aspectos do sistema computacional de banco de dados distribuído não tenham sido gravadas em arquivos de saída ou mesmo em *logs*.

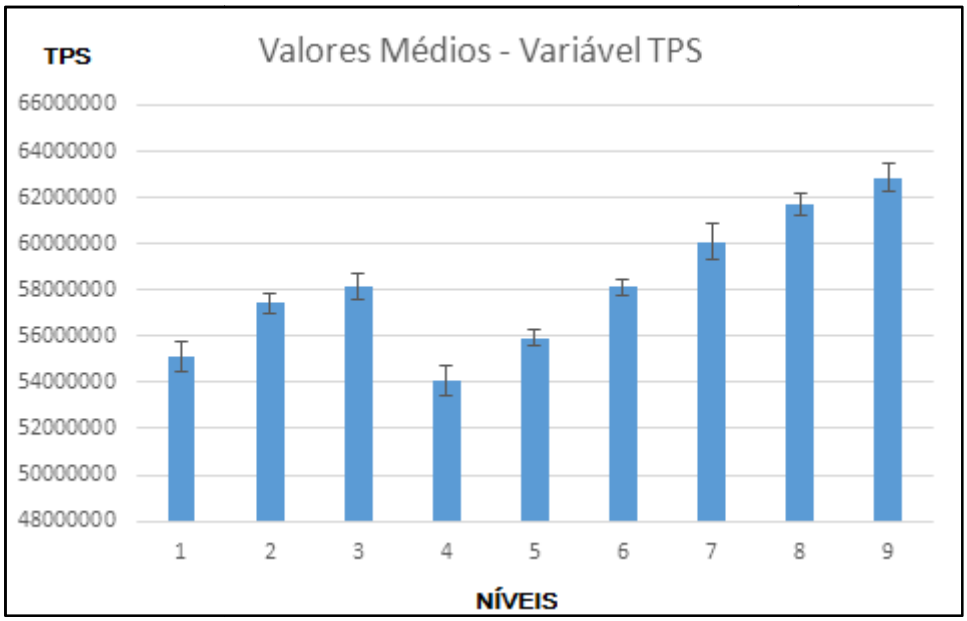

Figura 4.1 – Variáveis TPS 100 transações.
Fonte: Elaborado pela autora, 2016.

Na bateria de experimentos de 1 a 9, o número de usuários virtuais é aumentado de 5 em 5 e o número de transações de banco de dados executada permanece fixo em 100 unidades transacionais. É oportuno citar que o *PgBench*, a aplicação de *benchmark* usada para os testes, executa um conjunto de operações simples de banco de dados, constituídas por 5 *SELECT*, 5 *UPDATE* e 5 *INSERT*. Além disso, cada conjunto das 15 operações supracitadas é considerado 1 transação completa do *benchmark*, assim, para executar, por exemplo, 100 transações, serão executados os referidos 15 comandos por 100 vezes, causando um impacto maior na carga de trabalho imposta aos servidores de banco de dados.

Nos experimentos 1, 2 e 3 é possível notar o aumento das transações a despeito do aumento de usuários virtuais cujos números foram, respectivamente 50, 55 e 60. Foi possível notar que a carga de uso de CPU e memória das VMs e, portanto, dos hosts físicos considerados no ambiente de testes, foi relativamente baixa nos primeiros experimentos. No experimento 1, por exemplo, o valor médio da carga de CPU foi de cerca de 10%, mostrando que o sistema trabalhava com folga de potência. Contudo, ao chegar no experimento 4, com 65 usuários e 100 transações é possível notar o decréscimo no desempenho de TPS, mostrando que, possivelmente, na ocasião desse experimento, o sistema usou os recursos que havia alocado até então para o atendimento a clientes e mesmo assim não foi possível entregar melhor desempenho.

Notoriamente, na sistemática de arquitetura de BDD considerada, há uma máquina usada como gerenciador das demais máquinas do BDD, considerado como cabeça do cluster ou como *broker* (intermediador) das requisições. Assim, embora não existam dados armazenados para confirmar esta hipótese, considera-se que no experimento 4 o BDD não alocou recursos adicionais nas máquinas que dispunha para atendimento da carga de trabalho. É possível considerar essa premissa como verdadeira, pois, nos experimentos de 5 a 9, os quais possuíam o mesmo número de transações, porém, número crescente de clientes, o desempenho da variável de resposta TPS foi crescente, significando que, de alguma forma, a máquina coordenadora do sistema de BDD conseguiu alocar mais recursos computacionais para atender à demanda crescente e entregou melhor desempenho aos clientes virtuais.

No cenário de experimentação onde foram consideradas 100 transações por cliente virtual, é possível afirmar que a variabilidade dos resultados dos

experimentos (vide linha representada acima de cada barra azul no gráfico da figura 4.1) foi considerada baixa e aceitável, apresentando um baixo nível de ruído (variância) de um experimento para outro. Essa característica possibilidade considerar os resultados como verdadeiros e que a sistemática de testes aplicada foi conduzida corretamente.

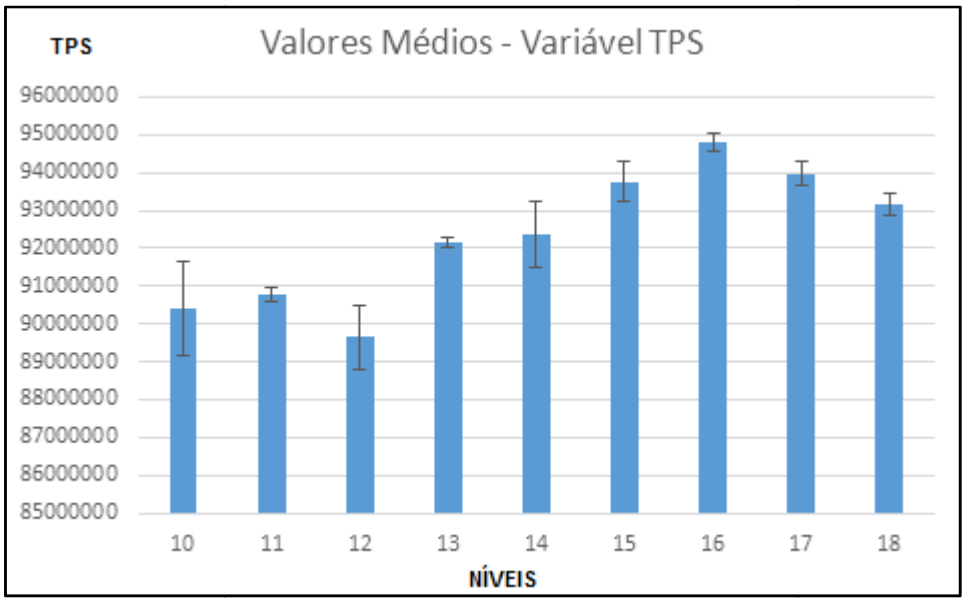

Figura 4.2 – Variáveis TPS 1000 transações.
Fonte: Elaborado pela autora, 2016.

Na bateria de experimentos de número 10 a 18 todos os experimentos consideraram o número fixo de 1000 transações por cliente e o número de cliente virtuais foi aumentando de 5 em 5 unidades partindo do valor 50 clientes no experimento 10 e chegando ao valor de 90 clientes no experimento 18.

Em relação ao cenário de experimentos com 100 transações por cliente é possível notar uma diferença no comportamento da variável de Transações por Segundo (TPS). Isso se justifica pelo aumento considerável no número de transações para 1000 unidades transacionais por cliente, valor este 10 vezes maior do que o considerado para o primeiro cenário experimental.

No experimento número 10 é possível notar um decréscimo em desempenho da variável de resposta se comparado ao experimento 9 (90 clientes e 100 transações), mostrando o impacto das transações no desempenho do sistema de BDD. É notável também a alta variabilidade encontrada no resultado deste experimento (vide linha representada acima de cada barra azul no gráfico da figura

4.2). Na sequência no experimento 11 nota-se o aumento do valor da variável TPS, indicando ganhos em desempenho, em seguida obteve-se no experimento 12 um resultado pior que no experimento 10 e então, a partir do experimento 13 até o 16 houve melhora considerável no desempenho, chegando ao pico máximo atingido em todos os experimentos: quase 95 milhões de transações efetivadas no experimento 16. A partir do experimento 17 houve declínio no desempenho da variável TPS.

O experimento 12 teve um comportamento bastante anômalo em relação aos demais, sendo que, para este resultado em específico, não se tem uma justificativa com informações de monitoramento que possam prover, com exatidão, a real causa de tamanha perda de desempenho. Fato interessante ocorrido nos dois cenários experimentais (experimentos com 100 transações e experimentos com 1000 transações) é que em ambos os cenários, quando do uso de usuários virtuais com valores entre 60 e 65 há essa queda no desempenho.

Nos experimentos de 13 a 16, infere-se que esteja sendo feita alocação de recursos para o sistema de BDD de modo a melhorar o desempenho para os clientes virtuais. A queda de desempenho obtida nos experimentos 17 e 18 pode ser justificada por meio de duas hipóteses: no experimento 16 o sistema já estava alocando o máximo de recursos computacionais (compreendendo as 3 VMs com o sistema de BDD) e a partir dos experimentos 17 e 18, devido ao maior número de clientes e, consequentemente, de transações o desempenho decai por não haver como alocar mais capacidade computacional; outra hipótese seria que o comportamento dos experimentos 17 e 18 pode ser homônimo ao experimento 12 e que, se houvessem mais experimentos com a continuidade do aumento dos valores dos fatores, o sistema ainda iria continuar aumentando o desempenho.

É oportuno frisar que, embora nos experimentos 10, 12, 14 e 15 a variabilidade tenha sido maior, os valores obtidos como média são considerados aceitáveis, partindo ponto que a variabilidade obtida não atinge 50% do valor da média da variável TPS para cada um dos referidos experimentos, portanto, o segundo cenário também é considerado correto e verdadeiro do ponto de vista de avaliação de desempenho do sistema alvo.

Neste capítulo foram apresentadas as considerações finais a cerca desta monografia que visou principalmente medir o desempenho e replicação de bancos de dados distribuídos através de uma ferramenta própria para análise de *benchmark* além de atestar que foi concluída a análise e experimentação expostos neste

trabalho. O próximo tópico deste trabalho detalhará a conclusão de todos os experimentos e resultados obtidos através dos testes realizados.

CONCLUSÃO

Este trabalho apresentou um estudo sobre sistemas de banco de dados distribuídos homogêneos, utilizando virtualização por meio de *VirtualBox*, e um ambiente de configuração homogênea constituído por um BDD de 3 máquinas. O sistema operacional considerado para as VMs foi o Linux Debian32 *bit* versão 7.6.0 e o *benchmark* utilizado nos experimentos foi o *PgBench*. O SGBD utilizado foi o PostgreSQL versão 9.4.5. O planejamento experimental executado foi o de fatorial completo totalizando 18 experimentos na combinação dos valores dos fatores considerados. Cada experimento foi repetido 10 vezes e, posteriormente, foram calculados, a média aritmética, o desvio padrão e o intervalo de confiança para cada um dos experimentos. Uma análise estatística foi efetuada com a demonstração gráfica dos resultados.

É possível concluir, por meio dos resultados obtidos e pela avaliação de desempenho apresentada, que um sistema de BDD oferece uma complexa relação entre o conjunto de recursos computacionais atrelados ao mesmo e o real desempenho entregue em tempo de execução.

A variável de reposta considerada, isto é, Transações por Segundo (TPS), servem como indicador para a verificação do volume de atendimento que o BDD está executando a cada unidade de tempo. Quanto maior seu valor, possivelmente será melhor o desempenho, pois, pode-se entender que o BDD está operando com maior eficiência no atendimento aos clientes e da carga de trabalho manifestada pelos mesmos.

Conforme já justificado no capítulo de discussão dos resultados experimentais, houve várias ocasiões de oscilação no desempenho da variável de resposta, conforme os valores dos fatores eram gradualmente aumentados. Essa oscilação, embora seja possível ter uma ideia de sua motivação, não foi possível ser diagnosticada com exatidão, pois, as informações de monitoramento que estavam disponíveis eram apenas aquelas apresentadas no relatório de saída do *benchmark PgBench* e as informações de tempo real dos computadores que compunham o BDD (as quais, embora observadas, não foram armazenadas). Assim, dada a importância de tal análise, considera-se todos os pontos de justificativa acerca dos

comportamentos anômalos das análises deste trabalho como inferências sem comprovação das vias de fato.

Por fim, é possível afirmar que um sistema de BDD, por meio do uso de um *benchmark* simples, tal como é o *PgBench*, pode apresentar diversas situação de entrega de desempenho, relacionada a variável de transações por segundo, satisfatórias, processando a grande maioria das transações e aproveitando a infraestrutura montada para o propósito da salvaguarda de dados.

TRABALHOS FUTUROS

A partir da análise crítica dos resultados e conclusões do presente trabalho, que apesar de encontradas dificuldades durante seu desenvolvimento,foi possível realizar e cumprir os requisitos de análise aqui apresentadas.A partir disso, pode-se apontar os seguintes itens como trabalhos futuros que podem ser desenvolvidos como complemento do estudo de sistemas de banco de dados distribuídos discutidos nesta monografia.

- Ampliar a tabela de experimentos para 12 níveis no fator clientes virtuais considerados;
- Acrescentar na tabela de experimentos um nível adicional de 5000 transações para o fator número de transações;
- Acoplar, a cada uma das VMs, um *software* de monitoramento e gravação das informações de desempenho dos recursos computacionais (CPU, memória, leituras/escritas em disco, utilização de interface de rede, etc) de modo que, após a finalização dos experimentos, seja possível cruzar as informações da variável de resposta TPS com as informações de desempenho de tempo real das máquinas, possibilitando saber o que ocorre, exatamente, com o equipamento em tempo de serviço;
- Acoplar ou configurar o BDD baseado em PostgreSQL de modo que possibilite a verificação da porcentagem de uso das máquinas componentes do BDD em tempo real, permitindo visualizar como o nó mestre (cabeça do *cluster* ou *broker* do sistema) está efetuando a distribuição da carga de trabalho para seus páreas;
- Acrescentar outras variáveis de resposta para análise, tais como tempo de resposta das transações, porcentagem de transações atendidas e porcentagem de transações não atendidas, entre outras possibilidades.

REFERÊNCIAS BIBLIOGRÁFICAS

BÖSZÖRMENYI, Z.; SCHÖNIG H. J. **PostgreSQL Replication**. Birmingham: Pack Publishing Ltd., 2013.

CASANOVA, M. A.; MOURA, A. V. **Princípios de sistemas de gerência de banco de dados distribuídos**. Campinas, 1999.

COULOURIS G,; DOLLIMORE J,; KINDBERG T,. **Sistemas distribuídos**. Tradução João Tortello. 4. ed.Porto Alegre: Bookman, 2007.

DRAKE J.D.; WORSLEY J.C. **PostgreSQL prático**.[s.l.]O'Reilly Media, 2002.

ELMASRI R.; NAVATHE S.B. **Sistemas de banco de dados**. Tradução Daniel Vieira. 6. ed. São Paulo: Pearson Addison Wesley, 2011.

MANNINO M. V. **Projeto, desenvolvimento de aplicações e administração de banco de dados**. Tradução Beth Honorato, Diana Prairo Heller, Lizandra de Moura Guerras, Suely Sono e Murrai Ascuccio. 3 ed. São Paulo: McGraw-Hill, 2008.

MILANI, A. **PostgreSQL – Guia do programador**.São Paulo: Novatec, 2008.

ORACLE. **Database 2 Day Developer's Guide**. 2016. Disponível em: <https://docs.oracle.com/cd/B28359_01/appdev.111/b28843/tdddg_procedures.htm> Acesso em 1 jul. 2016.

ORACLE. **Database Administrator's Guide**. Disponível em: <http://docs.oracle.com/cd/B19306_01/server.102/b14231/ds_concepts.htm> Acesso em: 18 ago. 2014.

ORACLE. Database **Application Developer's Guide - Fundamentals**. 2016. Disponível em:

<https://docs.oracle.com/cd/B19306_01/appdev.102/b14251/adfns_triggers.htm> Acesso em: 1 jul. 2016

ORACLE. **Oracle Benchmarks**. Disponível em: <http://www.oracle.com/us/solutions/performance-scalability/index.html> Acesso em: 18 ago. 2014.

ÖZSU M.T.; VALDURIEZ P. **Principles of distributed database systems**.3. ed. Nova York: Springer, 2011.

PGPOOL. **Welcome to Pgpool wiki! – What is Pgpool-II?**.Disponível em: <http://www.pgpool.net/mediawiki/index.php/Main_Page> Acesso em: 1 out. 2015.

POSTGRESQL. **PostgreSQL 9.0.23 Documentation.** Disponível em: <https://www.postgresql.org/docs/9.0/static/sql-select.html> Acesso em: 1 out. 2015

POSTGRESQL. **PostgreSQL 9.1.22 Documentation.** Disponível em: <https://www.postgresql.org/docs/9.1/static/pgbench.html> Acesso em: 1 out. 2015

RAINER K. R.; CEGIELSKI C. G. **Introdução a sistemas de informação**. Tradução Multinet Produtos. 3. ed. Rio de Janeiro: Elsevier, 2011.

RAMAKRISHNAN R.; GEHRKE J. **Sistemas de Banco de dados**. Tradução Acauan Pereira Fernandes, Celia Taniwak, João Tortello. 3. ed. São Paulo: McGraw-Hill, 2008.

SILBERSCHATZ A. et. al. **Sistemas de banco de dados**. Tradução Daniel Vieira. 6. ed.Rio de Janeiro: Elsevier, 2012.

SMITH, G. **PostgreSQL 9.0 – High Performance**. Birmingham: Pack Publishing Ltd., 2010.

TANENBAUM A.S.; VAN STEEN M.; **Sistemas distribuídos: princípios e paradigmas**. Tradução Arlete Simille Marques. 2. ed.São Paulo: Pearson Prentice Hall, 2007.

TPC. **About the TPC**. Disponível em: <http://www.tpc.org/information/about/abouttpc.asp> Acesso em: 17 ago. 2014.

TPC. **TPC Benchmarks**. Disponível em: <http://www.tpc.org/information/benchmarks.asp> Acesso em: 18 ago. 2014.

TPC benchmark b. **Transaction Processing Performance Council TPC**. San Jose: Junho, r. 2, p. 4-13. 1994.

ANEXO A – INSTALAÇÃO DEBIAN

Para instalação do sistema operacional Debian nas máquinas virtuais, foi usado uma imagem ISO disponível em http://cdimage.debian.org/debian-cd/8.2.0/amd64/iso-cd/ em sua versão 7.6 *Wheezy*.

Após o sistema ser inicializado, a tela principal é apresentada, sendo necessário selecionar a primeira opção "*Install*" de acordo com a figura A.1.

Figura A.1 – Primeiros passos Debian

Posteriormente no momento da instalação, é iniciado o processo da escolha do idioma, para o presente trabalho foi escolhido Português do Brasil. Após isso o sistema fará um pré-carregamento de todas as suas configurações até a tela seguinte, de configuração de rede no qual a primeira tela pede-se o nome da máquina conforme a figura A.2.

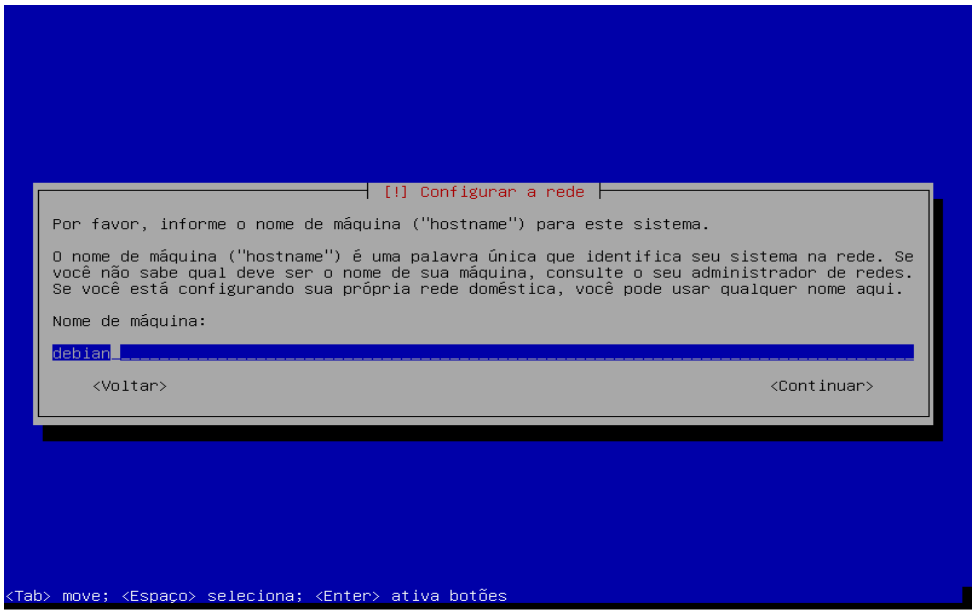

Figura A.2 – Configuração de rede

Após isso, é pede-se o nome de domínio, no qual foi atribuído LOCAL.POOLER para cada servidor e depois, é necessário definir usuários e senhas para as maquinas. Foram definidos os usuários para cada servidor como pooler, node 1 e node 2 para cada um dos servidores conforme figura A.3

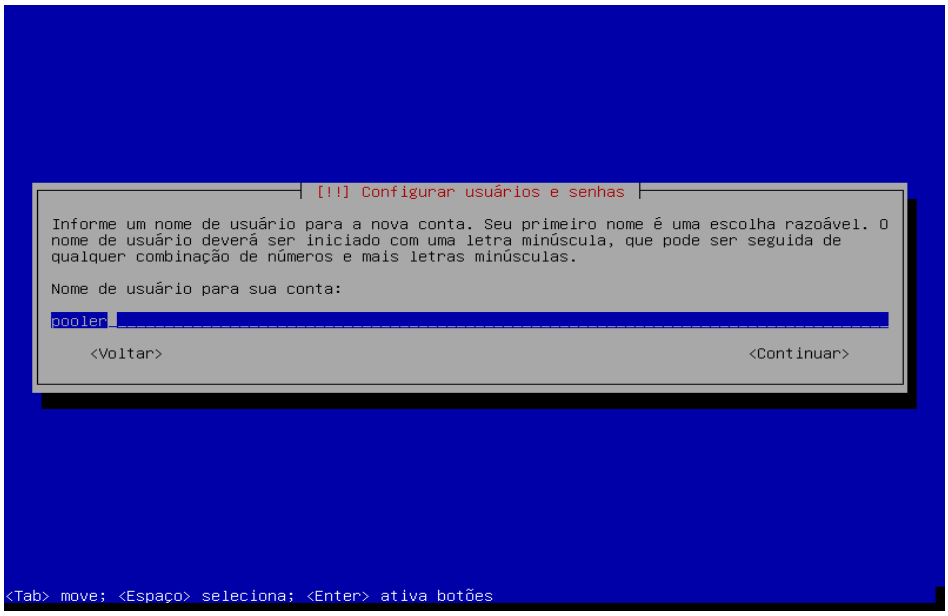

Figura A.3 – Configuração de usuários e senhas

Em seguida, é necessário criar uma senha. Assim, as próximas configurações de localidade serão previamente carregadas. Conforme a tela A.4, o sistema

requereque o usuário selecione qualopção de particionamento de discos será usada, foi utilizado a opção para usar o disco inteiro.

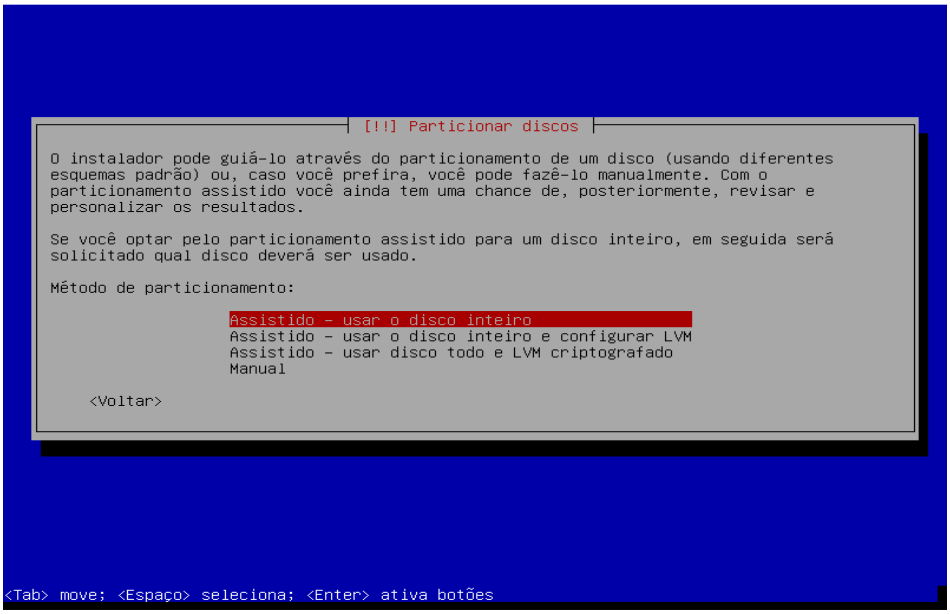

Figura A.4 – Opções de particionamento de discos

Foi utilizada a opção para usar o disco inteiro e o disco particionado do tipo SCISI1 conforme figura A.5 e seu esquema de particionamento foi definido pela opção padrão como mostra a figura A.6.

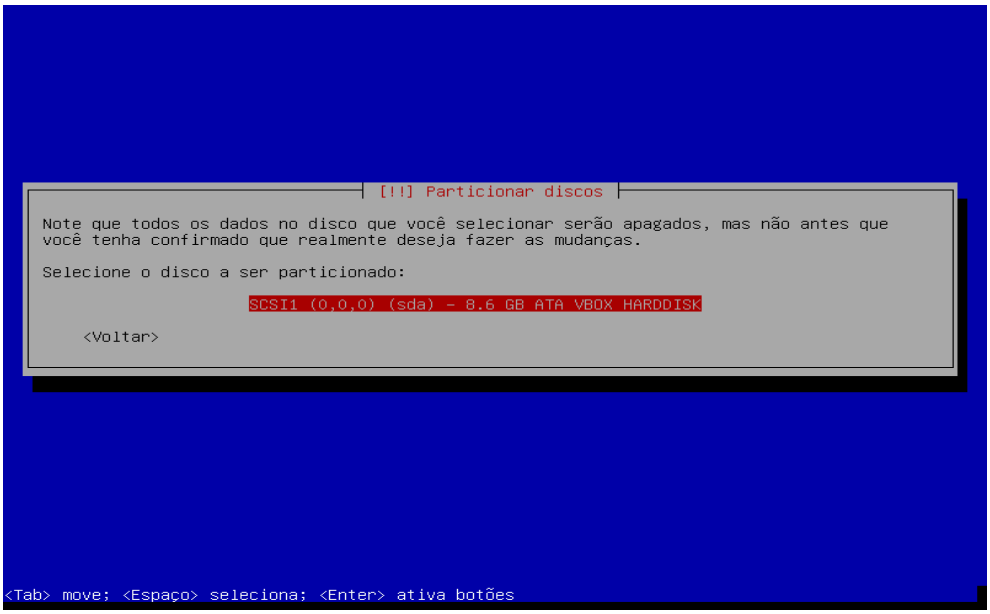

Figura A.5 – Seleção de disco

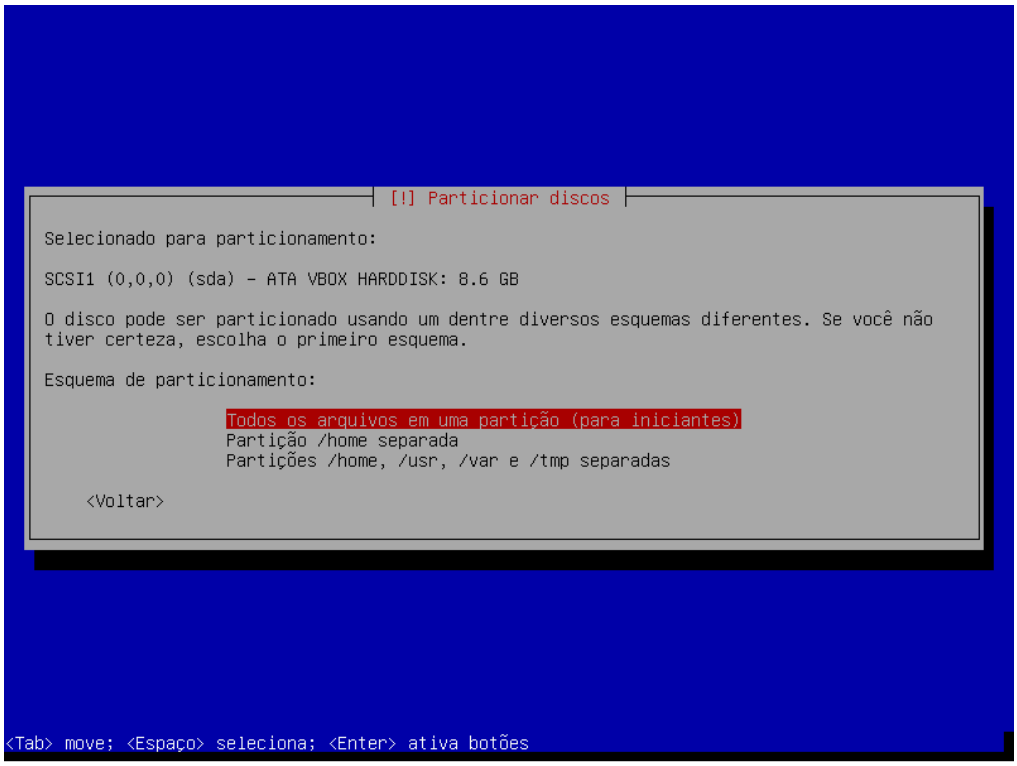

Figura A.6 – Esquema de particionamento

Na sequência, o carregamento das partições do sistema, é necessário finalizar o particionamento e escrever as mudanças no disco e confirmar para que o instalador particione os discos. Depois, o sistema será instalado. Em seguida, é solicitado configuração de gerenciador de pacotes do Debian. Durante esta instalação, é requisitado o uso de um espelho de rede, que não foi utilizado nesta instalação. O restante das configurações de pacotes serão carregadas. Ao finalizar o configurador de pacotes do Debian, é necessário escolher qual software será instalado. Foi escolhida a primeira opção, “Debian desktop environment”. A figura A.7 ilustra a opção.

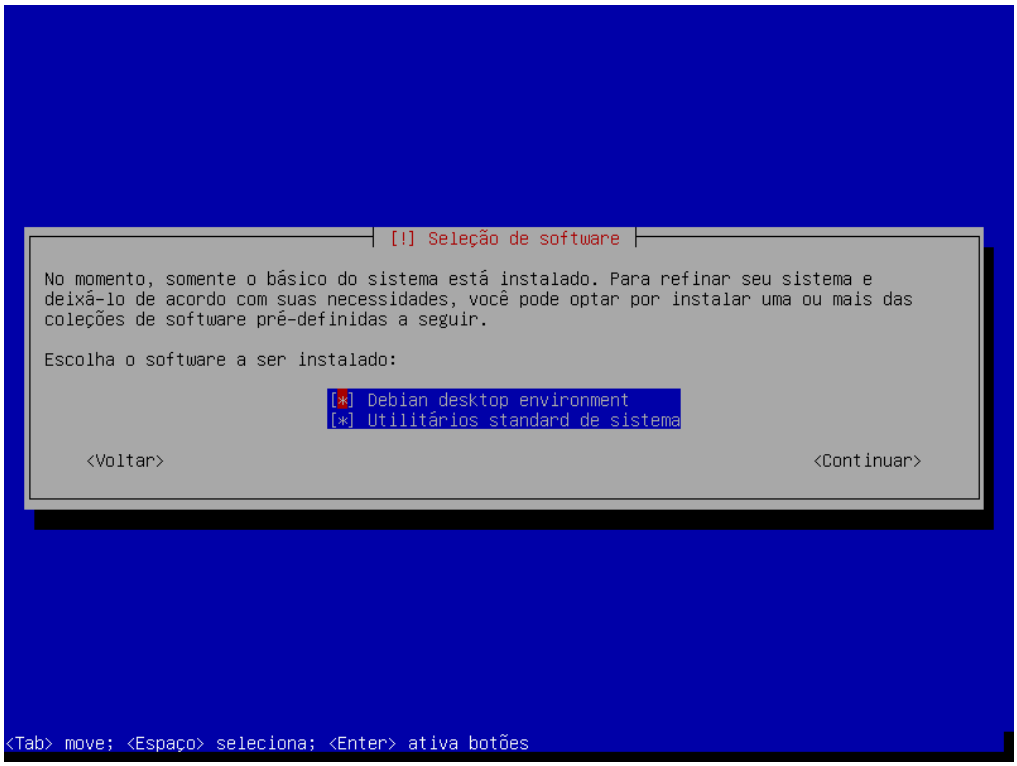

Figura A.7 – Seleção de software

Após todos os programas serem instalados, o sistema solicita instalar o carregador de inicialização GRUB em um disco rígido conforme a figura A.8

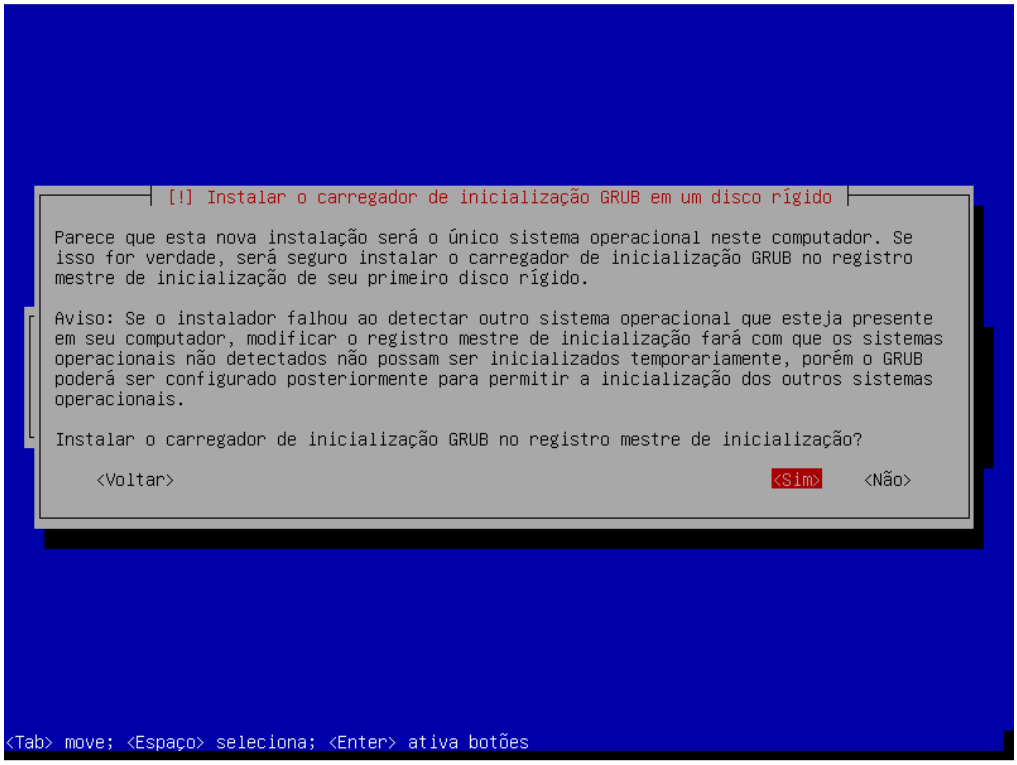

Figura A.8 – Carregador de inicialização GRUB

Desta forma, a instalação do sistema operacional será concluída conforme figura A.9 e o sistema irá inicializar-se, carregando sua tela de login e senha de acordo com a figura A.10

Figura A.9 – Finalização da instalação.

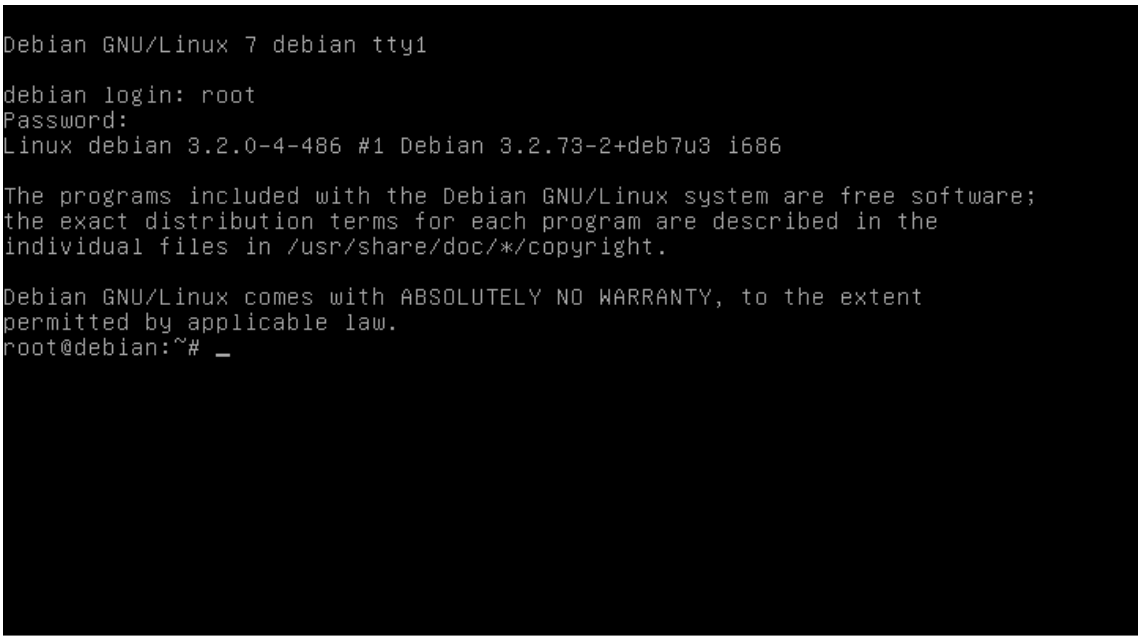

Figura A.10 – Prompt de comando Debian.

ANEXO B – INSTALAÇÃO POSTGRESQL

Para realizar e gerenciar as conexões de banco de dados, foi imprescindível a instalação do banco de dados PostgreSQL. Para fazer o *donwload* da instalação correspondente, é necessário baixar no site “postgresql.org”. Abaixo consta todos os passos de sua instalação. Para iniciar, após o instalador ser executado, clicar na opção "*next*" conforme figura B.1 e B.2.

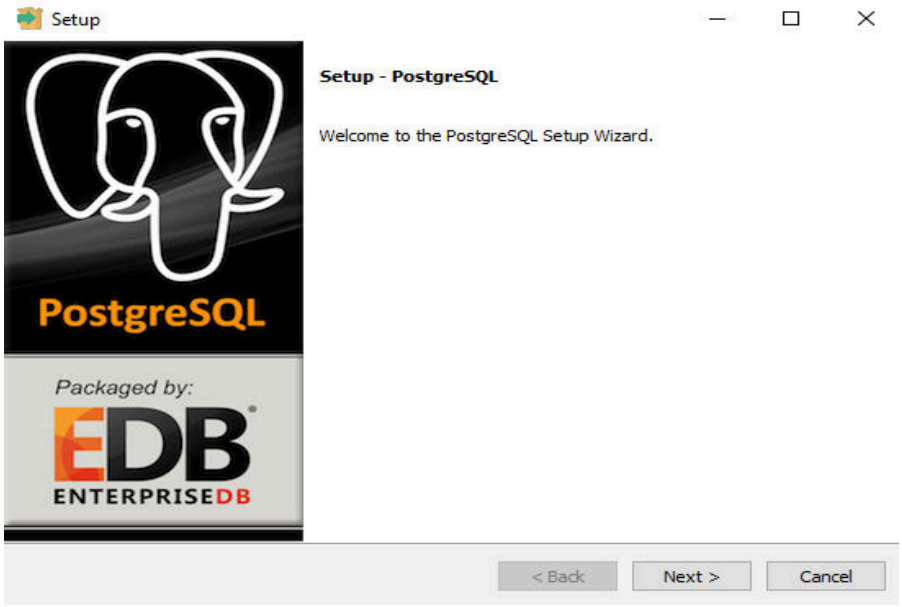

Figura B.1 – *Setup*PostgreSQL

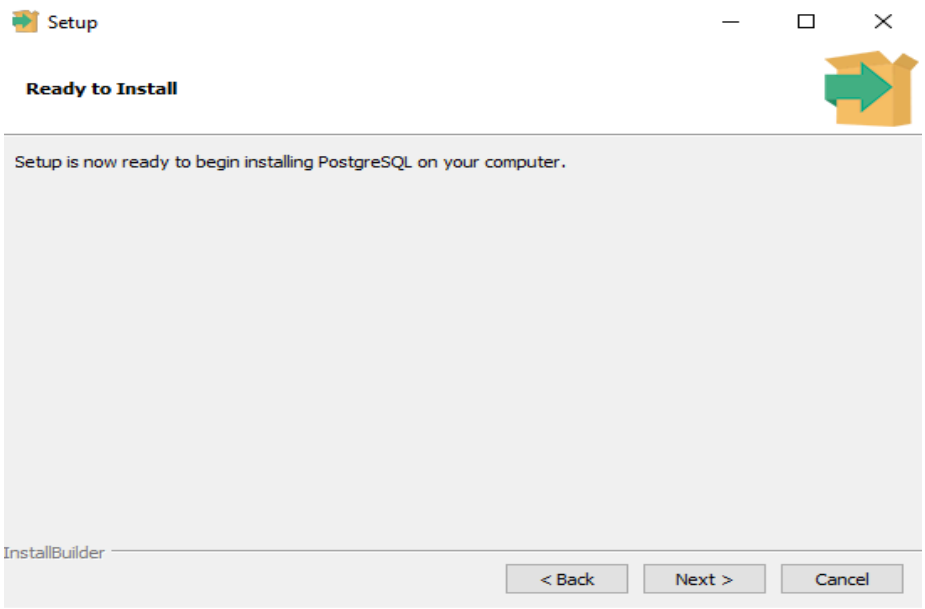

Figura B.2 – InstalaçãoPostgreSQL

Em seguida, o instalador adiciona os pacotes e configurações disponíveis na máquina e a instalação é finalizada conforme figura B.3.

Figura B.3 – Instalador de pacotes adicionais ao PostgreSQL

O*StackBuilder* complementa a instalação do PostgreSQL, adicionando softwares integrantes ao banco de dados. Selecionar o PostgreSQL 9.4 (x64) *onport* 5432 na lista disponibilizada e clicar em "*next*" como mostra a figura B.4.

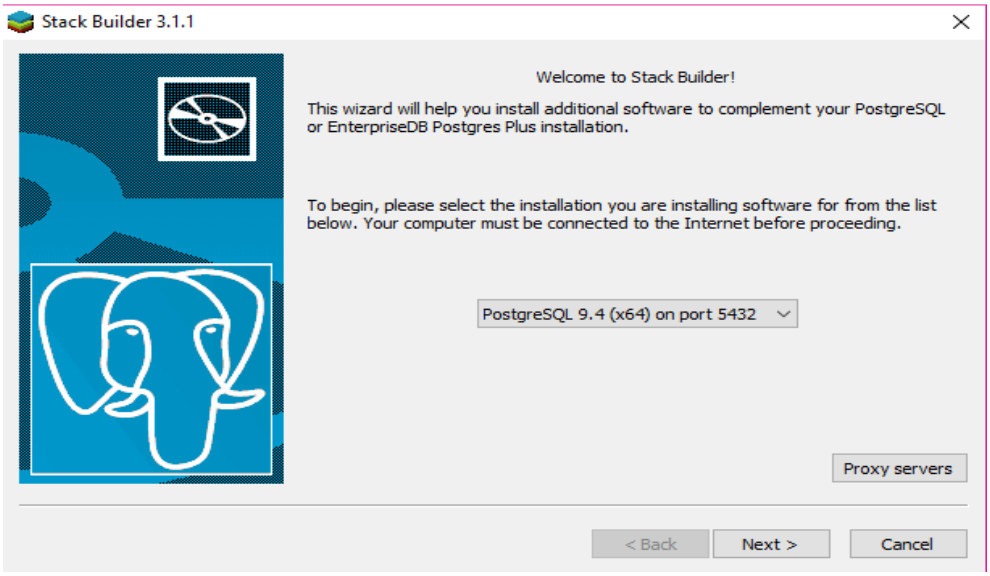

Figura B.4 – Pacotes adicionais

Na figura B.5, o passo é fundamental selecionar as aplicações desejadas para instalação, foi selecionado somente o banco de dados PostgreSQL em sua versão 32 bit que disponibiliza juntamente em sua instalação o PgAdmin que é fundamental para controle dos banco de dados utilizados neste trabalho. Clicar em "*next*" e confirmar a instalação na próxima caixa de dialogo para *download* do programa e posterior instalação.

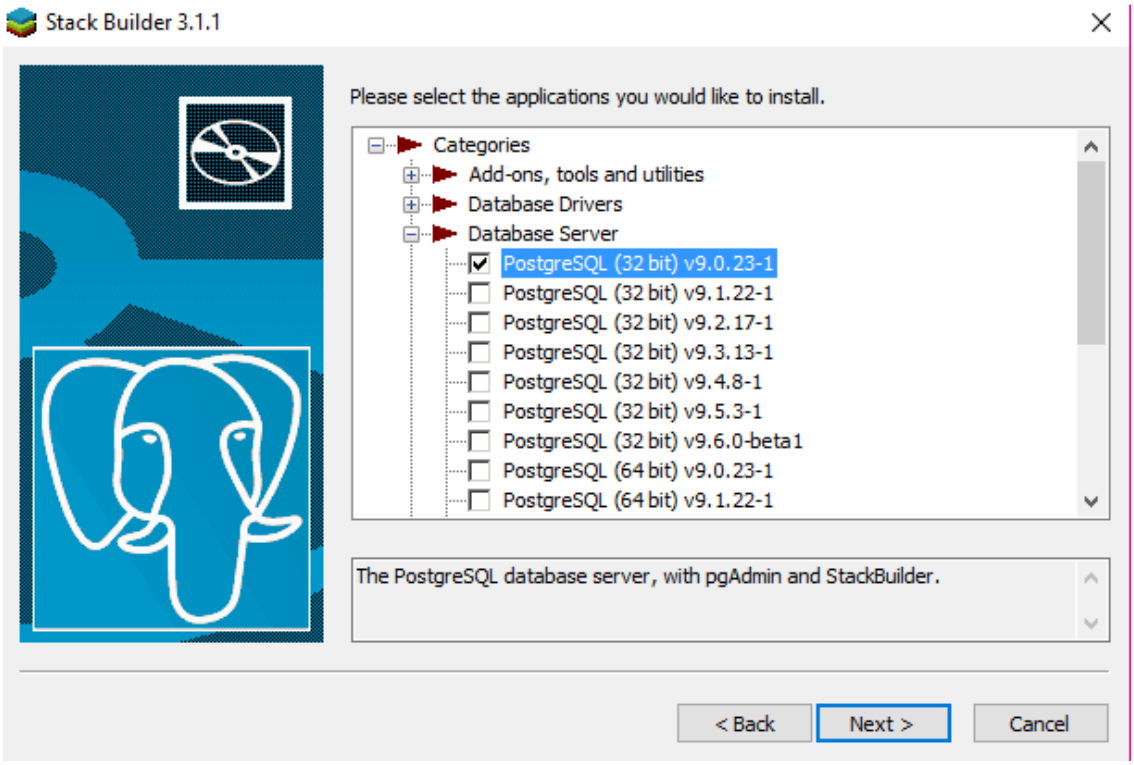

Figura B.5 – Selecionando componentes

Para continuar, clicar em "*next*" para iniciar a instalação e a caixa de dialogo a seguir disponibiliza o diretório que será instalado a aplicação escolhida. Foi instalado no diretório raiz, C: conforme figura B.6.

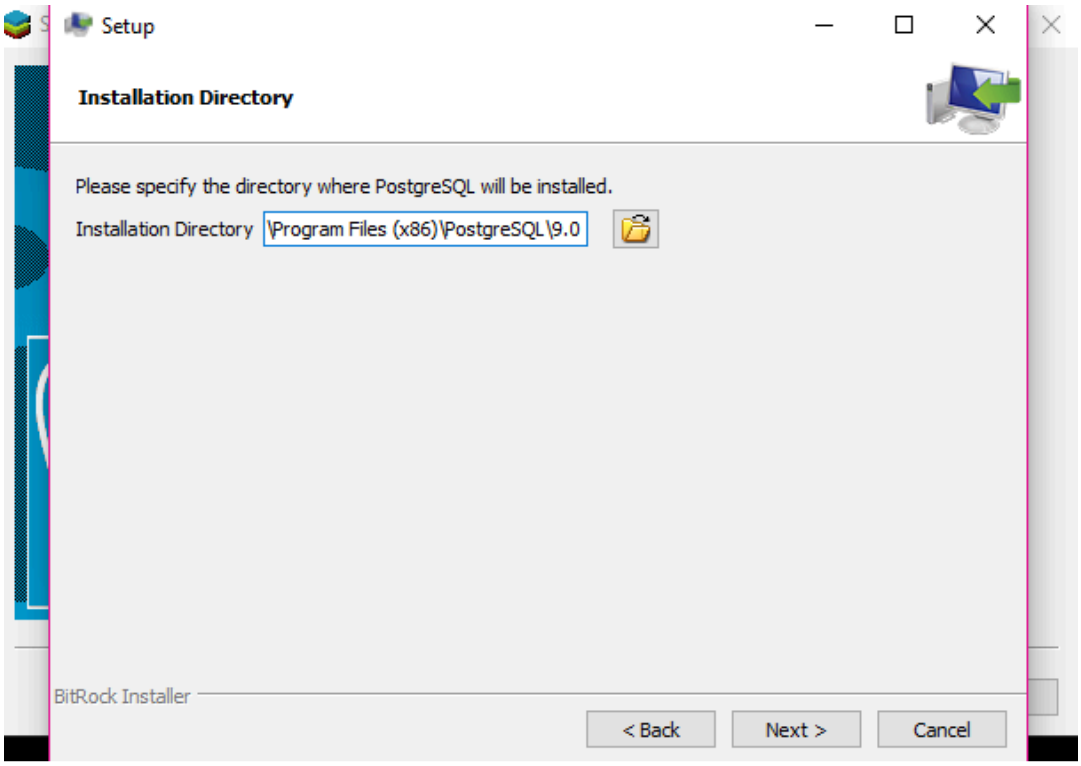

Figura B.6 – Selecionando o diretório

A seguir, é necessário criar o *password* que será usado para as futuras conexões do banco de dados. É preciso defini-lo e clicar em "*next*"

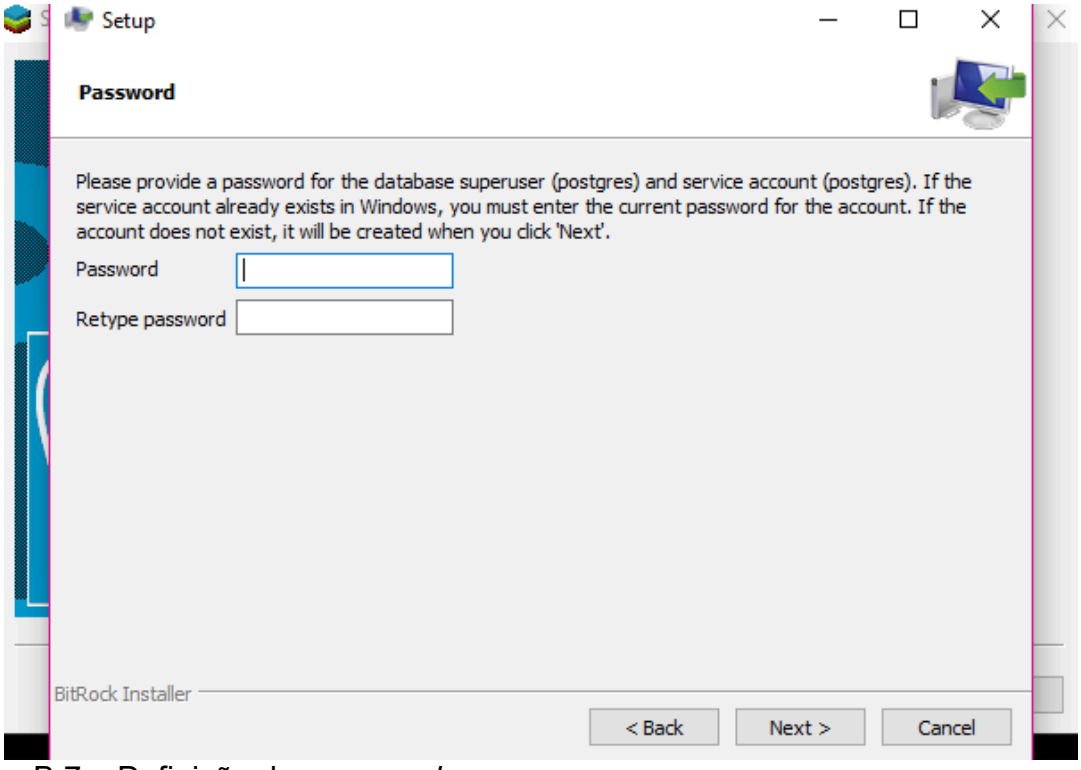

Figura B.7 – Definição de *password*

O próximo passo é selecionar a porta que fará a comunicação entre o banco e máquinas, neste caso foi utilizado a porta padrão 5433 conforme figura B.8.

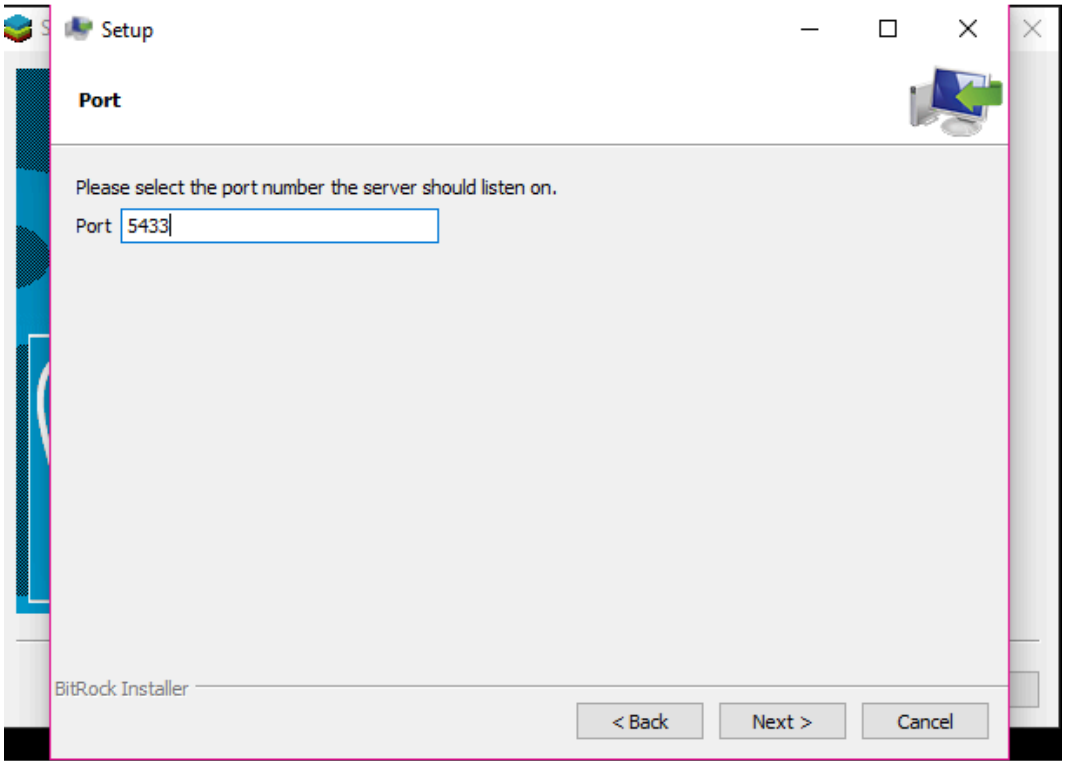

Figura B.8 – Seleção da porta correspondente

A seguir, é solicitado o local do usuário, que foi atribuído a *Portuguese*, *Brazil* de acordo com figura B.9.

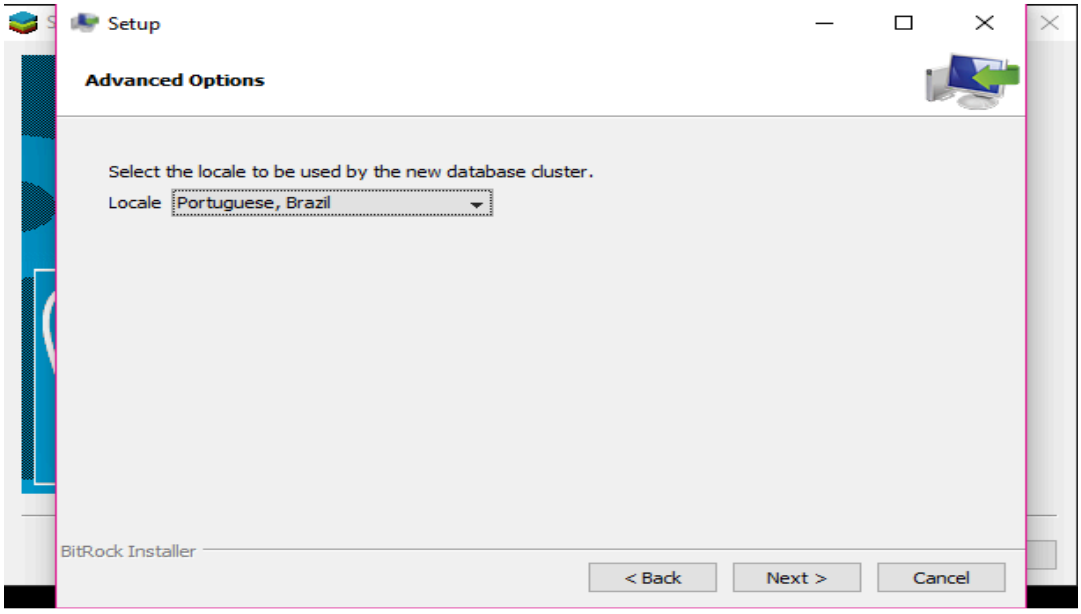

Figura B.9 – Seleção de localização cluster banco de dados

Concluindo, a instalação do PostgreSQL está finalizada conforme figura B.10.

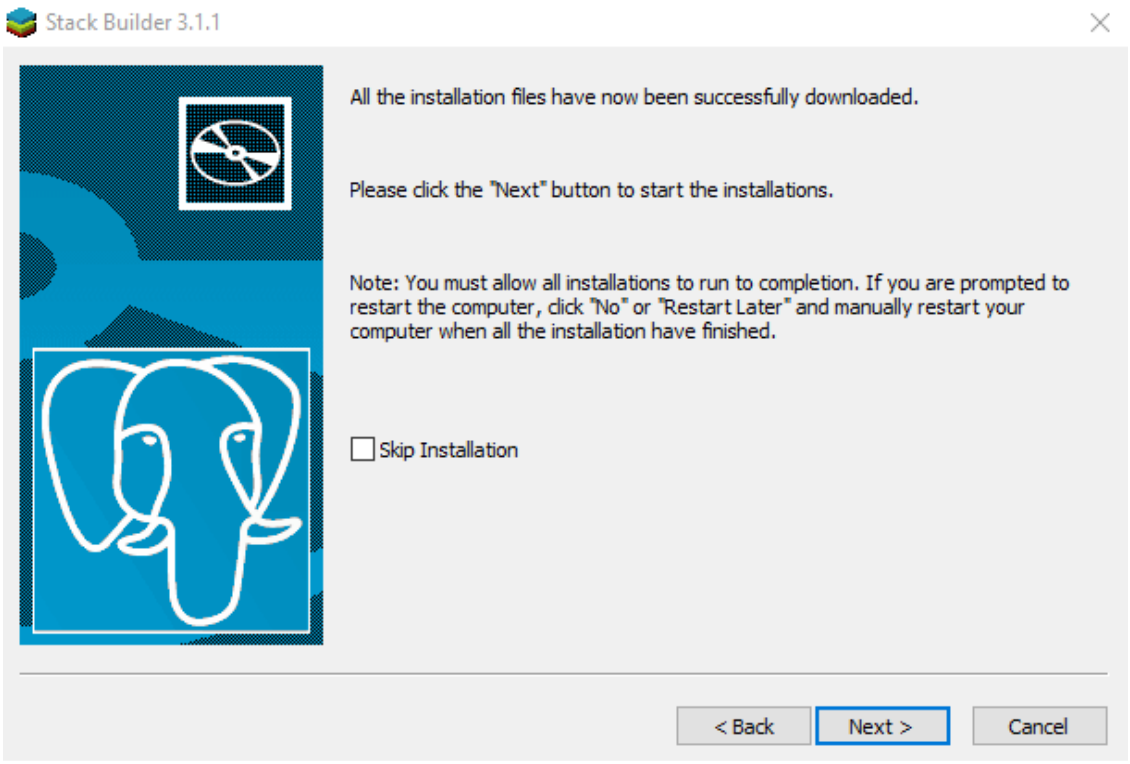

Figura B.10 – Conclusão do instalador PostgreSQL.

ANEXO C – INSTALAÇÃO PGPOOL

Para instalar o Pgpool, é primordial que o *download* da ferramenta seja feito no sistema operacional debian, através da linha de comando:

wget -c http://www.pgpool.net/download.php? f=pgpool-II-3.1.1.tar.gz

Após finalizá-lo, o comando a seguir irá descompactar o PgPool para ser movido para outro diretório específico:

tar xvzf pgpool-II-3.1.1.tar.gz

Depois de descompactá-lo, é necessário entrar no diretório especificado e instalá-lo:

```
$ cd pgpool-II-3.1.1
$ ./configure
$ make
$ make install
$ make clean
```

Após este passo, será preciso copiar o arquivo base de configuração do Pgpool para altera-lo:

```
$ cp /usr/local/etc/pgpool.conf.sample /usr/local/etc/pgpool.conf
$ pico /usr/local/etc/pgpool.conf
```

Ao entrar no arquivo especificado, os parâmetros que permitirão a realização da comunicação entre os nós e pooler, ficarão disponíveis para que o usuário possa habilitá-los, além da possibilidade de configurar a replicação e balanceamento de cargas que o Pgpool executará. O arquivo de código a seguir foi utilizado para elaborar o ambiente de testes, algumas opções permaneceram com especificação padrão da ferramenta. As linhas de comando modificadas para o propósito deste

trabalho foram sinalizadas através de comentários que explanam o significado de cada parâmetro.

```
# ----------------------------
# pgPool-II configuration file
# ----------------------------

#------------------------------------------------------------------------
# CONNECTIONS
#------------------------------------------------------------------------

# - pgpool Connection Settings -

listen_addresses = '192.168.100.1'
                            # Endereço de IP utilizado para o modo de escuta.
port = 5432
                            # Número da porta
socket_dir = '/var/run/postgresql'

# - pgpool Communication Manager Connection Settings -

pcp_port = 9898
                            # Número da porta PCP
pcp_socket_dir = '/var/run/postgresql'

# - Backend Connection Settings -

backend_hostname0 = '192.168.100.1'
backend_port0 = 5432
backend_weight0 = 1
backend_data_directory0 = '/var/lib/postgresql/9.3/main/'
backend_flag0 = 'ALLOW_TO_FAILOVER'
```

```
backend_hostname1 = '192.168.100.2'
backend_port1 = 5432
backend_weight1 = 1
backend_data_directory1 = '/var/lib/postgresql/9.3/main/'
backend_flag1 = 'ALLOW_TO_FAILOVER'

backend_hostname2 = '192.168.100.3'
backend_port2 = 5432
backend_weight2 = 1
backend_data_directory2 = '/var/lib/postgresql/9.3/main/'
backend_flag2 = 'ALLOW_TO_FAILOVER'

#endereços de IP habilitados para conexão

# - Authentication -

enable_pool_hba = on
                    # Utilizar o arquivo pool_hba.conf para autenficação do usuário
pool_passwd = 'pool_passwd'

authentication_timeout = 60
                    # Tempo de espera em segundos para completar a
autenficação do cliente.

# - SSL Connections -

ssl = off
                    # Desabilitado suporte SSL
#ssl_key = './server.key'

#ssl_cert = './server.cert'

#ssl_ca_cert =
```

```
#ssl_ca_cert_dir =

#---------------------------------------------------------------------------
# POOLS
#---------------------------------------------------------------------------

# - Pool size -

num_init_children = 32
                        # Número de pools
max_pool = 8
                        # Número de conexões por pool
# - Life time -

child_life_time = 300
                        # O pool existe após ficar inativo pelos segundos especificados.
child_max_connections = 0

connection_life_time = 0

client_idle_limit = 0

#---------------------------------------------------------------------------
# LOGS
#---------------------------------------------------------------------------

# - Where to log -

log_destination = 'stderr'

# - What to log -

print_timestamp = on
                        # Mostrar o timestamp em cada linha
```

```
log_connections = on
                        # Registro de conexões
log_hostname = on
                        # Hostname será mostrado no status e nos registros se as
conexões forem logadas.
log_statement = off

log_per_node_statement = off

log_standby_delay = 'none'

# - Syslog specific -

syslog_facility = 'LOCAL0'

syslog_ident = 'pgpool'

# - Debug -

debug_level = 0

#------------------------------------------------------------------------
# FILE LOCATIONS
#------------------------------------------------------------------------

pid_file_name = '/var/run/postgresql/pgpool.pid'
                        # Nome do arquivo PID.
logdir = '/var/log/postgresql'
                        # Diretório do arquivo de status do pgPool

#------------------------------------------------------------------------
# CONNECTION POOLING
#------------------------------------------------------------------------
```

```
connection_cache = on
                        # Ativar conexões de pool

reset_query_list = 'ABORT; DISCARD ALL'

#reset_query_list = 'ABORT; RESET ALL; SET SESSION AUTHORIZATION DEFAULT'

#------------------------------------------------------------------------
# REPLICATION MODE
#------------------------------------------------------------------------

replication_mode = on
                        # Ativa o modo de replicação.
replicate_select = off

insert_lock = on

# - Degenerate handling -

replication_stop_on_mismatch = off

failover_if_affected_tuples_mismatch = off

#------------------------------------------------------------------------
# LOAD BALANCING MODE
#------------------------------------------------------------------------

load_balance_mode = on
                        # Ativa o modo de balanceamento de carga.
ignore_leading_white_space = on

black_function_list = 'nextval,setval'
```

```
#------------------------------------------------------------------------
# MASTER/SLAVE MODE
#------------------------------------------------------------------------

master_slave_mode = off

master_slave_sub_mode = 'slony'

# - Streaming -

sr_check_period = 0
                        # Disabled (0) by default
sr_check_user = 'nobody'

#sr_check_password =

delay_threshold = 0

#------------------------------------------------------------------------
# PARALLEL MODE
#------------------------------------------------------------------------

parallel_mode = off

#pgpool2_hostname =

# - System DB info -

system_db_hostname  = 'localhost'

system_db_port = 5432

system_db_dbname = 'pgpool'
```

```
system_db_schema = 'pgpool_catalog'

system_db_user = 'pgpool'

#system_db_password =

#------------------------------------------------------------------------
# HEALTH CHECK
#------------------------------------------------------------------------

health_check_period = 0

health_check_timeout = 20

health_check_user = 'nobody'

#health_check_password =

health_check_max_retries = 0

health_check_retry_delay = 1

#------------------------------------------------------------------------
# FAILOVER AND FAILBACK
#------------------------------------------------------------------------

fail_over_on_backend_error = on

search_primary_node_timeout = 10

#------------------------------------------------------------------------
# ONLINE RECOVERY
#------------------------------------------------------------------------
```

```
recovery_user = 'postgres'

recovery_password = 'postgres'

recovery_1st_stage_command = '/usr/share/doc/pgpool2/examples/pgpool_recovery'

recovery_2nd_stage_command =
'/usr/share/doc/pgpool2/examples/pgpool_recovery'

recovery_timeout = 90

client_idle_limit_in_recovery = 0

#------------------------------------------------------------------------
# WATCHDOG
#------------------------------------------------------------------------

use_watchdog = off
```