UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RODRIGO DLUGOKENSKI

Bancos de dados para monitoramento de desempenho de grandes redes

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

Orientador: Prof. Dr. Valter Roesler
Co-orientador: Eng. William Lautschënlager

Porto Alegre
2016

CIP – CATÁLOGO NA PUBLICAÇÃO

Dlugokenski, Rodrigo
Bancos de dados para monitoramento de desempenho de grandes redes / Rodrigo Dlugokenski. -- 2016.
65 p.
Orientador: Valter Roesler.
Coorientador: William Lautschënlager.
Trabalho de conclusão de curso (Graduação) -- Universidade Federal do Rio Grande do Sul, Instituto de Informática, Curso de Ciência da Computação, Porto Alegre, BR-RS, 2016.
1. bancos de dados. 2. avaliação de desempenho. 3. sequência de tempo. 4. séries de tempo. 5. monitoramento de desempenho. I. Roesler, Valter, orient. II. Lautschënlager, William, coorient. III. Título.

AGRADECIMENTOS

Ao Professor Valter e William por toda confiança.

Cíntia, minha esposa, que trilhou todo esse caminho ao meu lado.

Helena, minha filha. Especialmente. Sorrisos, sorrisos, sorrisos. Todos gratuitos. *-*

RESUMO

As operadoras de rede atualmente sentem a necessidade de monitorar em tempo real o desempenho fim a fim de suas redes (ex.: vazão de dados, qualidade da experiência do usuário, taxa de perda de pacotes, etc.). O problema enfrentado é o baixo desempenho de bancos de dados relacionais frente à grande quantidade de dados que é adquirida nos testes de monitoramento.

Nesse contexto, têm surgido novos tipos de banco de dados que conseguem lidar com essas quantidades massivas de dados, e permitem que se construam aplicações de visualização destes dados em tempo quase real.

O objetivo desse trabalho será comparar algumas dessas soluções de armazenamento de dados – seu funcionamento e suas implicações – neste caso, uma solução relacional tradicional – o *PostgreSQL* – versus uma especializada – *time series databases* – representado aqui pelo *InfluxDB*.

Para isso serão definidas métricas e será efetuada uma implementação de método de teste para que sejam analisadas as soluções.

Palavras-chave: bancos de dados; sequência de tempo; sequência de eventos; operador histórico; avaliação de desempenho.

Data storages for real time monitoring of big networks

ABSTRACT

Nowadays, network operators are feeling the need for real time monitoring of end-to-end performance from their networks (e.g.: data throughput, quality of experience, lost packets percent, etc.). But a frequent problem of using this approach is the low performance of common relational database solutions facing the enormous quantity of data generated by such measurements.

On that aspect, the database market is offering new types of data storage solutions that promise to handle such challenges, making new ways to build real time monitoring applications on top of them.

The main objective of this work is to verify some of these solutions, compare some of their characteristics, and, after ruling out most of them, it will take two solutions to a proper benchmark. One of them will be a relational database – *PostgreSQL*, and another one will be a non-relational solution – *InfluxDB*.

In order to make things work out, this article will define some metrics, use them to implement a benchmark method, and finally show the benchmark results to analyze both solutions.

Keywords: databases; benchmark; time series; event series; operational historian; performance analysis.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento, Durabilidade; características desejáveis em bancos de dados
ACK	*Acknowledged,* ou "recebido"
API	*Application Programming Interface*, interface de interação com aplicações
BASE	*Basically Available, Soft state, Eventual consistency*, ou disponibilidade básica, estado leve, consistestência eventual; tipo de disciplina para bancos de dados
BRIN	*Block Range Index.* É um tipo de índice do PostgreSQL para blocos de dados
B-TREE	Árvore B, uma maneira de organizar o acesso à dados por uma determinada chave.
CAP	*Consistency availability partition tolerance*, tolerância entre consistência, disponibilidade e particionamento
CLI	*Command Line Interface*, Interface via linha de comando
DBMS	*Database Management Systems*, ver SGBD
DOCSIS	*Data Over Cable Service Interface Specification*, comunicação em redes de TV
DSL	*Digital Subscriber Line,* comunicação digital em redes telefônicas
FIFO	*First In – First Out,* uma disciplina de descarte/saída em filas
FTP	*File Transfer Protocol*, protocolo de transferência de arquivos
GIT	Um sistema de gerenciamento de código fonte
HTML	*Hyper Text Markup Language*, uma linguagem de marcação de hipertexto
HTTP	*Hyper Text Transfer Protocol*, protocolo de transferência de hipertexto
ICMP	*Internet Control Message Protocol*, protocolo de diagnóstico e controle da Internet
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
InfluxQL	Linguagem de consulta de banco de dados do InfluxDB
IoT	*Internet of Things*, ou Internet das Coisas, em tradução livre.
IP	*Internet Protocol,* conjunto de protocolos de comunicação via internet
IPv4	*Internet Protocol version 4*, protocolo estruturante da Internet, versão atualmente dominante
IPV6	*Internet Protocol version 6*, evolução do IPv4
ISO	*International Standards Organization,* ou Organização Internacional para Padronização
ITU	*International Telecommunication Union, ou* União Internacional de Telecomunicações
JS	*JavaScript*, uma linguagem de programação difundida na internet
JSON	*JavaScript Object Notation*, notação de objetos estilo JS, um meio de representar uma estrutura de dados
MVCC	*Multi-version concurrency control*
NewSQL	Classe de banco de dados que propõe desempenho NoSQL com garantias ACID
NIST	*National Institute of Standards and Technology*, equivalente Norte Americano do Instituto Nacional de Metrologia
NoSQL	*Not Only SQL*, visão alternativa ao SQL para gerenciamento de dados
PPP	*Point-to-point protocol,* protocolo de troca de dados entre vizinhos lógicos
PSQL	Acrônimo para *PostgreSQL*
QoE	*Quality of Experience,* ou Qualidade de Experiência
QoS	*Quality of Service,* ou Qualidade do Serviço
RDBMS	*Relational DBMS,* ou SGBD relacional
REST	*Representational State Transfer*, ou Transferência do Estado Representacional, um estilo arquitetural para troca de informações

RPC	*Remote Procedure Call,* um meio de chamada de procedimentos remotos
RRD	*Round Robin Database*, um banco de dados temporal circular
RTT	*Round Trip Time,* Tempo de ida e volta
SGBD	Sistema Gerenciador de Banco de Dados
SQL	*Structured Query Language*, meio de acesso a dados em SGBDs
SSD	Disco de estado sólido (armazenagem de dados)
SSH	*Secure Shell,* um terminal de acesso remoto seguro
SSL	*Secure Sockets Layer*, um protocolo de camada de comunicação segura
TCP	*Transmission Control Protocol,* um protocolo de controle de transmissão
TMUX	*Terminal Mutiplexer*. Um multiplexador de terminais que também permite manter a sessão atual rodando mesmo desconectado.
TSDB	*Time Series Database* ou Banco de dados de séries temporais
UDP	*User Datagram Protocol,* um protocolo de transmissão simples
UIT	*Ver ITU*
UNIX	Atualmente uma classe de sistemas operacionais
USB	*Universal Serial Bus,* barramento serial universal
W3C	*World Wide Web Consortium,* ou Consórcio da WWW, entidade de padronização da WWW
WAL	*Write Ahead Log.* É um registro de dados a serem inseridos em BDs em breve
WWW	*World Wide Web,* ou Grande Rede Mundial (de computadores)
XML	*eXtensible Markup Language*, linguagem de marcação de documentos

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivações

Bancos de dados são parte fundamental de qualquer procedimento de coleta e de análise de variados tipos de dados. Para estes diversos tipos de dados e necessidades analíticas que existem, também há diferentes tipos de bancos de dados. No entanto, em geral, é bastante natural apenas tomar um banco de dados de uso cotidiano – como os bancos relacionais – e usar para toda e qualquer tarefa. No entanto não se deve esquecer de um detalhe importante: o crescimento das redes de computadores, tanto em tamanho de infraestrutura, quanto em horas de uso por usuários e dispositivos.

> *Computer networks like the Internet are gaining importance in social and economic life ... it seems to be certain that networks will change the socioeconomic structures we know today. (MAIER e KAUFMANN, 2001, p. 1)*

Esses dois fatores de crescimento mostram que a necessidade dessas redes é cada vez mais vital, e, como algo cada vez mais utilizado, cria-se o desafio de mensurar o desempenho dessas redes a fim de ganhar mais confiança na operação das mesmas.

> *Since the early days of data communications, network performance measurement has played a key role in the continuous enhancement and evolution of packet switching technologies. (PÁSZTOR, 2003, p. 1)*

Para resolver o desafio citado, é preciso verificar as redes de computadores cada vez mais, para garantir seu bom funcionamento, por exemplo, colocando sensores e testadores em todos os lugares. Mas, e a infraestrutura dessas informações? O que fazer com tantos dados criados, para onde enviar e como fazer a consulta destes dados – de maneira eficiente?

> *It's not uncommon to have to deal with petabytes of data, even when carrying out traditional types of analysis and reporting. As a result, it has become harder to do the same things you used to do. (DUNNING e FRIEDMAN, 2015, p. 11)*

Uma das possíveis soluções para essa dificuldade que se está perseguindo é a adoção de soluções não tradicionais, como bancos de dados não relacionais, ou ainda, bancos especializados em determinado tipo de dados, no caso específico de redes de computadores e transmissões de dados, bancos de séries temporais de dados.

> *A TSDB is optimized for best performance for queries based on a range of time. New NoSQL approaches make use of non-relational databases with considerable advantages in flexibility and performance over traditional relational databases (RDBMS) for this purpose. (DUNNING e FRIEDMAN, 2015, p. 12)*

1.2 Objetivos

O objetivo deste trabalho é comparar, sob diferentes métricas de desempenho e de utilização de recursos, possíveis soluções para a ingestão, seleção e sintetização de dados, sejam com bancos relacionais, sejam com bancos mais especializados, no contexto da área de redes.

Com isso, visa-se a responder questionamentos sobre a adequação dessas soluções para este contexto. Além disso, busca-se mensurar em termos quantitativos os ganhos que podem ser alcançados nestas escolhas.

Nesse trabalho é proposto, portanto, avaliar essa parte da infraestrutura que é necessária para avaliação de desempenho de redes de computadores em larga escala.

1.3 Estrutura

Visando alcançar os objetivos apresentados, este trabalho está particionado em capítulos, da seguinte maneira:

- **Conceitos:** serão fornecidas informações dos principais conceitos abordados e de tecnologias utilizadas, para que se esteja bem situado naquilo que será discutido posteriormente, subdivido assim:
  - *Redes*: O alvo é o armazenamento de informações de desempenho de redes de computadores, utilizando uma tecnologia específica.
  - *Bancos de dados:* Uma breve introdução dos tipos que serão testados nesse escrito.
  - *Linguagem e acesso aos dados:* Os tipos de linguagens e protocolos de documentos utilizados para fazer o processamento dos dados, desde a entrada, até a saída.
  - *Ambiente:* Brevemente serão vistos os conceitos tecnológicos que serão usados no ambiente de teste.
- **Sistemas de gerenciamento de dados:** Nesse capítulo serão discutidos algumas das possíveis escolhas dos bancos de dados que podem ajudar a alcançar os objetivos propostos na seção 1.2, bem como definir critérios para escolher quais sistemas serão testados.
- **Metodologia de avaliação:** Serão definidos nessa parte do trabalho quais as métricas serão empregadas para comparar os candidatos escolhidos, especificidades do ambiente utilizado para fazer as comparações, além de como foi gerada a carga de teste utilizada.
- **Testes: Execução e Resultados:** Serão mostrados os resultados da aplicação da metodologia, junto com uma análise crítica desses resultados.
- **Conclusões:** Apresentar a resposta que atinja os objetivos propostos e fazer uma análise do trabalho como um todo e de seu possível futuro.

2 CONCEITOS

2.1 Introdução

Neste trabalho, como visto na seção 1.2, busca-se soluções adequadas para o armazenamento e consulta de dados obtidos através da medição de redes de computadores.

Portanto, nas seções seguintes, primeiramente serão mostrados conceitos considerados importantes para avaliação de desempenho de **redes de computadores**, em especial uma noção da complexidade dessas redes, e em seguida a suíte de ferramentas NetMetric, que faz medições de redes fim-a-fim.

Após isso, na seção 2.3, serão apresentados conceitos pertinentes a **bancos de dados**, o alvo da avaliação desse trabalho. Serão apresentadas brevemente as tecnologias testadas com uma discussão do histórico delas, além da relevância no contexto do NetMetric.

Ambos conceitos, de redes de computadores e de bancos de dados, serão utilizados para o desenvolvimento do restante deste trabalho.

2.2 Redes de computadores

2.2.1 As redes e sua complexidade

Redes de computadores são um aspecto imperativo na funcionalidade de sistemas cada vez mais complexos de computação e de automatização de tarefas hoje. Por exemplo, como controlar dezenas de robôs em um ambiente em que uns cooperam com os outros[1]? As redes de computadores estão envolvidas neste e em muitos outros aspectos da vida contemporânea.

Considerada a utilização descrita acima, o fato é que normalmente se ignora como funciona todo o processo de comunicação, toda a teia de tecnologias envolvidas e seus pormenores – mas que realmente nem importam, na visão da aplicação final (e dos usuários, por consequência).

Tabela 2.1 - Modelo Abstrato de Camadas do TCP/IP

Camada	Unidade	Função	Exemplos
5. Aplicação	Dados	Interface de alto nível, incluindo acesso à arquivos remotos, compartilhamento de recursos, compressão de dados, codificação, criptografia, etc.	HTTPS, FTP, SSH, Samba, IMAP, SMTP, RPC, GZIP, SSL, Telnet
4. Transporte	Segmentos	Protocolos de comunicação entre quaisquer pontos da rede que oferecem a segmentação dos dados, multiplexação e, opcionalmente confirmação de recebimento dos mesmos.	TCP, UDP
3. Internet ou Rede	Pacotes	Estruturação e gerenciamento de uma rede de múltiplos nós, incluindo endereçamento, roteamento e controle de tráfego	IPv4, IPv6, ICMP
2. Link	Quadros	Transmissão confiável entre dois nós adjacentes por uma camada física	PPP, IEEE 802.2, L2TP, MAC, LLDP
1. Física	Bits	Transmissão e recepção de sequências de bits em um meio físico	DSL, USB, DOCSIS, Ethernet

Fonte: Adaptação das RFC 1122 (BRADEN, 1989), RFC 1123 (BRADEN, 1989) e (KUROSE e ROSS, 2012)

Outro ponto que deve ser levado em consideração nos capítulos a seguir é a complexidade da área de redes, que pode ser percebida na Tabela 2.1, um extrato de dois documentos

[1] Como foi feito em (HSIEH, COWLEY, *et al.*, 2006)

importantes da área, as RFCs 1122 e 1123, que tentam organizar toda a cadeia de tecnologias que formam uma rede. As RFCs (*Request for Comments)* são memorandos de normas e protocolos necessários para que os tantos equipamentos diferentes de uma rede se comuniquem. Esses documentos, no momento, totalizam 7429[2].

No final de toda essa complicada estruturação de uma rede, para quem é usuário, o detalhe que sempre será notado é – se o funcionamento é adequado, ou ainda – se o funcionamento é rápido, de maneira tal que não comprometam a utilidade dessas aplicações.

2.2.2 Avaliação de desempenho

A Tabela 2.1, além do grande número de documentos de normatização, apresenta uma ideia geral da complexidade de uma rede de computadores. No entanto, o detalhe mais observado nesse trabalho é como medir o seu estado de funcionamento, e o que medir, que é outra tarefa bastante trabalhosa, dado o notável número de elementos presentes.

Em várias áreas da ciência e tecnologia a avaliação de desempenho terá definições possivelmente diferentes, portanto será posto a seguir a definição que será considerada nesse trabalho.

De maneira ampla, um meio natural de pensar sobre avaliação de desempenho em canais de transferência é: injetar a carga bruta de um lado, o mais rápido possível, e observar na saída a carga que deixou este canal, a fim de verificar diversas propriedades, como a consistência da carga – se tudo chegou como foi injetado, e como a vazão de transferência – se foi tão rápido quanto o esperado.

Uma analogia bastante ingênua que pode ser feita é do desempenho hidráulico em vasos comunicantes. Você coloca o máximo de água possível de um lado de uma tubulação qualquer e verifica o outro lado. Dependendo da forma que esse vaso toma no meio do caminho pode demorar para que tudo que injetamos aparecer na outra ponta. Pior, a água pode ter vazado ou ter sido contaminada no meio do caminho. Aqui já temos vários dados possíveis de verificar, como a vazão, taxa de vazamento e taxa de contaminação, **métricas**, para resumir.

No caso das redes de computadores, visto a complexidade apresentada anteriormente, a avaliação é dramaticamente mais densa que o exemplo acima. O princípio é o mesmo, mas os pontos em que é possível observar, como nas diferentes camadas da Tabela 2.1 – e nos inúmeros e diferentes dispositivos que as implementam – fazem o número de métricas passíveis de coleta ser bastante alto, como oportunamente será mostrado.

Adicionalmente, por cada dispositivo e cada canal em que os dados passam de um ponto A para um ponto B, cada camada pode adicionar e/ou remover dados anexos – "cabeçalhos", a fim de fazer com que os dados originalmente injetados na rede cheguem ao destino desejado.

Além da adição e remoção constante desses cabeçalhos ao longo do caminho, devem se considerar mais dois pontos: os limites físicos dos meios de transmissão, e sua natureza falha, que podem provocar erros de transmissão em qualquer ponto, a qualquer momento; e os limites humanos, onde a programação feita nesses dispositivos pode apresentar erros de software. Sim, é possível a retransmissão desses dados originais, mas também há um **custo** associado ao fazê-lo.

Resumindo, parte da definição de avaliação de desempenho é que, através dela se obtém o resultado das observações coletadas em qualquer ponto desse sistema (medições), ou simplesmente dados de medições, **associando um custo ou uma grandeza às métricas desejadas**.

A segunda parte ainda não definida é bastante simples: após a coleta desses dados, alinhados aos objetivos desejados do sistema avaliado, definimos o quão bom ou ruim é uma

[2] Estatísticas em tempo real em http://www.arkko.com/tools/rfcstats/

determinada grandeza de uma métrica, e então avaliamos se esses números estão dentro do aceitável ou não. A avaliação em si não é objeto deste trabalho.

Por último, e mais importante, nesse trabalho avaliamos o desempenho dos bancos de dados, e não o desempenho das redes. Utilizamos a avaliação de desempenho de redes apenas como contextualizador do objetivo proposto. Essa avaliação é feita com medições fim-a-fim, com a suíte de ferramentas NetMetric, que estão apresentadas já na subseção 2.2.3.

2.2.3 Medição fim-a-fim

A medição fim-a-fim é um meio difundido para obtenção de dados de medições, como ocorre em (LAI e BAKER, 2000), (EKELIN, NILSSON, *et al.*, 2006), entre outros[3]. É simplesmente o ato de, no nó-fim de uma rede, lançar pacotes especiais para um outro nó-fim da rede, comportamento típico da comunicação via internet.

Neste caso – geralmente – não interessam dados de medição retornados ativamente por equipamentos intermediários, e sim – principalmente – a observação do comportamento desses pacotes nas comunicações feitas entre esses dois nós fim.

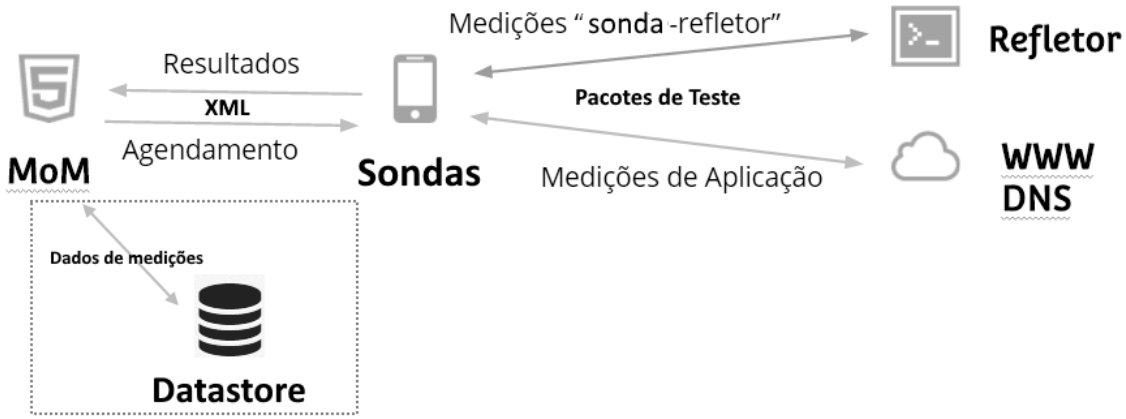
Em suma, é o ato de simular o comportamento típico de tráfego na internet e monitorar em um – ou ambos – nó-fim o que acontece com a transmissão.

Um ótimo exemplo é a observação do *RTT*, ou *Round Trip Time*, o tempo ida e volta de um pacote. Lançamos um pacote de tamanho fixo (dependendo do objetivo), e o outro nó-fim responde imediatamente com um *ACK*, ou *acknowledged*, uma mensagem simples avisando que foi recebido. Então o tempo entre o início do envio do pacote e o recebimento do *ACK* pelo mesmo nó emissor é medido.

2.2.4 NetMetric

O NetMetric é uma suíte de ferramentas de medição fim-a-fim para redes *IP*. Ele envolve desde os dispositivos, meios e padronização de transmissão nos nó-fim, até a coleta, ingestão e visualização de diversas métricas observadas nas redes medidas, um resumo pode ser visto na Figura 2.1.

Figura 2.1 - Resumo da arquitetura do NetMetric



Fonte: (DLUGOKENSKI, 2015)

[3] Basta verificar outros exemplos em http://bit.ly/dlue2e.

Todos os conceitos citados anteriormente estão relacionados no NetMetric e são usados de alguma maneira específica. É responsabilidade dos mecanismos de ingestão e visualização – o banco de dados, o *webservice* de coleta e o *webservice* de visualização – tratar adequadamente todos os tipos de dados inerentes a esses conceitos, também parte da suíte.

Justamente no contexto do NetMetric que esse trabalho se situa, e onde fora observada a necessidade de avaliar alternativas para manter, consumir e mostrar esses dados da melhor maneira possível. Dentro do NetMetric, esse trabalho se situa no destaque pontilhado da Figura 2.1.

Para que esses dados sejam observados, o NetMetric conta com uma entidade de teste chamada **sonda**. As sondas são agentes de teste que disparam diversos testes fim-a-fim, a fim de verificar mais de uma dezena de métricas, e algumas dessas métricas podem ser disparadas contra outra centena de nós da rede, como o caso do tempo de resposta contra os *top 100* sites do Alexa.

No presente momento centenas dessas sondas, estão realizando testes nas redes instaladas, com muitos testes ocorrendo a cada minuto. Aqui se começa a perceber o tamanho do desafio de manter tantos dados intactos e disponíveis para que os analistas utilizem posteriormente em seus diagnósticos e relatórios.

A seguir serão apresentadas quais tecnologias podem ser usadas para manter esses dados acessíveis.

2.3 Bancos de Dados

2.3.1 *Big Data*

O conceito básico de *big data* refere-se ao tratamento de volumes muito grandes de dados em grandes velocidades, e que, simultaneamente, tem uma ampla variedade de tipos de dados e de fontes (diversidade), além de requisitos pertinentes à veracidade dos dados (fidelidade), segundo (HAN, JIA, *et al.*, 2015).

Com os conceitos vistos nas seções 2.2.3 e 2.2.4, e imaginando que, no mínimo, centenas de sondas podem ser distribuídas em uma grande rede, torna-se perceptível que o desafio de avaliação de desempenho de grandes redes é um problema de *big data*.

No mundo *big data*, atualmente, existem muitas soluções para a enorme variedade de tipos de dados que se apresentam. Cada uma se mostra desenhada para contemplar um problema específico, e, muito comumente, juntam-se várias dessas soluções em grupo para resolver, em alto nível, um desafio de determinado produto ou serviço.

Um exemplo é do conjunto *Elasticsearch* – que faz busca e análise de termos textuais – *Logstash* – que faz normalização de dados recebidos, ambos produtos da Elasticsearch BV, mais o *Apache Hadoop*, que ou guarda dados não estruturados em um serviço de dados *Hadoop Distributed File System*, ou guarda dados estruturados em um serviço *Apache HBase*. Esse é um bom exemplo de estrutura para um serviço de análise de registros textuais (*logs*), ou de documentos XML (sites de notícias, por exemplo).

No entanto, voltando ao domínio deste trabalho, dados que são recebidos em documentos XML do NetMetric podem ser resumidos em apenas uma estrutura muito simples, que são basicamente dados numéricos ao longo de uma linha de tempo. Por exatamente ser possível estruturá-los de maneira mais simples, a necessidade real para uma solução de avaliação de desempenho de redes é outra.

Primariamente, o que se necessita é de um banco de dados que absorva e retorne esses dados brutos extraídos da maneira mais rápida possível, equivalente a operações de *CREATE e READ* na terminologia de bancos de dados.

Portanto, neste trabalho, todas outras soluções de *big data* que ajudam a modelar análises e fazer buscas de comportamentos específicos dos dados deve ser visto com outra abordagem,

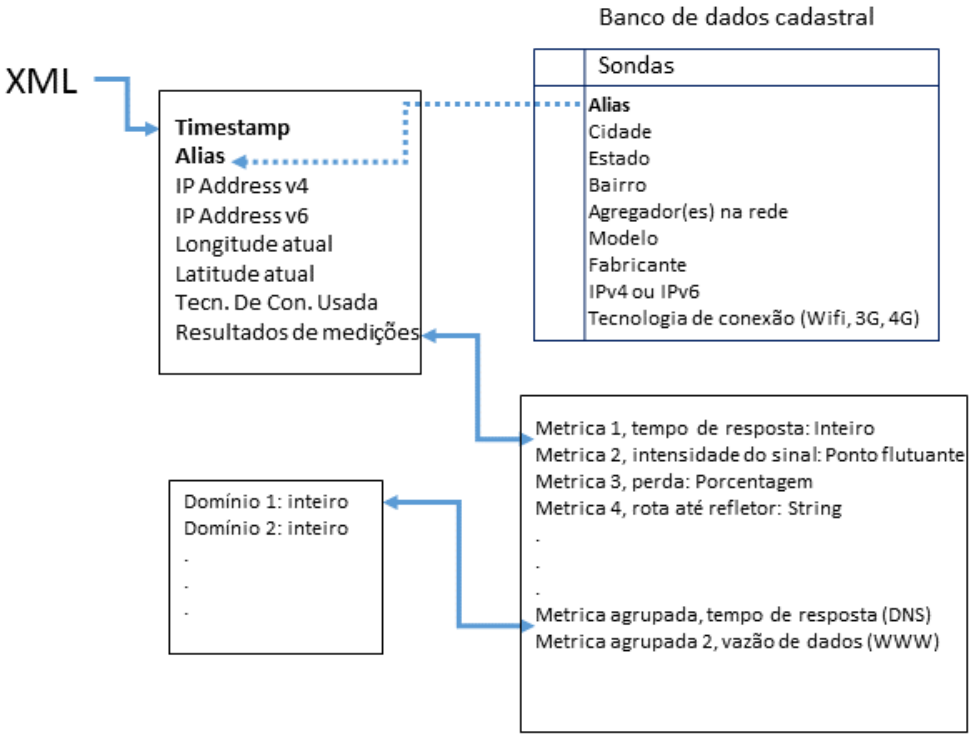
em outros trabalhos. Nas seções seguintes, por fim, verifica-se, após as modelagens, que por haver uma estrutura de dados bem-comportada, podemos usar ou bancos de dados relacionais, ou bancos especializados em séries de tempo.

2.3.2 Dados do NetMetric em alto nível

Antes de serem apresentados os conceitos dos bancos de dados propostos para esse trabalho, deve-se verificar a natureza e grandeza da maior parte desses dados fornecidos pelo NetMetric. Assim, nesta seção está descrito uma ideia geral do formato em que os dados são entregues por cada sonda a um centralizador de dados (implementado como um *webservice*).

Em geral, as sondas do NetMetric entregam um documento XML, fixo em um ponto qualquer de uma linha do tempo (um *timestamp*), com diversos tipos de números representando as grandezas das métricas, ou do tipo ponto flutuante, ou do tipo inteiro, ou, em alguns casos, texto puro (*strings* longas), resumido na Figura 2.2.

Figura 2.2 - Resumo de um documento XML do NetMetric



Fonte: (DLUGOKENSKI, 2015)

Esse documento, que é resultado de diversos testes em uma única sonda, pode conter de dezenas até centenas desses números. A sonda envia um documento nesse formato para o *webservice*, que fica responsável por extrair os dados e enviar ao banco de dados.

Após esse passo, é importante lembrar que os dados não devem sofrer nenhum tipo de abreviação ou interpolação, para que futuras análises dos dados não sofram excessivamente com variações impostas pela tecnologia escolhida para armazená-los.

2.3.3 Bancos Relacionais (SQL, NewSQL)

Os bancos de dados relacionais referidos nesse trabalho são sistemas de gerenciamento de bancos de dados inspirados no modelo relacional de Codd (CODD, 1970). A maioria deles teve influência do IBM System R, projeto do fim da década de 70, que utilizou do modelo de

Codd para montar um protótipo de um sistema de banco de dados relacional, e este, por sua vez, culminou em dois produtos comerciais: o IBM DB2 e o Oracle, segundo (SUMATHI e ESAKKIRAJAN, 2006, p. 6).

A maioria dos sucessores, incluindo o Oracle moderno, e alternativas, como o MySQL e o PostgreSQL, implementam a visão de várias tabelas, com colunas (propriedades e atributos), e linhas (os dados em si, ou tuplas). Estas tabelas se inter-relacionam através de chaves estrangeiras, e utilizam a linguagem SQL, que possui operadores relacionais para fazer o nexo entre os dados dessas diversas tabelas (SUMATHI e ESAKKIRAJAN, 2006, p. 67-70).

Outras características notáveis são a obrigatoriedade de uma estrutura previamente montada (*statically typed schema)* e fortes garantias ACID.

A primeira pode ser boa ou pode ser ruim, pois ao mesmo tempo que oferece garantias extras de consistência de acesso, pois o atributo (coluna) sempre existe, mesmo que vazio, simultaneamente exige a programação de uma propriedade raramente utilizada, ou desnecessária dependendo do contexto, além de um fino planejamento prévio, pois o *schema* deve prever explicitamente o atributo.

Já a segunda característica, das garantias ACID, ou melhor - Atomicidade, Consistência, Isolamento, Durabilidade, está presente nos três bancos modernos citados anteriormente, e passa uma confiança grande para o arquiteto do sistema de que os dados estão sempre em um estado seguro, resumindo (SUMATHI e ESAKKIRAJAN, 2006, p. 319-335).

Finalmente, sobre os bancos "*NewSQL*". Estes são de implementação recente, inclusive após o surgimento do movimento NoSQL (visto na próxima seção), onde eles mantêm as garantias ACID e o modelo relacional com performance similar à de bancos não relacionais, usando métodos diferentes de recuperação e controle de concorrência em relação aos inspirados no IBM System R, além de usar o cenário atual em que a computação distribuída (nuvem) está acessível a todos, refletindo isso no seu design, como visto por (GROLINGER, HIGASHINO, *et al.*, 2013, p. 4-5,7,10).

2.3.4 Bancos não relacionais (NoSQL)

Em contraponto aos bancos relacionais tradicionais, os quais foram encontrando muitos desafios com a realidade *Big Data* contemporânea, em que alta velocidade de ingestão, ampla variedade de estruturas e grande volume de dados são requisitos, começaram a surgir questionamentos e implementações de soluções alternativas (GROLINGER, HIGASHINO, *et al.*, 2013, p. 1,2).

Um questionamento relevante que ocorreu refere-se ao teorema *CAP* (*consistency, availability and partition tolerance*), uma conjectura que diz que apenas duas dessas três propriedades podem ser satisfeitas simultaneamente em ambientes de dados compartilhados (BREWER, 2000). Trabalhos como (BREWER, 2012), do mesmo autor, revisitaram esse tema e constataram que na verdade podemos ajustar diferentes níveis nesse tripé, conforme a necessidade arquitetural.

Assim, surgiu a sigla BASE (*Basically Available, Soft state, Eventual consistency*), que antagoniza com a proposta ACID, vista na seção anterior. Como percebe-se em (PRITCHETT, 2008, p. 51-53), que cunhou a sigla, há como se trabalhar com consistência eventual, que ao invés de travar uma grande quantidade de dados ao inserir ou atualizar dados, como nas soluções tradicionais, faz isso através de um controle mais otimista, que se traduz em uma performance percebida maior.

Simultaneamente, na prática, diversas soluções começaram a ser desenvolvidas no atual boom das empresas "*ponto-com"* e da onipresença da nuvem computacional. Organizações como a Apache (com o HBase), a Amazon (com o Dynamo) e o Google (com o BigTable) já vinham trabalhando desde a década passada na manipulação desse tripé, para oferecer os requisitos da era *big data*, como vê-se em (CHANG, DEAN, *et al.*, 2008). O resultado que se

chegou é que essa proposta funciona, com suas devidas limitações consideradas (BAILIS e GHODSI, 2013), assim o movimento NoSQL (*Not Only SQL)* encontrou muitos usos e é atualmente parte do alicerce das empresas citadas acima.

Na questão das limitações citadas em (BAILIS e GHODSI, 2013, p. 3-4), um exemplo é a questão da atualização de dados e a checagem de unicidade, onde dados podem ser inseridos mais de uma vez pela aplicação em caso de haver particionamento dos servidores de dados. No NetMetric todos os dados resultantes dos testes são inseridos apenas uma vez, sem modificações posteriores, e a unicidade não é um requisito, já que a o *webservice* de ingestão recebe somente uma vez o documento XML de resultados.

Ainda há uma vantagem de se usar essa abordagem. Normalmente os bancos não relacionais também relaxam a questão do *schema* previamente definido – oferecendo estratégias do tipo *dynamic schema* ou *schemaless* (GROLINGER, HIGASHINO, *et al.*, 2013, p. 4). Portanto em um ambiente com métricas bastante distintas, com atributos variados, e que, dependendo dos dispositivos usados nos testes, variam de um para outro, não há a necessidade de paradas para planejamento e reconstrução do *schema*.

Um exemplo visto no NetMetric da dificuldade acima descrita, é do uso indevido de campos com semântica diferente ou com nome vago, como o campo 'upavg', que na atual implementação do NetMetric tem diferentes significados para vazão (média da taxa de bits enviados) e para o tempo ida-e-volta (média de tempo bidirecional) – isso foi feito para evitar grandes custos de manutenção e de escrita de código. Ou exemplo é que métricas simples como o tempo ida-e-volta para um refletor exija minimamente cinco atributos (*timestamp*, origem, destino, tempo, ipv4/ipv6, wifi/3g/fibra), e o tempo de resposta DNS exige mais quatro (resposta, ok/nok, ip do dns, tcp/udp), e assim sucessivamente para outras métricas, os quais começam a apresentar atributos diferenciados.

2.3.5 Bancos de Séries Temporais (Time Series)

O conceito de utilizar softwares específicos para guardar eventos ao longo do tempo não é novo. A indústria já encontrava soluções proprietárias sob a alcunha de *operational historian.* Esses softwares, das mais respeitadas empresas globais, como *General Electric (Proficy Historian), Schneider Electric (Wonderware Historian), National Instruments (Citadel), Honeywell (Uniformance)*, entre outros, já estavam disponíveis muito antes da atual corrente *open-source* de *TSDBs (Time Series Databases)*.

Mas como as informações desses softwares proprietários são bastante limitadas, e a propaganda comercial envolve somente usos industriais, este trabalho não vai poder sublinhar essas diferenças para os novos softwares *open source*, como o OpenTSDB, o Atlas (do NetFlix), o InfluxDB, etc.

Falando nestes, em geral são baseados em *datastores* NoSQL, como o Cassandra (do Facebook – ex. KairosDB[4]), ou o Apache HBase (OpenTSDB[5]), ou ainda soluções caseiras, como o InfluxDB[6] (com motor próprio em linguagem Go), ou ainda puro C++, no caso do Akumuli[7].

Nas documentações destas últimas soluções, especialmente em (DIX, 2015) e (OPENTSDB, 2015), é possível verificar que seu design é completamente voltado para um tipo de tupla: `(timestamp, métrica, valor, marcadores).` Os dados são guardados em arquivos organizados em janelas do tempo, sem sobreposição temporal entre arquivos, o que faz o uso de soluções NoSQL do tipo chave/valor (Apache HBase, BoltDB) e do tipo colunar (Facebook Cassandra), que utilizam o teorema CAP revisado, serem até desejados. **É a tupla**

[4] No site https://github.com/kairosdb/kairosdb é possível verificar essas informações
[5] No site http://opentsdb.net/ é possível verificar essa afirmação
[6] Mais em https://influxdata.com/time-series-platform/influxdb/
[7] Informações em http://akumuli.org/

acima que o NetMetric mais utiliza nos dados de teste, e, muito provavelmente, outras aplicações de teste de redes.

Além disso, há a preocupação de utilizar os mais recentes avanços em compressão, com o Snappy[8], ou o LZO[9], que prometem uma compressão de dados justa e muito rápida (GOOGLE INC., 2016).

Outro destaque é o design voltado para resolução simplificada de conflitos (INFLUXDATA INC., 2016), onde podemos retirar de uma característica do NetMetric, e possivelmente outras soluções de monitoramento de rede. Cada sonda (ou sensor) envia dados pertinentes somente a si, e nunca conflita com dados de outras sondas. A inserção também é única, sem atualizações posteriores, dispensando uma solução pessimista de resolução de conflitos ou de ordenamento de dados.

Ainda, na tabela Tabela 2.2, há o resumo dos *trade-offs* escolhidos no *design* do InfluxDB, retirados da documentação oficial, e se estes impedem um sistema de monitoramento como o NetMetric. Estas escolhas são normalmente feitas em outros bancos de séries de tempo.

Tabela 2.2 - Prós e contras das escolhas tomadas no InfluxDB e sua influência na arquitetura do NetMetric

Característica	Prós	Contras	Impedimentos?
Se o mesmo ponto do tempo é reenviado, ele deve ser exatamente o mesmo.	Solução de conflito simplificada.	Dados divergentes são rejeitados.	**Nenhum, é um benefício**
Remoção de pontos são raros.	Essa restrição permite otimizar a escrita e consulta.	Remoção de dados é muito limitada, e deve ser feita em janelas de tempo largas.	**Nenhum**
Atualizações de dados existentes são raros.	Mesmo acima.	Mesmo acima.	**Nenhum, não há atualizações**
A maioria dos dados inseridos é de pontos recentes na linha do tempo.	A inserção é mais rápida, e algoritmos otimistas de ordenação podem ser usados.	Escrever pontos aleatórios tem desempenho ruim.	**Nenhum, é exatamente como ocorre no NetMetric**
Escalabilidade. Voltado para altos volumes.	Aguenta muitos dados.	Simplificações de vários recursos são necessárias.	**Nenhum**
A habilidade de escrever e consultar é prioridade em relação a consistência de visualização.	Mesmo acima, além de não travar a consulta.	Dados já enviados podem ser só vistos após alguns segundos/minutos.	**Talvez. Mas o tempo gasto em soluções tradicionais atrasa a consulta na mesma escala.**
Nenhum ponto é muito importante sozinho.	A solução pode se focar em grandes conjuntos de dados e agregações.	Não há uma chave primária, o mecanismo diferente deve ser levado em conta na aplicação.	**Nenhum, as consultas são tipicamente por janela de tempo, métrica e alguns marcadores**

Fonte: Adaptado de (INFLUXDATA INC., 2016), com última coluna por (DLUGOKENSKI, 2016)

2.3.6 RRD – Round Robin Database

Os bancos do tipo *RRD*, representado exclusivamente pelo RRDtool, é um tipo de ferramenta para dados numéricos guardados em intervalos fixos, com intervalo de durabilidade também fixo, que é bastante famosa no meio de gerência de redes e normalmente cogitado em situações como a do NetMetric.

[8] https://github.com/google/snappy
[9] http://www.oberhumer.com/opensource/lzo/

Esse tipo de banco de dados reserva um espaço fixo de dados, onde eles são guardados de maneira encadeada (como em uma lista encadeada), onde dados muito velhos são descartados ao término do espaço reservado, abrindo lugar para os mais novos, conforme documentação oficial da ferramenta (OETIKER, 2015).

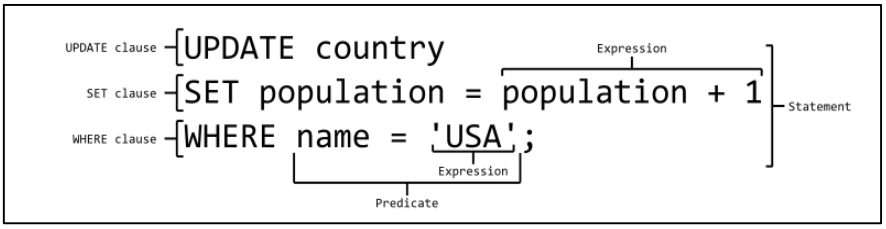
2.4 Linguagens e acesso aos dados

2.4.1 SQL

SQL, ou *Structured Query Language*, ou ainda Linguagem de Consulta Estruturada, é uma linguagem declarativa de interação com bancos de dados, padrão nos bancos de dados relacionais e baseado no cálculo relacional ou álgebra relacional, segundo (SUMATHI e ESAKKIRAJAN, 2006, p. 111,112). Um exemplo de uso pode ser visto na Figura 2.3.

Ela é utilizada neste trabalho para fazer as inserções e consultas no PostgreSQL.

Figura 2.3 - Diversos elementos da linguagem SQL que compõem uma afirmação



Fonte: (WIKIPEDIA, 2015)

2.4.2 InfluxQL

O *InfluxQL* é um subconjunto da linguagem *SQL*, utilizado para consultas (exclusivamente) no InfluxDB. Ele prevê características específicas para análise de dados em uma série temporal, conforme documentação oficial (INFLUXDATA INC., 2015). Um exemplo está na Figura 2.4, onde consultamos os dados dos últimos 7 dias.

Figura 2.4 - Um exemplo de consulta no InfluxQL

```
SELECT water_level FROM h2o_feet WHERE time > now() - 7d


name: h2o_feet
----------
time                   water_level
2015-08-18T00:18:00Z   7.762
2015-08-18T00:30:00Z   7.5
```

Fonte: (DLUGOKENSKI, 2016), adaptado de (INFLUXDATA INC., 2015)

2.4.3 Influx Line Protocol

O *Influx Line Protocol* é uma linguagem para inserção de dados específico do banco de dados *InfluxDB* (INFLUXDATA INC., 2015). Específica para inserção de dados em séries de tempo, prevê para inserção somente a tupla `(métrica, marcadores, valor(es), timestamp)`, à exemplo da Figura 2.5.

Figura 2.5 - Exemplo de inserção no InfluxDB usando o Line Protocol

```
#Métrica,tag1,tag2,… valor1,valor2,… timestamp
cpu,host=server01,region=uswest value=1
1434055562000000000
temperature,machine=unit143,type=assembly
internal=22,external=130 1434055562005000035
```

Fonte: (DLUGOKENSKI, 2016)

2.4.4 XML

XML, ou *Extensible Markup Language*, ou ainda Linguagem de Marcação Extensível, é uma linguagem de marcação de documentos, que define regras de codificação e de formatação de documentos que devem ser simultaneamente legíveis por humanos e fáceis de processar em programas de computador, e é definido em um padrão aberto da *W3C*, conforme (W3C, 2008). Um exemplo de documento está na Figura 2.6.

Esta linguagem é utilizada no NetMetric no processo de envio de resultados de teste para o *webservice* de coleta, posteriormente processada e enviada ao banco de dados.

Figura 2.6 - Exemplo de representação XML para livros

```
<Books>
    <Book ISBN="0553212419">
        <title>Sherlock Holmes: Complete Novels...
        <author>Sir Arthur Conan Doyle</author>
    </Book>
    <Book ISBN="0743273567">
        <title>The Great Gatsby</title>
        <author>F. Scott Fitzgerald</author>
    </Book>
    <Book ISBN="0684826976">
        <title>Undaunted Courage</title>
        <author>Stephen E. Ambrose</author>
    </Book>
    <Book ISBN="0743203178">
        <title>Nothing Like It In the World</title>
        <author>Stephen E. Ambrose</author>
    </Book>
</Books>
```

Fonte: Imagens do Google, Autor desconhecido

2.4.5 Ruby

O Ruby é uma linguagem de programação de uso geral, de tipagem dinâmica e de paradigma múltiplo (funcional, orientado à objetos e imperativo), com uma gramática bastante expressiva e uma biblioteca padrão rica e poderosa, traduzindo (FLANAGAN e MATSUMOTO, 2008, p. 2).

A sua última versão (2.3.0) vai ser utilizada nos *scripts* de conversão, consultas e inserção de dados. Também nesses *scripts* o Ruby vai medir os tempos de execução das consultas nos bancos de dados. Na Figura 2.7 vemos um exemplo de um trecho de código Ruby.

Figura 2.7 - Exemplo de redefinição de uma classe em Ruby

```
class Time
  def yesterday
    self - 86400
  end
end

today = Time.now
# => 2013-09-03 16:09:37 +0300
yesterday = today.yesterday
# => 2013-09-02 16:09:37 +0300
```

Fonte: (DLUGOKENSKI, 2016)

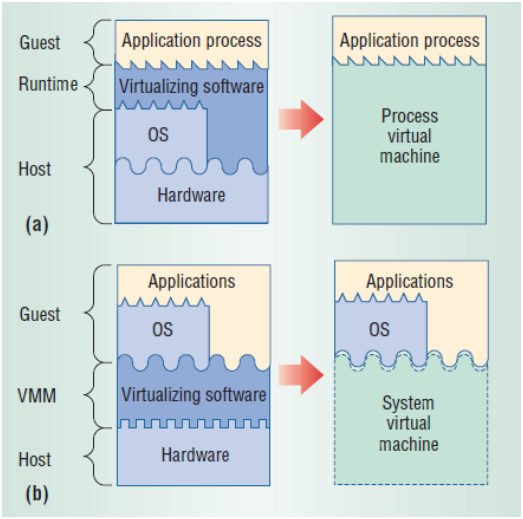
2.5 Ambiente

2.5.1 Máquinas Virtuais

Virtualização é um meio de mapear recursos (processador, memória, dispositivos E/S) de um sistema operacional em execução, mas com um nível de abstração tal que os subsistemas virtualizados percebam falsamente exclusividade de acesso (SMITH e NAIR, 2005, p. 32,33). Cada máquina virtual, portanto, é um subsistema praticamente isolado de outras máquinas, especialmente da máquina hospedeira (que tem acesso direto aos recursos de *hardware*), a exemplo da Figura 2.8.

O principal benefício neste trabalho é que se possa isolar as configurações de um serviço, principalmente da máquina hospedeira. Com essa característica podemos criar máquinas com comportamento múltiplas vezes reproduzível (rodar da maneira esperada diversas vezes), com redução drástica das inspeções e modificações em arquivos de configuração, evitando que erros de configuração se propaguem entre os diferentes serviços testados.

Figura 2.8 - Exemplo de virtualização de processo (a) e de sistema (b)



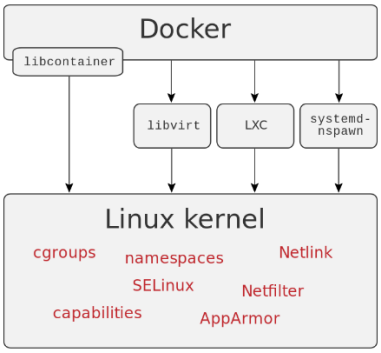
Fonte: (SMITH e NAIR, 2005, p. 34)

2.5.2 Máquinas Docker

A virtualização Docker é uma implementação de máquinas virtuais que utiliza recursos do *kernel* do Linux para virtualizar o próprio *kernel*, assim as aplicações da máquina virtual

percebem acesso ao sistema operacional Linux de maneira isolada de outras máquinas Docker, conforme Figura 2.9 (HYKES, 2014).

Figura 2.9 - Modelo de virtualização do Docker



Fonte: (HYKES, 2014)

Elas têm um nível de abstração diferente das máquinas virtuais tradicionais, pois não buscam virtualizar o hardware completo, somente o acesso ao *kernel*, assim são consideradas mais leves – sua inicialização ocorre na casa dos milissegundos tipicamente. São praticamente iguais ao conceito de máquina virtual de processo, item (a) da Figura 2.8.

3 SISTEMAS DE BANCO DE DADOS

3.1 Exemplos e candidatos considerados

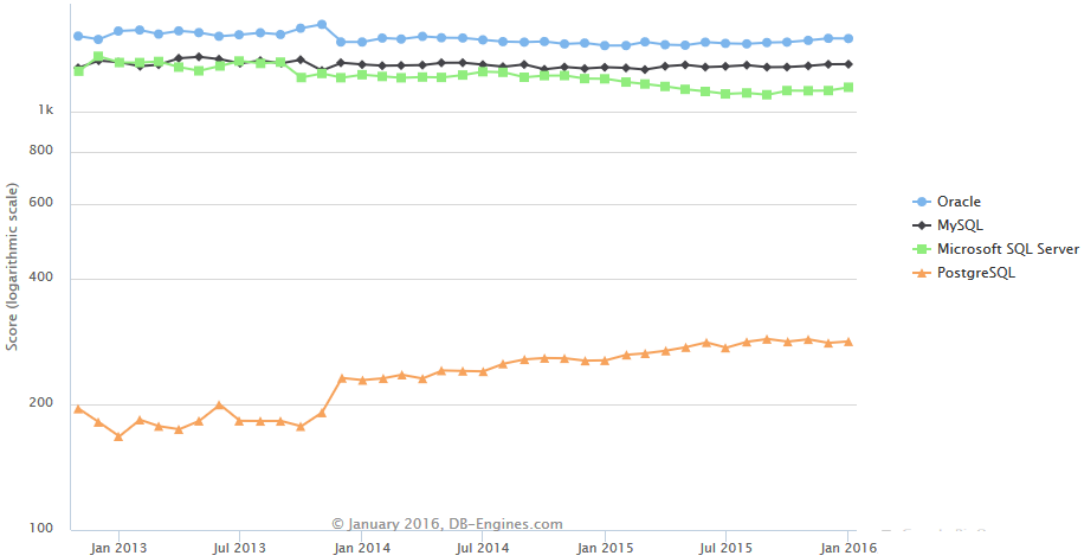
Várias possibilidades já foram citadas no capítulo anterior, serão enumeradas boa parte delas, com um pequeno histórico de cada uma.

Como a quantidade de candidatos é enorme, foi utilizado o ranking do site http://db-engines.com, de (SOLID IT GMBH, 2016), para escolher os quatro mais populares de cada categoria, visto no dia 08 de janeiro de 2016. Esse site possui um claro método de cálculo citado no próprio site.

3.1.1 Bancos Relacionais (RDBMS)

Os quatro mais populares bancos de dados relacionais, segundo o ranking de (SOLID IT GMBH, 2016), e resumidos na Figura 3.1, são:

Figura 3.1 - Os quatro bancos de dados relacionais mais populares



Fonte: (SOLID IT GMBH, 2016)

- **Oracle** (http://www.oracle.com/br/database)

Um dos primeiros bancos de dados relacionais, o Oracle ainda persiste com o domínio do mercado tradicional de bancos de dados relacionais, conforme análise do Gartner Research Group (FEINBERG, ADRIAN, *et al.*, 2015). Esse banco de dados é proprietário e exige licenças de uso.

- **MySQL** (http://www.mysql.com/)

Outro banco de dados que ganhou bastante popularidade com a Internet, o MySQL é um banco de dados *open source*, atualmente mantido pela Oracle, que vende suporte técnico. Utiliza por padrão um motor considerado moderno, o InnoDB.

- **Microsoft SQL Server** (http://bit.ly/dlumssql)

Outro banco de dados relacional proprietário, agora da Microsoft, possui diversas edições diferentes otimizadas para diferentes tipos de usos, seja para guardar dados de aplicações locais, seja de aplicações na nuvem.

- **PostgreSQL** (http://www.postgresql.org/)

É um dos bancos de dados *open source* mais antigos do mercado, auto intitulado "o mais avançado banco de dados *open source*", alegando que suportam a maioria das características previstas pelo padrão SQL:2011 da ISO, além de inúmeros tipos específicos de dados, incluindo endereços *IP*, *UUIDs* e *JSON*.

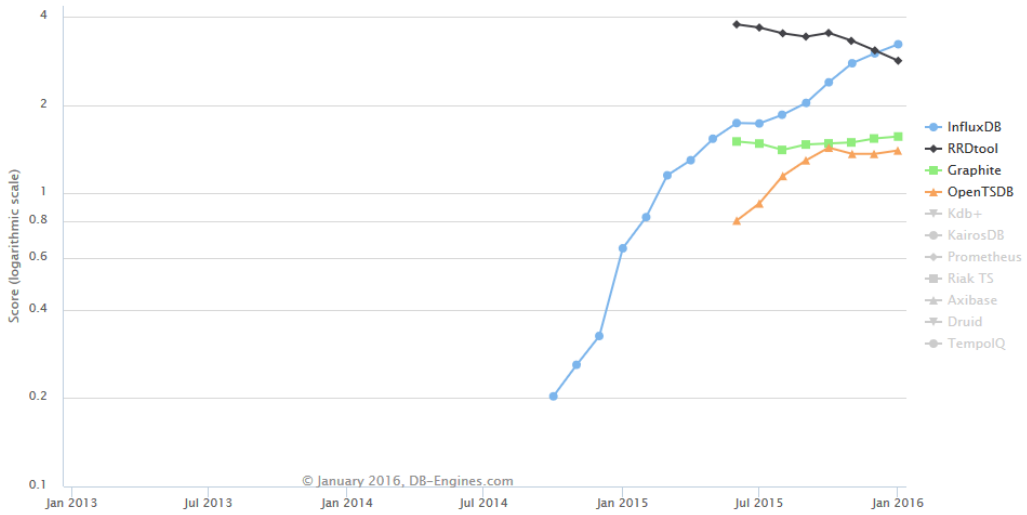
3.1.2 Bancos não relacionais (NoSQL)

Não será considerado nenhum banco deste tipo neste trabalho, apesar de apresentados os conceitos na seção 2.3.4, já que haverá candidatos nos bancos especializados de séries temporais que utilizam bancos não relacionais como ponto de partida ou como subsistema de dados.

3.1.3 Bancos de Séries Temporais (TSDBs)

Os quatro mais citados bancos de dados de séries temporais, conforme o ranking de (SOLID IT GMBH, 2016), e resumidos na Figura 3.2, são:

Figura 3.2 - Popularidade dos bancos de dados de séries temporais



Fonte: (SOLID IT GMBH, 2016)

- **InfluxDB** (https://influxdata.com/time-series-platform/influxdb/)

O InfluxDB é um banco de dados *open source* específico para lidar com dados de séries temporais, prometendo simplicidade de instalação, com poucas dependências, e ainda flexível, com alta escalabilidade para desdobramentos mais complexos. Utiliza o banco NoSQL BoltDB como subsistema para guarda dos dados.

- **RRDtool** (http://oss.oetiker.ch/rrdtool/)

O RRDtool é outro banco de dados *open source*, para registros de eventos de alta performance em séries de tempo. Diferentemente dos demais, ele não é baseado em subsistemas

NoSQL, e implementa um *storage* com tamanho pré-definido, que descarta dados antigos após o término desse espaço de guarda de dados, como em uma lista encadeada FIFO.

- **Graphite** (https://graphite.readthedocs.org/)

O Graphite é uma ferramenta de coleta de dados que promete rodar em hardware barato, conta com um renderizador de gráficos embutido, que promete visualização dos dados em tempo real, e também com um motor de guarda de dados especializado.

- **OpenTSDB** (http://opentsdb.net/)

Segundo o site oficial, o OpenTSDB é um banco de dados de séries temporais que guarda os dados exatamente como foram enviados (sem perdas de granularidade). Ele utilizou inicialmente o banco NoSQL Apache HBase como motor de guarda de dados, mas também suporta agora o Cassandra e o BigTable.

3.2 Critérios de Seleção

Nesta seção vamos enumerar as características desejáveis, que basicamente estão na Tabela 2.2, na página 28; verificar quais candidatos as apresentam, e, por fim, escolher um candidato de cada tipo que apresentem o maior número delas.

Um detalhe importante para ser lembrado é que as possibilidades de implementação são tantas, com tanta teoria envolvida (basta olhar excelentes trabalhos como (SHASHA e ZHU, 2013) e (ANDRÉ-JÖNSSON, 2002)), que não serão discutidos aqui os detalhes de cada um dos recursos.

3.2.1 Recursos obrigatórios, desejados e oferecidos

Nesta subseção serão comparados recursos oferecidos diretamente pelos softwares, desde aqueles necessários, até aqueles que são desejados para o NetMetric.

3.2.1.1 Controle de concorrência otimista

O intuito em recursos de controle de concorrência otimistas é que os dados de monitoramento de redes são basicamente únicos, e virtualmente sem conflitos de inserção e com nenhuma atualização. E no fim, só precisamos do máximo de desempenho possível na ingestão, além de uma visualização consistente, mas não necessariamente a mais nova (falando em escala de segundos e talvez poucos minutos). Logo esse é um dos recursos mais desejados.

Os candidatos do tipo TSDB utilizam esse tipo de controle por padrão (exceto o RRDtool), já os candidatos do tipo relacional também oferecem, mas não necessariamente por padrão. A maneira como foi implementada e a aderência as propriedades ACID também influenciam no desempenho, mas não será discutido nesse trabalho os motivos de cada um. Confira o resumo na Tabela 3.1 para uma melhor visualização.

Tabela 3.1 - Qualificações e referências relativos ao controle de concorrência

Candidato	Qualificado	Nome	Referência
Oracle	Sim	MVCC	(KAPILA, 2015)
MySQL	Sim	MVCC	(KAPILA, 2015)
Microsoft SQL	Sim	Optimistic CC	(KAPILA, 2015)
PostgreSQL	Sim	MVCC	(KAPILA, 2015)
InfluxDB	Sim	Parallel Writes	(DIX, 2015)
RRDtool	Não	WAL	RRDcached[10]
Graphite	Não	-	Não encontrada
OpenTSDB	Sim	MVCC, WAL	(CHANAN, 2013)

Fonte: (DLUGOKENSKI, 2016)

3.2.1.2 Indexação por sequência de blocos ou acesso por aproximação

Para maximizar o acesso em uma grande quantidade de dados ordenada por *timestamp*, é bastante desejável que existam opções de indexação em blocos de dados. As atuais implementações dos bancos relacionais não oferecem um grande desempenho nesse tipo de situação por motivos ignorados, mas que provavelmente não vislumbraram o uso em grandes quantidades de dados dessa natureza. Esse acesso também não precisa ser exato, pois com a quantidade de dados é necessário o descarte prévio de grandes áreas, que pode ser feito por aproximação, relacionando *timestamp* (final e inicial) e localização no banco.

Um exemplo que surgiu no momento de concepção desse trabalho são os índices BRIN do PostgreSQL versão 9.5. Nos testes apresentados em (POSTGRESQL WEB TEAM, 2016), o desempenho obtido é cinco vezes maior que com índices com árvores B simples em um conjunto de dados de apenas 13 GB. Portanto é mais que desejável esse tipo de recurso.

Porém, a grande diferença está em como os bancos especializados fazem isso. É possível fazer o acesso por aproximação ou de maneira exata, levando em conta o fato de que os dados são inseridos de maneira praticamente ordenada e de maneira imutável. No caso da aproximação é possível utilizar árvores B+ para guardar a localização de grandes blocos de tempo. A análise da documentação dos candidatos está na Tabela 3.2.

Tabela 3.2 - Qualificações e referências relativos a indexação por bloco

Candidato	Qualif.	Método	Referência
Oracle	Sim	*B-Tree Cluster*	(ORACLE, 2015)
MySQL	Não[11]	*Clustered Index*	(ORACLE, 2015)
Microsoft SQL	Não[12]	*Clustered Index*	(MICROSOFT, 2016)
PostgreSQL	Sim	*BRIN Index*	(POSTGRESQL WEB TEAM, 2016)
InfluxDB	Sim	*B+ Tree Block*	(INFLUXDATA INC, 2015)
RRDtool	Sim[13]	*Cálculo Exato*	(OETIKER, 2015)
Graphite	Sim	*Aprox. Calculada*	(GRAPHITE PROJECT, 2015)
OpenTSDB	Sim	*B+ Tree Block*	(MCDONALD, 2015)

Fonte: (DLUGOKENSKI, 2016)

[10] Apenas é citado algum tipo de controle de concorrência aqui: http://oss.oetiker.ch/rrdtool/doc/rrdcached.en.html#HOW_IT_WORKS

[11] O InnoDB do MySQL obriga a criação de um índice único comum, ou ele usa uma identificação do número da linha física, que não torna explícita/inferível a optimalidade para séries temporais.

[12] Somente são criados clusters em chaves primárias sem repetição. Os *timestamps* de uma série temporal deste trabalho são potencialmente repetíveis e localmente próximos.

[13] Os dados no RRD têm tamanho fixo e inserção ordenada imutável, portanto é factível a localização exata por álgebra simples.

3.2.1.3 Não alterar os dados de maneira implícita

A malha de sondas de teste do NetMetric é muito grande, com resultados sendo enviados de maneira bastante caótica, com *timestamps* apresentados em qualquer segundo, apesar dos testes serem realizados, em média, a cada 15 minutos em cada sonda. É imperativo que os dados sejam somente armazenados, para que não haja influências negativas em dados derivados e relatórios.

Aqui temos apenas dois candidatos **desqualificados**, mas de maneira bastante categórica. O **RRDtool** altera o valor usando uma curva de tendência de pontos anteriores, armazenados em um *timestamp* discretizado próximo (uma espécie de arredondamento do tempo em segmentos), fazendo com que basicamente nenhum valor de métrica do NetMetric seja real, e sim um valor tendencioso. Já o **Graphite** simplesmente coloca em um dos *timestamps* disponíveis o valor sem alteração, uma espécie de escorregamento do tempo (*time-shifting)*, que, apesar de menos grave, é indesejável. Uma comparação de ambos métodos pode ser encontrada em (TRUBETSKOY, 2015).

3.2.1.4 Armazenamento de strings

É desejável que sejam guardados valores em formato *string*, como registros de teste do *traceroute*, e aqui ficam **desqualificados** o **RRDtool**, o **Graphite** e o **OpenTSDB**, que guardam apenas números inteiros ou de ponto flutuante.

3.2.1.5 Ordenamento otimista na inserção

É desejável inserir os dados de maneira ordenada, já que temporalmente os dados recebidos ficam próximos da ordenação ideal. Dos dados recebidos fora de ordem, normalmente estes afetam poucos dados inseridos anteriormente, muitas vezes estão no mesmo bloco, ou no máximo em um bloco anterior. Assim é desejável a inserção ordenada, mas com um reordenamento esporádico otimista.

Aqui nenhum dos bancos relacionais permite controlar a inserção dos dados, já nos bancos especializados somente o **InfluxDB** citou a existência dessa característica (INFLUXDATA INC., 2016).

3.2.1.6 Conclusão

Nesta seção restaram apenas o **Oracle**, o **PostgreSQL**, o **InfluxDB** para os recursos obrigatórios. Ainda se inclui o **OpenTSDB** caso se dispense os recursos desejados. No caso do Oracle há dúvidas quanto se a implementação do *B-Tree Cluster* é boa para o caso do NetMetric, que, por falta de documentação detalhada, não há como afirmar sem testes práticos.

3.2.2 Facilidade de instalação ou de desdobramento

Para que os testes sejam feitos de maneira fácil, na mesma máquina, e sem contaminações (de arquivos de configuração, de bibliotecas conflitantes ou de alteração do sistema operacional) o ideal é que sejam disponibilizadas imagens em máquina virtual Docker, como explicado na seção 2.5.2, da página 31. Assim é possível fazer o *deployment* (desdobramento) em tempo mínimo, testar e deixar a máquina de teste limpa para o próximo candidato.

Assim, foi dado preferência aos candidatos que disponibilizam máquinas virtuais Docker, relacionados na Tabela 3.3.

Tabela 3.3 – Disponibilidade de imagens Docker dos candidatos avaliados

Candidato	Dispon.	Link
Oracle	Não[14]	https://github.com/wscherphof/oracle-12c
MySQL	Sim	https://hub.docker.com/_/mysql/
Microsoft SQL	Não	-
PostgreSQL	Sim	https://hub.docker.com/_/postgres/
InfluxDB	Sim	https://hub.docker.com/_/influxdb/
RRDtool	Sim	https://hub.docker.com/r/haineault/rrdtool/
Graphite	Sim	https://hub.docker.com/r/hopsoft/graphite-statsd/
OpenTSDB	Sim	https://hub.docker.com/r/cloudflare/opentsdb/

Fonte: (DLUGOKENSKI, 2016)

Um outro detalhe a ser considerado é no caso de se desejar escalar a múltiplos servidores. As máquinas Docker permitem serem replicadas de maneira igual em todas máquinas físicas. Configura-se uma vez e aplica-se a mesma configuração em todas as demais.

Conclusão: Juntando os candidatos escolhidos da seção anterior, ficamos apenas com o **PostgreSQL** e o **InfluxDB**, talvez o OpenTSDB se relaxada a questão de armazenar valores do tipo *string* (seção 3.2.1.4). O Oracle também não foi desqualificado, mas por exigir um passo de construção da imagem trabalhoso, e por questões de licenciamento, foi abandonado como candidato.

3.2.3 Facilidade de configuração, de uso e de manutenção

Como no passo anterior a preferência pelo InfluxDB e pelo PostgreSQL ficou clara, nesta seção vamos avaliar somente a documentação de ambos. O Oracle e o OpenTSDB serão avaliados superficialmente.

Preferencialmente, todos os quesitos abaixo devem estar relacionados em suas documentações oficiais, acessíveis pelos *sites* oficiais citados na seção 3.1.

3.2.3.1 Documentação

A documentação é a principal arma para que as possibilidades de configuração, de casos de uso e de manutenção sejam viáveis.

Nesse quesito o **PostgreSQL** tem uma documentação básica bastante grande, e uma *wiki* com casos de uso e dicas de performance, além de inúmeros livros publicados.

Já o **InfluxDB**, por ser um aplicativo de nicho, não demanda uma documentação gigantesca, e o site oficial da sua documentação é bastante suficiente para fazer uso dessa ferramenta, especialmente no caso de uso desse trabalho.

Citando o Oracle e o OpenTSDB, ambos possuem uma vasta biblioteca de documentos. No caso do Oracle não fora encontrado um caso de uso parecido com o deste trabalho, que são muito genéricos, mas com quantidade à par do PostgreSQL. O OpenTSDB já conta com uma documentação muito boa, melhor que a do InfluxDB, devido a estar mais tempo no mercado.

3.2.3.2 Obsolescência dos dados

Outro quesito de manutenção é a possibilidade de obsolescência e até descarte de dados muito antigos. No NetMetric só faz sentido guardar dados brutos das medições por um ano ou dois, até menos para métricas específicas, como saídas do *traceroute*. Após isso não há necessidade de análises profundas dos dados originais.

Nesse aspecto, o **InfluxDB** possui duas características específicas para esta situação: *continuous queries* e *retention policy*. A primeira são consultas realizadas sempre que os dados

[14] É possível construir uma imagem Docker com instruções no link referido, mas a Oracle não permite oficialmente sem licença prévia, além de tomar tempo (mais que 15 min).

necessários estão disponíveis. Estes então são guardados. É específico para pré-calcular dados derivados e agrupá-los (*downsampling)*. O segundo se refere a políticas de retenção. Com isso é bastante simples programar o descarte dos dados após um período de tempo. Usando ambos em conjunto, é possível fazer o *downsampling*, seguido de um descarte de dados.

No caso do **PostgreSQL**, é possível simular esse comportamento com *triggers*, ou disparadores, em conjunto com *functions*, ou funções. Demanda apenas mais mão-de-obra de um desenvolvedor, por esta ferramenta ser mais genérica e complexa, além de não suportar janelas de tempo (*"*`GROUP BY time(<tempo>)`*"* – existente no InfluxDB*)*.

Sobre o Oracle e o OpenTSDB. No primeiro, usam-se recursos praticamente iguais aos do PostgreSQL. No segundo, um comando externo deverá fazer uma consulta, utilizar funções de agregação por janelas de tempo, guardar em uma nova métrica e então ordenar a remoção dos dados obsoletos.

3.2.3.3 Acesso aos dados por aplicações

As aplicações devem possuir acesso fácil, seja por bibliotecas, seja por *APIs* (interfaces programáveis) utilizando protocolos padrão, como o *HTTP*. No caso do PostgreSQL e do Oracle, existem diversas bibliotecas, em especial no primeiro caso, que é suportado por padrão em diversas bibliotecas de ORM, como a biblioteca *ActiveRecord,* usada pelo *webservice* do NetMetric.

Por outro lado, o InfluxDB e o OpenTSDB, possuem menos opções de bibliotecas (apenas uma de cada para a linguagem Ruby, por exemplo, que estejam atualizadas para as últimas versões destes bancos de dados[15]), mas compensam por oferecer acesso padronizado e bem documentado via *API* sob *HTTP*.

3.2.4 Contribuidores e comunidade ativos

Outro fator importante para escolha das ferramentas, fora o desempenho avaliado neste trabalho, é a questão do nível de atividade ao redor dessas ferramentas. Inclui-se tanto o interesse dos usuários e potenciais desenvolvedores, quanto o interesse dos potenciais contribuidores e mantenedores.

Avaliando o **PostgreSQL**, por (SOLID IT GMBH, 2016), é possível verificar um alto interesse dos usuários, com uma tendência de alta para a popularidade. Já o interesse dos contribuidores é alta, pois colaboradores de várias empresas de grande porte contribuem, como EnterpriseDB, Mozilla, CitusData, Heroku, VMWare, NTT e Google[16].

No caso do **InfluxDB**, utilizando também dados de (SOLID IT GMBH, 2016), a pontuação é certamente menor que do PostgreSQL e Oracle, devido a ser uma ferramenta de nicho. Mas há uma tendência forte de alta para sua popularidade, argumento que pode ser fortalecido pelas empresas usuárias citadas no *site* oficial, como eBay, Mozilla, NIST, etc. Além disso as estatísticas *Pulse* do GitHub[17] revelam que seus contribuidores estão bastante ativos.

Finalmente, há o caso do Oracle, que possui maior popularidade de todas soluções pelo ranking acima utilizado. Isso indica um forte uso da ferramenta.

---

[15] Para o InfluxDB: https://github.com/influxdata/influxdb-ruby. Para o OpenTSDB: https://github.com/opower/time-series

[16] Verificável em http://www.postgresql.org/community/contributors/

[17] O *Pulse* é um medidor de atividade do GitHub. Acesso em https://github.com/influxdata/influxdb/pulse/monthly

4 METODOLOGIA DE AVALIAÇÃO

Neste capítulo será apresentado a base do método de avaliação, onde dois trabalhos anteriores são mostrados e discutidos, visto que o segundo deles mostra como arquitetar um teste de *big data*, e o primeiro é um influente trabalho já realizado com uma carga genérica.

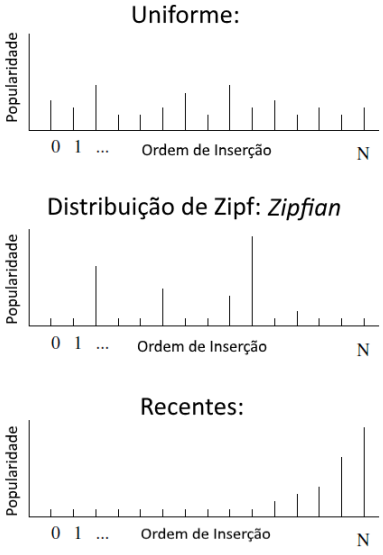
4.1 Introdução: Moldando o método

Para a moldar o método de avaliação utilizado nesse trabalho, foram estudados relatórios prévios relacionados a *benchmarking* voltado ao mundo *big data*. Chegou-se à conclusão de que dois trabalhos são significativos para o desenvolvimento deste.

O primeiro deles refere-se a (COOPER, SILBERSTEIN, *et al.*, 2010), relatando a experiência do *Yahoo! Research* em criar uma ferramenta padrão de análise de desempenho de sistemas na nuvem, que são fortemente baseados em diversos tipos de bancos de dados, especialmente os do tipo NoSQL, como o Hadoop e o Cassandra. Havia inclusive uma necessidade de comparação destes (e outros) com o banco relacional MySQL.

Mais importante, porém, é o relato sobre os tipos de *workloads*, ou cargas de trabalho, citando as decisões de projeto dessa carga de trabalho voltada as necessidades que eles identificaram, incluindo a criação da carga, os meios usuais de acesso, e as distribuições para consulta de dados. Esse relatório citado vai ter um impacto nas decisões que este trabalho aqui vai apresentar nas seções seguintes, como, por exemplo, as distribuições de probabilidade da Figura 4.1.

Figura 4.1 - Distribuições de probabilidade de escolha de um conjunto de dados



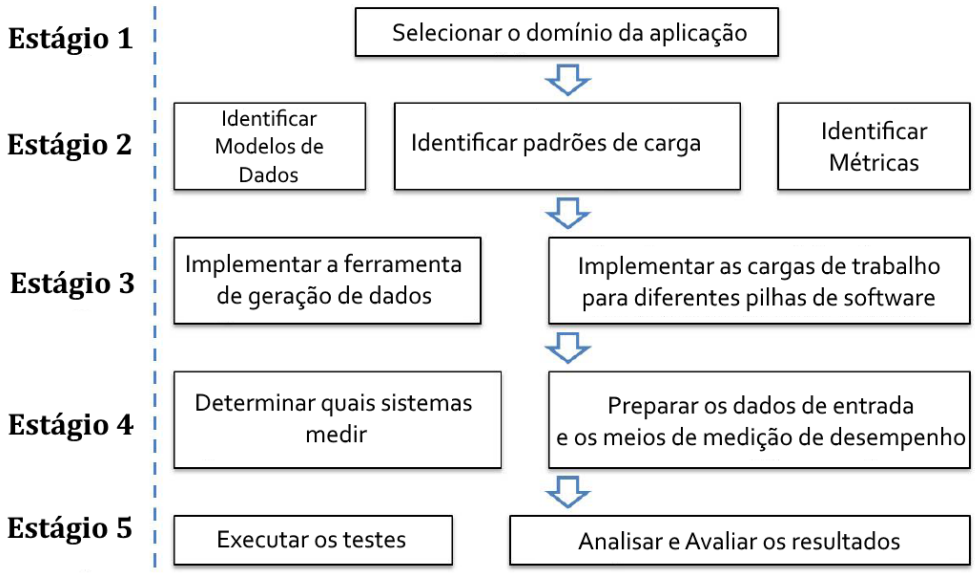
Fonte: Tradução de (COOPER, SILBERSTEIN, et al.*, 2010, p. 147)*

O segundo trabalho que teve grande impacto foi o de (HAN, JIA, *et al.*, 2015), que tomou outros diversos trabalhos anteriores e moldou um *framework*, ou esqueleto, bastante significativo. Outro fator que é relevante comentar é do reconhecimento de que aplicações diferentes – e específicas – exigem cargas de trabalho diferentes (seção 2.3.2).

Esse último fator explica a necessidade de implementação de uma carga de trabalho específica, que faz com que se apresente neste capítulo e no capítulo 5 um método baseado nas demandas do NetMetric, e de, provavelmente, outros tipos de medição de redes, que foram apresentadas nas seções 2.2.4 e 2.3.5.

Ainda em (HAN, JIA, *et al.*, 2015), foi apresentado a Figura 4.2, que relaciona cinco estágios sugeridos para a avaliação de sistemas *big data.*

Figura 4.2 - Avaliação de desempenho de big data em cinco estágios



Fonte: tradução de (HAN, JIA, et al.*, 2015, p. 3)*

No **estágio 1**, pede-se a seleção do domínio de aplicação, que, no caso desse trabalho, é o NetMetric (e possivelmente outras ferramentas de medição de redes), onde diversas sondas de teste, em diversas localidades da rede (tanto em termos geográficos, quanto em termos da logística da rede), enviam resultados em ordem de marcação de tempo próxima, na maior parte do tempo. Essa apropriação foi feita na seção 2.1, especialmente a subseção 2.2.4, onde é dado a ideia do NetMetric.

Para o **estágio 2**, deve-se identificar os modelos de dados e os padrões de carga, que no caso desse trabalho, são representados por, na maior parte, um número associado à uma métrica, ao longo de uma linha de tempo. Para a consulta é sempre desejável que seja mostrada ordenada, mesmo que a inserção não seja completamente ordenada (mas é boa parte do tempo, especialmente dentro de uma janela de tempo – um *buffer* temporal). Isso foi descrito nas subseções 2.2.4 e 2.3.2, com mais detalhes sendo fornecidos na seção 4.2.

As métricas identificadas por este autor, que nesse presente trabalho chamam-se simplesmente **métricas quantitativas**, são medições de desempenho dos bancos de dados (não confundir com métricas de rede, que é relacionado ao desempenho da rede que o NetMetric mede). Essas métricas de desempenho de banco de dados são apresentadas a seguir, ao longo de toda seção 4.3.

No **estágio 3** deve-se apresentar uma solução de geração dos dados de teste, além da implementação específica para cada pilha de software (em se tratando de *big data*, normalmente uma solução de aquisição e análise de dados é formado por diversos softwares que trabalham em cadeia, apesar de aqui ser apenas avaliado a parte do banco de dados puro). Essa parte será mostrada ao longo da seção 4.5, juntamente com o código que implementa esse gerador.

Já no **estágio 4**, pede que se mostre quais sistemas serão testados, que se prepare os dados de teste, além do método de medição, como e onde foi medido. Os sistemas que serão testados são o PostgreSQL e o InfluxDB, escolha justificada ao longo da seção 3.2. A preparação dos

dados e os métodos de medição (como foi medido) estará apresentado parcialmente na seção 4.3, completado pelas seções 4.4 e 4.5.

Por fim, o **estágio 5**, onde os resultados serão apresentados ao longo do capítulo 5. Ainda há o capítulo 6, que deve encerrar esse trabalho com conclusões baseadas desses resultados e com uma avaliação geral do trabalho.

4.2 Modelagem dos dados: conjuntos de teste

Uma primeira modelagem dos requisitos do NetMetric, a ideia geral – aquela que resultados de medições de rede são enviados de maneira quase caótica de diversos pontos da rede, em um volume massivo – já foi mostrada na seção 2.3.2, e aqui vamos continuar em um nível mais baixo, com alguns detalhes dos dados que o banco recebe do webservice do NetMetric e uma boa noção da magnitude do volume de dados, baseado complemente em exemplos de uso real.

Para fazer uma comparação fidedigna da realidade, foi construído um conjunto de dados inspirado em uma das mais utilizadas características por analistas ao consultar o *webservice* do NetMetric, que é dos testes realizados em servidores DNS pelo NetMetric.

O NetMetric, ao realizar os testes de DNS, atualmente busca apenas duas métricas: tempo de resposta e *status* (sucesso da consulta). No entanto cada sonda testa, a cada hora, um conjunto de 50 sites, os 50 mais acessados do Brasil, segundo o ranking do Alexa[18], contra uma média de 10 servidores DNS. Considerando um universo de 500 sondas, temos 250.000 resultados por hora, que podem ser representados por 250.000 registros de bancos de dados.

Nos testes de DNS são, primariamente, armazenados, além do tempo de resposta (inteiro) e sucesso da consulta (booleano), são armazenados marcadores (ou filtros) referentes aos seguintes campos:

- Nome da sonda (*alias)* [*string*:32 caracteres]
- IPv6 [*boolean:*1 byte]
- URL [*string:*256 caracteres]
- Servidor DNS *(server)* [*string:*64 caracteres]

Resumindo, temos 4 colunas de atributos, mais 1 representando o *timestamp*, com precisão de segundos, mais 2 representando os **valores de atraso e de sucesso** (um inteiro de 32 bits e uma cadeia de 8 caracteres, respectivamente), além do **nome da métrica** (simplificaremos para uma cadeira de 16 caracteres), que moldam o equivalente a 8 colunas em uma planilha de dados. Ainda leve em consideração que temos 250.000 registros por hora.

Considerando que cada caractere ocupa 2 bytes na codificação UTF-8 e o *timestamp* tem 4 bytes em formato UNIX, temos uma soma que pode ser feita da seguinte maneira: `[4+64+1+512+128+4+16+32] = 761` **bytes por registro** (quando completamente cheios), com a média aproximada de **220 bytes por registro** (calculada após algumas iterações do código de geração). Isso em dados brutos, sem cabeçalhos, ou formatação para as linguagens dos bancos de dados testados.

As consultas típicas a esse conjunto de dados são de registros dos últimos 30 minutos, ou da última hora. Apesar de nãos ser usual, também vamos forçar a consulta para o último dia. A consulta típica de uma hora se faz normalmente filtrando um servidor DNS específico, ou seja, para esta hora, são buscados 5.000 registros desses 250.000.

[18] Disponível em http://www.alexa.com/topsites/countries/BR.

4.3 Métricas quantitativas

Para avaliar os candidatos escolhidos, foram definidas algumas métricas quantitativas, que podem ser medidas em números, através de testes bem definidos. Estes testes serão divididos e explicados em cada uma das subseções seguintes e basicamente se dividem em vazão, degradação e ocupação.

4.3.1 Vazão na consulta de dados

O primeiro teste que será mostrado será o de vazão de dados na consulta. Este é um dos mais importantes testes, pois está diretamente relacionado a experiência de uso que o analista faz. Essa métrica define o quão rápido os dados requisitados são oferecidos ou para a aplicação geradora de gráficos e relatórios, ou diretamente ao analista.

Neste teste, utilizando os conjuntos de dados obtidos conforme seção 4.5 (Dados de Teste e Medição), serão submetidos a consultas de 3 diferentes tamanhos fixos, mas com naturezas e propósitos diferentes, para cada um desses conjuntos, detalhados na Tabela 4.1.

Tabela 4.1 - Tipos de consultas realizadas para vazão de dados de consulta

Tamanho do subconj.	Distribuição	Uso típico
2.500 em 125.000	Recentes	DNS – 30min
5.000 em 250.000	Recentes (x2)	DNS – 1h
2.500 em 125.000	Zipfian	DNS – 30min
5.000 em 250.000	Zipfian (x2)	DNS – 1h

Fonte: (DLUGOKENSKI, 2016)

Explicando o método de teste, conforme vemos na Tabela 4.1, o tamanho da janela de tempo utilizado na primeira bateria de testes será um que selecione aproximadamente 125.000 registros (do *timestamp* inicial ao *timestamp* final, que representem meia hora de dados). O segundo critério será utilizar um filtro que selecione 2.500 registros desse subconjunto.

A distribuição "**recentes"** se refere que a consulta será feita nos dados mais recentes (da última hora, por exemplo). Já a distribuição "***zipfian*"** utiliza uma distribuição específica, que busca valorizar as chamadas zonas quentes (por exemplo, um registro popular) em todo universo dos dados, do mais antigo ao mais recente.

Sobre a primeira, distribuição recentes, 100% dos dados da última janela de tempo são consultados, o que simula, por exemplo, uma aplicação de acompanhamento em tempo real, onde o analista fica acompanhando o sistema o dia todo. Ainda há uma chance de consulta da janela anterior, de 50% para cada uma das consultas anteriores, simulando um sistema em atraso ou o desejo do analista de ver dados anteriores. Por fim, uma chance de 33% de consultar cada antepenúltima janela dos dados mais atuais.

Já a distribuição "*zipfian"*, baseado na lei de Zipf, que foi explicado e utilizado em (COOPER, SILBERSTEIN, *et al.*, 2010, p. 147,149), simula uma certa aleatoriedade no acesso a dados antigos. Nessa distribuição um termo mais popular tem o dobro de acessos do menos popular, e assim por diante. Ele também de certa maneira cria clusters, onde vizinhos do dado mais popular também são populares.

Como no algoritmo *zipfian* original ele torna o 1º termo inserido o mais popular, e assim por diante, que não é o comportamento real do banco de dados, tem que se haver uma modificação. Neste trabalho foi feito exatamente a mesma modificação proposta por (COOPER, SILBERSTEIN, *et al.*, 2010), apenas adaptando código Java[19] em Ruby.

Assim, digamos, dados que estejam no meio desse conjunto ordenado por tempo, se tornam populares – além de seus vizinhos próximos. Da mesma maneira, um dado encontrado

[19] Disponível em https://github.com/brianfrankcooper/YCSB.

em outro ponto qualquer do banco se torna popular, não tanto quanto o primeiro, mas muito maior que os demais, com somente os vizinhos próximos tendo similar popularidade.

Para fins de verificação, se os dados forem reordenados por ordem de popularidade (acessos), ao invés da ordem temporal, é possível verificar a distribuição clássica *zipfian.*

Por fim, para simular o uso típico, as áreas selecionadas serão contíguas, sem saltos em relação ao *timestamp*, um subconjunto do tipo [*timestamp* inicial; *timestamp* final], chamada de janela de tempo da consulta. No entanto haverá uma escolha aleatória do filtro de servidor DNS, simulando o interesse do usuário pelo estado desses diversos servidores, que não é exatamente a mesma ordem de inserção no banco, mas que deve estar próxima, se o banco de dados ordenar por *timestamp*.

O resultado de cada um desses testes será o número de registros retornados por segundo de teste. Como a quantidade de dados testados é grande, ainda haverá uma rolagem (simplificação) dos resultados, minuto a minuto, pois segundo a segundo haveriam muitos pontos, utilizando a média.

4.3.2 Vazão na ingestão de dados

Na vazão ao ingerir dados, temos como objetivo avaliar o quão rápido é possível inserir registros. Os dados pré-definidos na linguagem esperada do banco de dados serão enviadas ao mesmo.

Esses dados, como característica do NetMetric, estão praticamente em ordem, com variações dessa ordem apenas dentro de uma janela de 30 minutos, onde estes dados podem chegar a qualquer momento em relação ao relógio atual.

Portanto, será aplicado, em cada janela de tempo, desordenação proposital de vários dos dados enviados, mas sempre respeitada a janela de tempo, portanto de maneira quase ordenada (janela de tempo sempre estão ordenadas, mas não os dados dentro dessa janela).

O resultado de cada um desses testes será o número de registros confirmados como entregues (pelo sistema de gerenciamento de BD) por segundo de teste, e, da mesma maneira que na consulta, serão rolados estes resultados de 5 em 5 minutos, obtendo a média, pois a quantidade de consultas é muito maior, levando bem mais tempo.

4.3.3 Armazenamento utilizado

Na métrica de armazenamento utilizado, será verificado se, após a inserção do conjunto de dados pré-definidos – exatas 20 milhões de linhas (ou pontos) – aproximadamente 4 GB em dados brutos – respectivamente, que será verificado em cada um dos conjuntos de dados de teste, em cada sistema.

O resultado será o espaço ocupado em disco para um dos sistemas de bancos de dados, buscando mostrar ao avaliador tanto o tamanho ocupado em relação ao número de dados brutos, quanto uma expectativa de crescimento do banco em relação ao número de registros inseridos ao longo do tempo em atividade.

4.4 Ambiente de validação

Nesta seção será apresentado todo ambiente de teste utilizado, desde a arquitetura do código à estrutura de máquinas, de maneira que se comprove a plausibilidade da avaliação.

4.4.1 Arquitetura do código produzido e das máquinas utilizadas

Conforme será explicado na seção 4.5, os dados serão gerados conforme modelo utilizado no NetMetric.

Pertinente à metodologia de avaliação, o desempenho desse código de geração, escrito em Ruby, não terá influência nos testes, pois os dados serão criados já com os comandos nativos

dos bancos testados, e pré-armazenados na máquina que simulará o cliente, enviando as requisições ao servidor dos bancos testados.

Será apresentado junto aos resultados, um teste de controle, utilizando o Netcat (que realiza conexões *socket* simples), na qual apenas joga os dados no disco, sem passar por um sistema gerenciador de banco de dados, com o objetivo de mostrar que o *script* em Ruby, a capacidade de processamento do cliente, do enlace de rede entre cliente e servidor, e da vazão de ambos os discos de armazenamento **não são** pontos de **gargalo** no teste.

4.4.2 Máquinas Docker

Conforme visto na seção 2.5, da página 31, o teste utilizando máquinas Docker não deve atrapalhar o nosso método de avaliação, em relação ao desempenho, além de facilitar a execução dos testes. Além do mais os dois candidatos recebem iguais condições nesse quesito, portanto com mesmo *overhead* (custo extra, em português).

4.4.3 Máquinas hospedeiras

A máquina hospedeira do Docker, representando o servidor de banco de dados, terá somente os serviços mínimos essenciais para rodar esses contêineres. Na máquina hospedeira será instalada a distribuição Ubuntu Server, e nesta estão instalados somente os serviços mínimos necessários para rodar o Docker, sem serviços extras de controle sofisticados, além do SSH e do TMUX, que permitem iniciar os testes e acompanhar o andamento dos mesmos.

Ainda há segunda máquina, representando o cliente, dispensa o Docker, necessitando somente do interpretador Ruby e de ferramentas essenciais como SSH, TMUX, BASH, além de acesso à rede e espaço em disco suficiente.

Lembrando ainda que estas máquinas são virtualizadas do tipo KVM, pois serão máquinas contratadas na nuvem, devido a indisponibilidade de máquina real capacitada para o autor deste trabalho.

Pelo contrato, para elas terão disponíveis 2 núcleos de uma máquina Xeon de penúltima geração da Intel, 7,5 Gigabytes de memória RAM acessíveis, 32 Gigabytes de disco do tipo sólido (SSD) e uma conexão de até 10 Gigabit por segundo entre cliente e servidor. Para mais detalhes, basta consultar o site da **Amazon EC2**, especificamente a máquina do tipo "***M3.large***".[20]

4.5 Dados de Teste e Medição

Agora, nesta seção, serão apresentados detalhes de formato exato dos dados a serem enviados, como eles foram obtidos e como será controlada a medição de desempenho.

4.5.1 Resumo do Modelo de Dados

Já foi verificado nas seções 2.3.2 e 4.2 que a modelagem de dados de teste será baseada nos moldes utilizados pelo NetMetric. O que falta completar, que será mostrado nesta seção, é de que maneira serão gerados os dados, e com quais características.

Também já foi visto que o modelo de registro basicamente segue o formato representado na Tabela 4.2.

[20] Documentação disponível em: https://aws.amazon.com/pt/ec2/instance-types/

Tabela 4.2 - Formato básico de um registro do NetMetric (Tupla de evento)

Timestamp	Métrica	Marcadores	Valores
Exemplo:			
1453123800	DNS	Alias=RS-POA-01; IPv6=false; etc;	Time=52; Status=OK;

Fonte: (DLUGOKENSKI, 2016)

Na Tabela 4.3, resumem-se alguns detalhes de cada campo e a cardinalidade dos dados no conjunto de testes.

Tabela 4.3 - Itens presentes na tupla de evento e as cardinalidades no conj. de testes

Campo	Subcampo	Cardinalidade	Exemplos
Métrica	-	1	DNS;
Marcadores	*Alias* (sonda)	100	RS-POA-xyz-001;
	IPv6	2	True; False;
	URL	50	google.com; g1.com.br;
	ServerDNS	10	8.8.8.8; 2001:4860:4860::8888;
Valores	*status*	8	ok; nxdomain;
	time	0 < x < 20000, inteiro	500 (ms - RTT)

Fonte: (DLUGOKENSKI, 2016)

Assim se resumem os dados presentes no conjunto de dados de teste. Com essa identificação precisa, é possível verificar que os registros do NetMetric, apesar de não serem exatamente uma tabela bidimensional, como o caso dos marcadores e valores – que apresentam subcampos – se comportam de maneira estruturada e podem ser adaptados para ambos motores de banco de dados testados.

4.5.2 Obtenção e envio dos dados para ingestão e consulta

Basicamente, o método de avaliação utiliza dados gerados previamente por um *script*, chamado aqui nesse trabalho de fase de geração ou **"*generate*"**.

Estes dados gerados foram moldados a partir das necessidades identificadas na aplicação específica (NetMetric), onde, de maneira controlada, podem se controlar as variáveis que influenciam o cálculo, mas que, ao mesmo tempo, gerem dados em volume, diversidade, velocidade e veracidade necessários, ideia que é apresentada tanto por (HAN, JIA, *et al.*, 2015, p. 1,2), quanto por (TAY, 2011, p. 1470).

> *Given a dataset D and a scale factor s, generate a synthetic dataset ~D that is similar to D but s times its size. (TAY, 2011, p. 1470)*

Para resolver o desafio do volume, diversidade e veracidade, será utilizada a modelagem apresentada nas seções anteriores (2.3.2, 4.2 e a recém apresentada seção 4.5.1), onde mostrou-se detalhes dos dados de medição do NetMetric.

Escolheu-se também as cardinalidades dos dados, que devem ser realistas no contexto dessa aplicação – e o são – mas que foram controladas, para não apresentar surpresas que invalidem os cálculos, como a mudança dinâmica do número de sondas, que alteram densidade de registros por janela de tempo nos dados do sistema real em produção.

Ainda em (TAY, 2011, p. 1471), apresenta-se o problema de se manter a correlação dos valores no conjunto de testes, especialmente em redes sociais. No caso específico do NetMetric, as amarrações são mais simples e podem ser descritas assim:

- *Ao longo do tempo, a cada janela de tempo de 30 minutos, as 100 sondas enviam um conjunto de 50 resultados de teste (medições de tempo e estado de 50 sites).*

Com essas amarrações, os dados podem ser gerados de maneira indefinida ao longo do tempo, conforme se deseja escalar o sistema, com "*s < 1"* – escolhido nesse trabalho, ou possivelmente *"s = 1"* ou ainda *"s > 1"*.

A variável que pode mudar em futuros trabalhos, ou novos testes, é basicamente o número de sondas, com as outras relativamente estáveis, assim o número de sondas foi escolhido para representar uma necessidade presente de validação, e que pode ser mudado no futuro, as outras foram simplesmente copiadas do sistema atual.

Outro desafio aparente é da velocidade necessária para que a simulação seja válida. Escolheu-se que dados serão pré-gerados na linguagem específica dos bancos, seja utilizando SQL, InfluxQL ou *Influx Line Protocol*.

Assim eles podem ter tamanhos diferentes do esperado em relação ao tamanho dos dados brutos, já que contém instruções e formatos diferentes. Mas os dados brutos também serão gerados para confirmar os cálculos apresentados na seção 4.2.

Estes dados (brutos e formatados) ficarão armazenados na máquina cliente, após essa geração dos dados, existe um *script* que simplesmente envia esses dados para o InfluxDB ou para o PostgreSQL, utilizando as bibliotecas de comunicação pertinentes para cada caso. Um resumo do algoritmo está na Figura 4.3.

Portanto, utilizando esse método, onde combinam-se dados formatados pré-gerados e um script de envio simples, espera-se que não haja gargalos na hora de transmitir os dados, conseguindo saturar a máquina com o banco de dados e obtendo resultados plausíveis. O já comentado teste do Netcat vai garantir que não existe essa contaminação, que foi detalhado na seção 4.4.1.

Figura 4.3 – Algoritmo resumido da fase "generate"

```
#Rotina de impressão: (parâmetro: linha [lista de colunas])
Para cada banco de dados:
    Imprima, na linguagem do Banco de dados, no arquivo ref. ao banco, a linha
recebida

#Fase de Inserção
Número de linhas = ?
Início da janela de tempo = Tempo arbitrário
Fim da Janela de tempo = Inicio + 30 minutos – 1 segundo
Enquanto o número de linhas não for alcançado:
  Para cada sonda (alias) da lista de sondas:
    Timestamp = segundo aleatório entre o início e fim da janela
    Para cada ServerDNS da lista de servidores DNS:
      Para cada url da lista de urls:
         Crie uma lista de colunas (linha) contendo:
            Timestamp, sonda, serverDNS, url, rand(ipv6), rand(status), rand(RTT)
         Rotina de impressão (linha acima)
    Início da Janela = Inicio + 30 minutos
    Fim da Janela = Fim + 30 minutos

#Fase de seleção – recentes
Número de clientes = 100
De 0 a número de clientes faça:
  Para cada serverDNS da lista de servidores DNS:
    Crie uma linha de consulta que filtre os resultados por:
        serverDNS AND Timestamp entre o início e fim da ÚLTIMA janela da fase
anterior
    Rotina de impressão (linha acima)

    Com 0.5 de chance crie outra linha com:
        serverDNS AND Timestamp entre o início e fim da PENULTIMA janela da fase
anterior
    Rotina de impressão (linha acima)

    Com 0.333 de chance crie outra linha com:
        serverDNS AND Timestamp entre o início e o fim da ANTEPENULTIMA janela da
fase anterior
    Rotina de impressão (linha acima)

#Fase de seleção – recentes dobrada
O mesmo algoritmo acima, mas o tamanho da janela de seleção de dados é na verdade
de duas janelas (a citada em caixa alta e a imediatamente anterior).

#Fase de seleção – zipfian
Número de clientes = 100
Lista de janelas = ScrambledZipfian(número total de janelas da fase de inserção)
De 0 a número de clientes faça:
  JANELA DE TEMPO = Próximo da lista de janelas
  Para cada serverDNS da lista de servidores DNS faça:
    Crie uma linha de consulta que filtre os resultados por:
        serverDNS AND Timestamp entre o início e fim da JANELA DE TEMPO
    Rotina de impressão (linha acima)

#Fase de seleção – zipfian dobrado
Mesma situação, toma-se o algoritmo da fase zipfian e dobra-se a janela
selecionada, tomando a JANELA DE TEMPO escolhida mais a imediatamente anterior
```

Fonte: (DLUGOKENSKI, 2016)

4.5.3 Método de medição das consultas e inserções

O método de medição de vazão das consultas foi projetado para ser o menos intrusivo e mais simples possível para análise posterior. A ideia é que o gancho de medição que é feito no algoritmo de teste não aplicasse muita interferência no próprio teste. Assim dividimos a medição em duas fases: a de obtenção de dados brutos de medição, e a de síntese desses dados brutos em dados mais resumidos para análise.

4.5.3.1 Fase Runner: Obtenção de dados de medição

Na primeira fase, de obtenção de dados brutos de medição – também chamado neste trabalho de *runner*, existe um gancho, feita do lado do cliente (que dispara o teste), o qual basicamente imprime na saída padrão um contador de linhas processadas e o marcador de tempo *(timestamp)* corrente dessa máquina, a cada segundo.

Nessa fase *runner*, rodamos, para cada especificação de banco de dados escolhida (que é um produto cartesiano entre motor de banco e entre possíveis configurações do mesmo banco), em sequência, uma subfase de inserção de dados, seguido da subfase de seleção de dados recentes, e por fim, a subfase de seleção de dados *zipfian*, conforme Figura 4.4.

Figura 4.4 - Algoritmo resumido da fase "runner"

```
#RUNNER
#Thread principal
Abra o arquivo de dados
Conecte ao banco de dados
Inicie Thread de Impressão de Contador
Enquanto não for o final do arquivo:
  Leia uma linha
  Envie essa linha para o banco de dados
  Contador global incrementa 1
Encerre Thread de Impressão de Contador

#Thread de Impressão de Contador
Faça indefinidamente:
  Durma por 10 ms
  Se o relógio atual – relógio anterior >= 1 segundo:
    Relógio anterior = Relógio atual
    Imprima na saída padrão:
       Relógio Atual (Formato UNIX); Contador global;
```

Fonte: (DLUGOKENSKI, 2016)

Nessa mesma Figura 4.4 perceba que tem uma implementação bem enxuta, que, como já foi comentado, visa causar o menor impacto possível na medição. No código ele é rodado uma vez para cada fase do teste (inserção, seleção de recentes, seleção de recentes dobrada, seleção *zipfian*, seleção *zipfian* dobrada), para cada configuração de banco de dados escolhida, com algumas diferenças inerentes ao método de conexão e envio de dados à esses sistemas de banco de dados.

Percebe-se também que a quantidade de medições criadas é bastante alta, o que exige uma sintetização desses dados para melhor compreensão, o que é feito na fase de *rollup*.

4.5.3.2 Fase Rollup: Síntese dos dados de medição

A segunda fase, que é a de síntese dos dados brutos criados na fase anterior, chamado aqui de *rollup*, agrupamos dados em janelas arbitrárias maiores do que 1 segundo (30 segundos,

por exemplo), mantendo a unidade de vazão "**linhas por segundo"**, simplesmente fazendo a média que utiliza cada valor dessa janela, pelo número de pontos da mesma.

O objetivo é de tornar os gráficos resultantes mais legíveis, e possivelmente amortecer pequenas flutuações de processamento. Na Figura 4.5 temos um resumo deste algoritmo.

Figura 4.5 - Algoritmo resumido da fase "rollup"

```
#ROLLUP
Número de segundos = PARAMETRO_SEGUNDOS
Abra o arquivo de dados de medição
Contador anterior = 0
Enquanto não terminar o arquivo de dados:
  Crie uma lista auxiliar vazia
  De 1 ao número de segundos faça:
    Pegue uma linha do arquivo e faça:
      Se for a primeira vez nesse grupo:
        Timestamp do grupo = timestamp da linha
      Contador atual = contador global
      Lista auxiliar recebe [contador atual – contador anterior]
      Contador Anterior = Contador Atual

  Caso a lista auxiliar não esteja vazia:
    Imprima na saída padrão:
      Timestamp do grupo; Soma dos Contadores; Número de pontos agrupados;
Média (Soma dos contadores / Número de Pontos; Menor Valor do Grupo, Maior
Valor do Grupo; Mediana do Grupo;
```

Fonte: (DLUGOKENSKI, 2016)

4.5.3.3 Medição do armazenamento utilizado

Por fim, na métrica de armazenamento utilizado, diferentemente da vazão, é utilizado um método ainda mais simples: após o fim do teste de inserção, basta rodar o comando unix "`du -sh`" apontado para a pasta de armazenamento dos arquivos daquele sistema de banco de dados.

Assim podemos dar por encerrado as devidas considerações acerca do método de medição e começar a analisar e discutir os resultados nas seções de resultados e de conclusões.

5 TESTES: EXECUÇÃO E RESULTADOS

Nesta fase, com os candidatos escolhidos e o método de avaliação bem definido, inicia-se a apropriação dos testes, mostrando, analisando e discutindo o andamento destes.

No entanto, antes de mostrar todos os resultados, houveram questões que que tiveram que ser arbitradas em relação tanto em relação a quantidade de variáveis de teste quanto da grande quantidade de dados coletados nos testes.

5.1 Variáveis de teste

A questão que apareceu em relação a quantidade de variáveis de testes, que veremos a seguir, é em relação a possibilidade de configuração – ou *tunning* – desses sistemas de banco de dados.

Em relação às configurações dos sistemas, há uma vasta gama de possibilidades de configuração para cada um deles, que para o **InfluxDB,** oferece uma gama de opções em relação a tamanho de diversos sistemas de cache, de arquivo WAL, entre outros.

Para o **PostgreSQL** a situação é ainda mais dramática, pois há um número muito grande de configurações, fora o exponencial número de combinações que pode ser feito. Assim neste trabalho a ideia é **manter as configurações padrão** desses dois sistemas.

Isso significa não utilizar *stored procedures*, ou "procedimento armazenado", no caso do PostgreSQL; ou escritas com protocolo UDP, no caso do InfluxDB, que apesar de promissora (na casa de dezenas de milhares de linhas por segundo), exige uma calibração muito fina para que não se perca linhas enviadas entre *buffers* UDP momentaneamente estourados, e não melhora o resultado na seleção.

No entanto, em meio a tudo isso, há uma ressalva, que será justificada na subseção abaixo.

5.1.1 PostgreSQL: Índices *BRIN*

Os índices *BRIN* ou ainda *Block Range Index* (Índices de blocos sequenciais, em livre tradução) foram anunciados para a versão 9.5 do PostgreSQL, durante a confecção deste trabalho. Por notória característica ser exatamente direcionada a atender demandas do objetivo principal deste trabalho, e por não ter um custo adicional grande em relação a possíveis modificações dos algoritmos e códigos já escritos, essa configuração adicional será avaliada nesse trabalho.

Serão utilizadas, para fins de teste, duas configurações do BRIN que parecem ser mais adequadas ao nosso caso, que são de 64 blocos por sequência e 128 blocos por sequência – considerando a configuração padrão do PostgreSQL de 8kb por bloco, representam em torno 600 a 800 linhas por *block range* no primeiro caso e de 1200 a 1600 linhas no segundo caso.

Ambas serão comparadas com o InfluxDB e a já tradicional solução com índice B-TREE, e, como ainda existe vaga documentação em que casos se aplica, logo mais importante será o *benchmarking* realizado nestas configurações.

5.2 Representação dos dados coletados

Dada a grande quantidade de dados obtidas, logo nos primeiros resultados obtidos em ensaios, se observou que apesar de variações consideráveis nos dados brutos em determinados pontos, havia um certo padrão. Também essa quantidade de dados, que são pontos segundo a segundo, ao longo de muitas horas em cada teste, levou necessidade de procurar uma maneira de derivar e concluir com mais facilidade.

Para isso foi desenvolvida nesse trabalho uma técnica de *rollup*, ou agregação de dados, explicado na seção 4.5.3.2, para que os dados mais relevantes ficassem visíveis. Com essa agregação também se percebeu que não havia necessidade de muitos gráficos para a tomada de

conclusões, já que o comportamento nos testes e em diferentes agregações testadas é repetitivo e conclusivo.

Utilizando a técnica de agregação supracitada, com janelas de agregação entre 1 segundo, 5, 10, 15, 30, 60, 120 e 300 segundos, se percebeu que para mostrar de forma compreensiva bastava que se utilizasse a janela de 30 segundos, com corte após 900 segundos – exagerado o suficiente para se convencer que há sim um padrão.

Assim, posto essas observações de como serão mostrados esses resultados – e porque eles são representados assim – podemos verificar esses resultados agregados nas próximas seções.

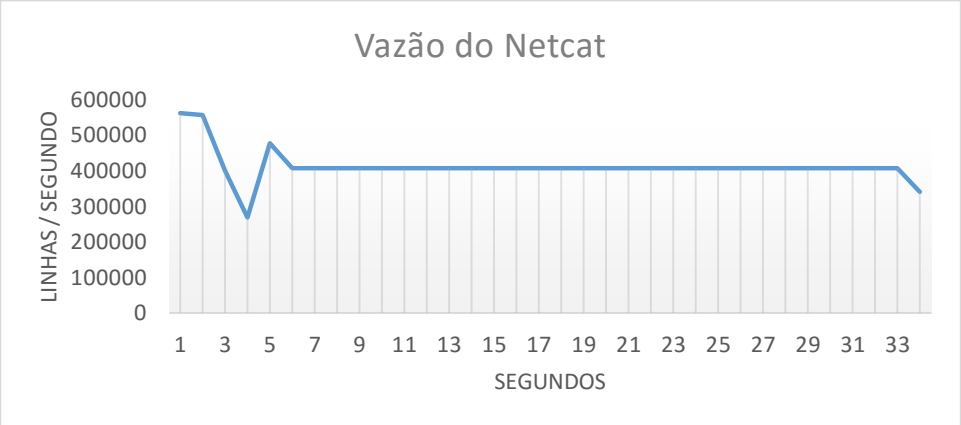
5.3 Teste de controle

O primeiro resultado que será discutido é do teste de controle. Como comentado na seção 4.4.1, optou-se por um teste bastante simples, mas que é significativo e testa tanto o algoritmo quanto teste o ambiente, no intuito de validar se, em conjunto, todos os elementos base (máquinas, infra de rede, componente e infra de armazenamento) dão vazão suficiente para que eles não sejam o gargalo.

A técnica é a seguinte: utilizando um soquete de rede TCP/IP (como ocorre na transmissão dos dados NetMetric ao sistema gerenciador de banco de dados), foram transmitidos os dados brutos, linha a linha, para o mesmo servidor de destino que depois foram instalados os *softwares* dos SGBDs.

No lado do servidor (que recebe os dados), fora utilizada a ferramenta **Netcat**, e esta foi redirecionada para um arquivo no disco local. Também foram utilizados determinados comandos de S.O. para desabilitar o cache de escrita desse sistema, que poderia esconder um possível gargalo no componente de armazenamento.

Figura 5.1 – Vazão do teste de controle



Fonte: (DLUGOKENSKI, 2016)

No lado do cliente, fora aberto um soquete simples, com comandos nativos do Ruby, no cerne (*loop* principal) do algoritmo de transmissão da fase "*runner*".

A conclusão desse teste, com resultados visíveis na Figura 5.1, mostram uma vazão na casa das 400.000 linhas por segundo, que fez o teste inteiro se encerrar em pouco mais de 30 segundos.

Nas próximas seções verificar-se-á que nenhuma das configurações de banco de dados testadas chegam próximo desse valor, levando ao entendimento de que o procedimento e ambiente são suficientes para o teste, da maneira proposta por este trabalho.

5.4 Vazão na consulta de dados

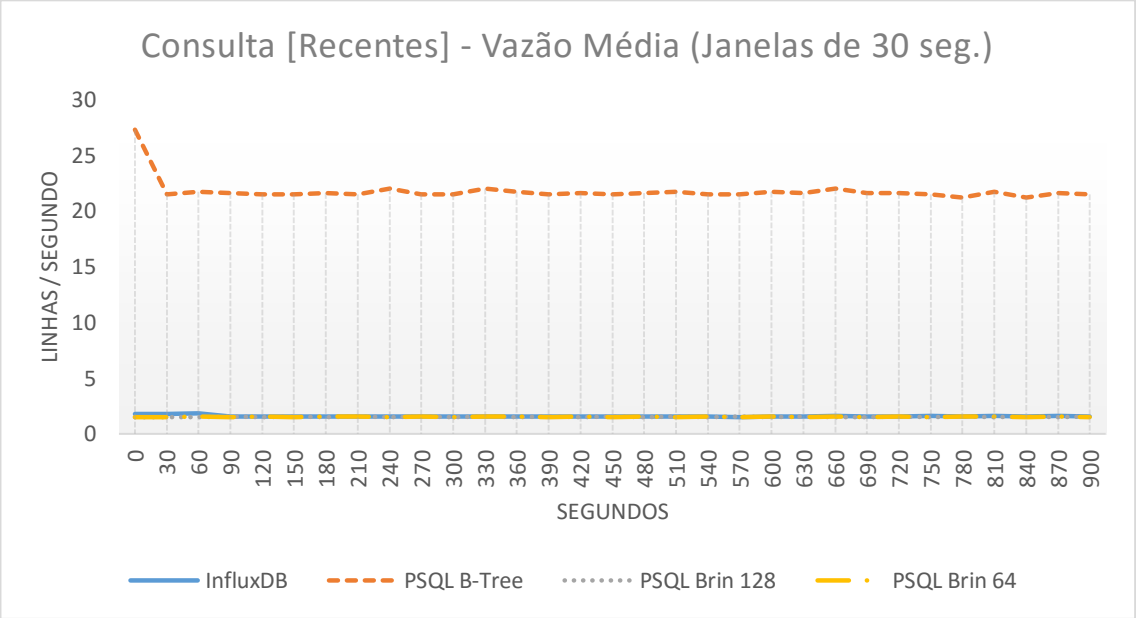
5.4.1 Dados recentes

Conforme discutido na seção 4.3.1, o teste de consulta (ou seleção) de dados recentes é muito importante para a responsividade da interface de usuário. O primeiro a ser apresentado é a vazão na consulta de dados recentes, que está na Figura 5.2.

Estes dados são aqueles que sempre são consultados por um operador humano visualizando um sistema de acompanhamento em tempo "real", algo previsto e implementado na interface de usuário do NetMetric.

Ainda nesse teste, apesar de não estar claro na figura 5.4, a média do PostgreSQL com BRIN é pouco mais baixa que no InfluxDB. No primeiro caso é de aproximadamente **1,55** linhas por segundo, tanto com tamanho de 128 blocos, quanto tamanho de 64 blocos. No segundo é de aproximadamente **1,62** linhas por segundo. No PSQL com B-TREE a vazão é de aproximadamente **21,8** linhas por segundo.

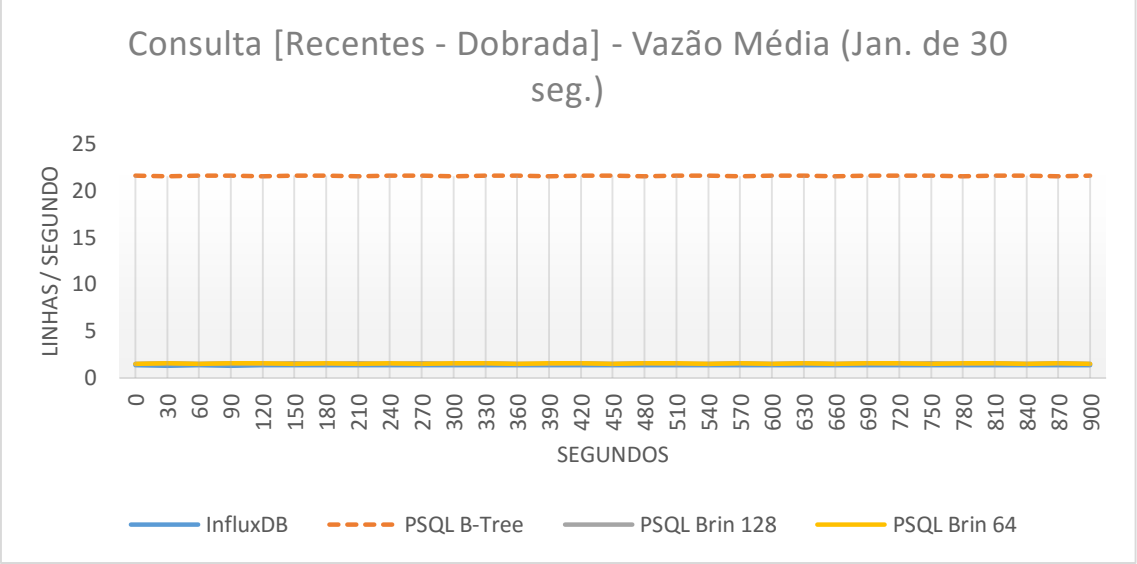
Figura 5.2 – Vazão média na seleção de dados recentes



Fonte: (DLUGOKENSKI, 2016)

Outra variação desse teste foi aplicada para se obter uma possível degradação no caso de se selecionar uma janela dobrada (a mais recente, e a segunda janela mais recente na mesma consulta). Na Figura 5.3 percebe-se que os resultados são similares, mas não idênticos.

Figura 5.3 – Vazão média na seleção de dados recentes, dobrada



Fonte: (DLUGOKENSKI, 2016)

Nessa variação a média geral foi de aproximadamente **1,54 l/s** para o PSQL com BRIN (sem alteração significativa), **1,35 l/s** para o InfluxDB, **21,4 l/s** para o PSQL com B-TREE. Informalmente foram feitos ensaios com janelas maiores que apontaram uma degradação ainda maior desses números, mas que nos dois últimos já estão suficientemente presentes para se afirmar que há essa degradação.

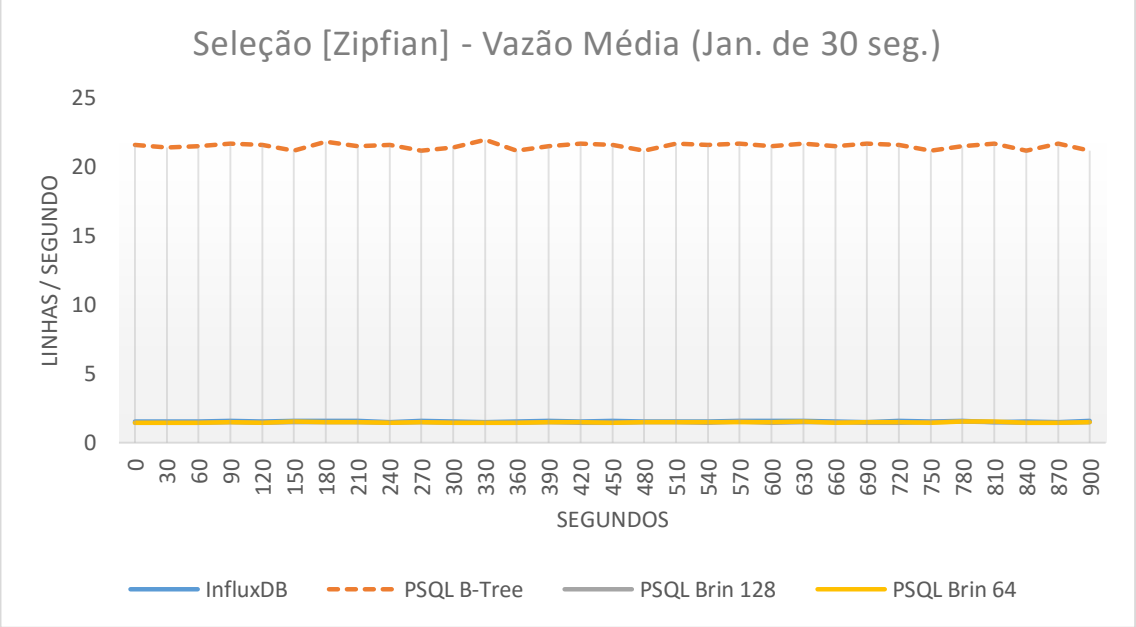
5.4.2 Dados distribuídos com a distribuição de Zipf

No teste de vazão de consultas, utilizando a distribuição de Zipf modificada, tal como foi explicado na seção 4.3.1, que é aquela que distribui as consultas em toda linha do tempo armazenada, mostrou alguns resultados interessantes.

Olhando a Figura 5.4, os números ficaram menores que na consulta de dados recentes, mas não muito diferentes. Se tem ali médias de **1,5 l/s,** para o PSQL BRIN, **1,56 l/s** para o InfluxDB (menor que em dados recentes, mas maior que em dados recentes com janela dobrada) e **21,2** para o PSQL com B-TREE.

No entanto, os resultados interessantes estão ao dobrar a janela de dados, onde ocorre uma redução drástica da vazão no PSQL com B-TREE (que ainda não retira dele o primeiro lugar entre as soluções comparadas). Como pode-se comprovar na figura, a média do PSQL com B-TREE baixa para **10,9 l/s**, enquanto das outras configurações fica menor, mas estável: **1,43 l/s** no PSQL com BRIN e **1,37 l/s** no InfluxDB.

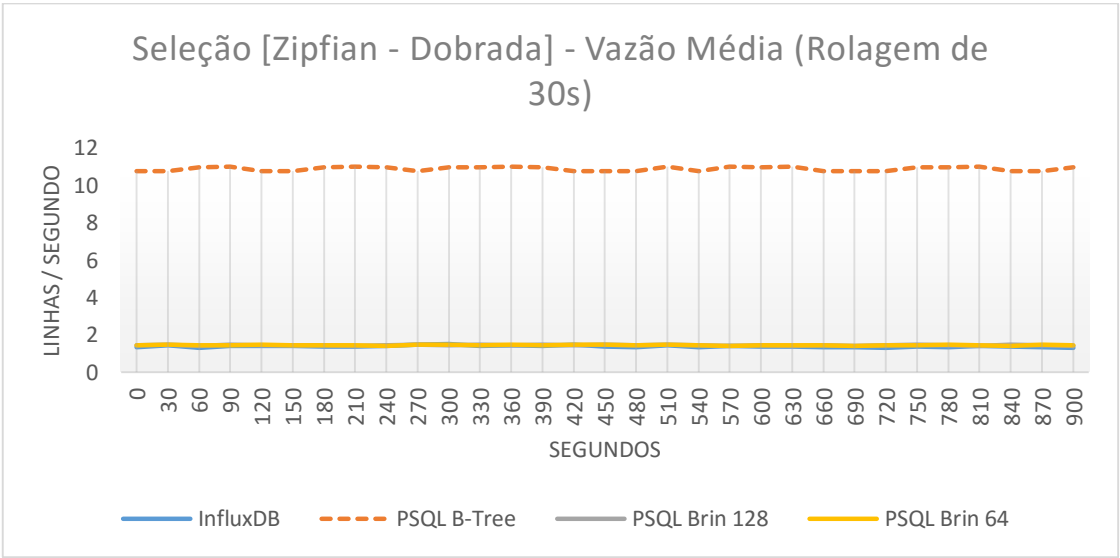
Figura 5.4 – Vazão média na seleção de dados com distribuição de Zipf



Fonte: (DLUGOKENSKI, 2016)

É possível que nesse teste tenhamos tocado diretamente em alguma fraqueza (seja ela causada por uma falha do sistema, seja ela causada por um débito técnico de implementação), pois conforme foi aumentado mais a janela, de maneira informal, esse comportamento não se repetiu, mantendo essa média entre 9 l/s e 10 l/s (aumentando a janela de consulta em 10x).

Figura 5.5 – Vazão média na seleção de dados com distribuição de Zipf, janela dobrada

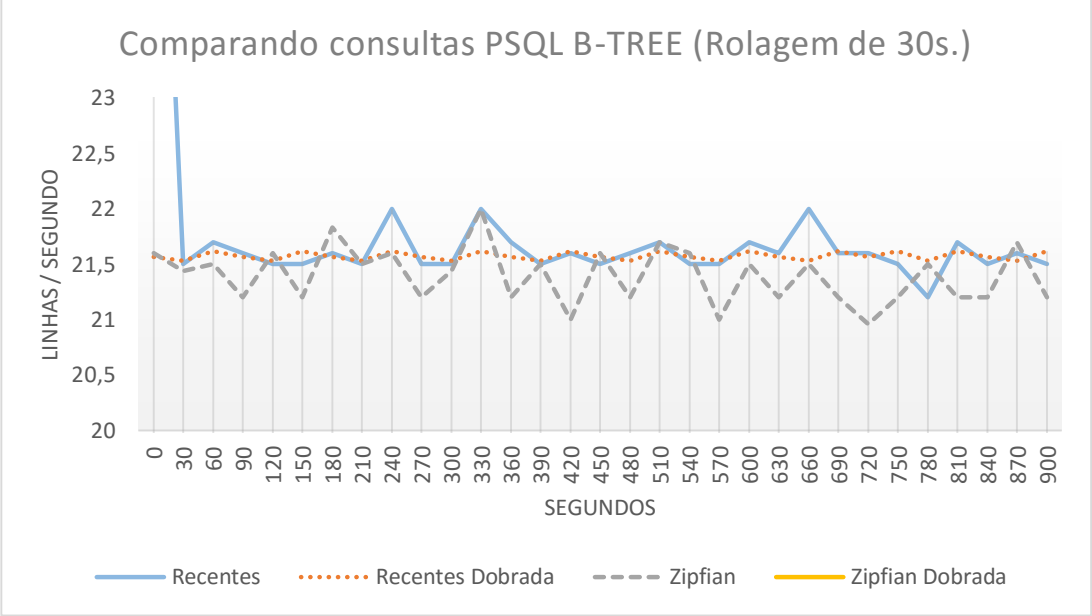


Fonte: (DLUGOKENSKI, 2016)

5.4.3 Comparando os tempos lado a lado

Agora que foi verificado claramente o campeão das consultas, é interessante comparar lado a lado as médias por tipo de consulta realizada, sejam elas de dados recentes, sejam dados recentes com janela de consulta dobrada, ou ainda dados com distribuição de Zipf modificado.

Figura 5.6 – Vazão média nas consultas de dados do PSQL com B-TREE



Fonte: (DLUGOKENSKI, 2016)

Com base na Figura 5.6, podemos verificar que consistentemente a distribuição de Zipf ficou abaixo das demais, com a seleção de dados recentes com janela dobrada em segundo. A consulta *zipfian* com janela dobrada está muito abaixo da linha de corte e foi deixada assim pois o gráfico perderia o detalhamento desejado.

Por fim, esses resultados mostram que, tanto a consulta de dados "aleatórios" no espaço de dados, quanto o tamanho dos dados requisitados para retorno tem influência no tempo de resposta, como era esperado.

5.5 Vazão na ingestão de dados

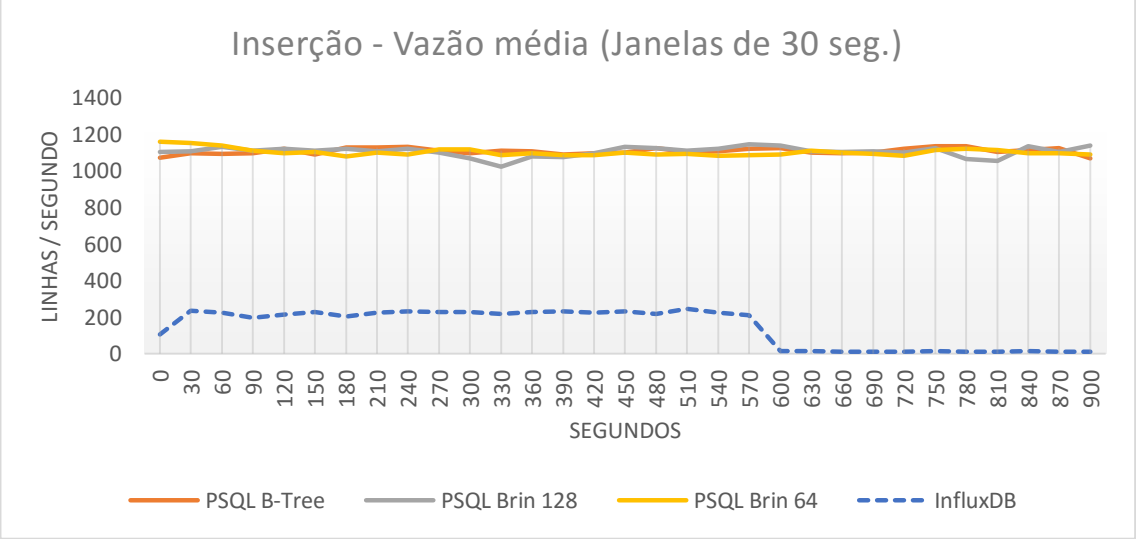
Agora vamos verificar os resultados do teste de ingestão (ou inserção) de dados. Ele é muito importante para entender, entre outros, quantas sondas podem ser ativadas no NetMetric sem que o sistema como um todo deixe de funcionar adequadamente.

Como é possível verificar na Figura 5.7, o PostgreSQL claramente é o vencedor, com uma taxa acima das **1000 linhas por segundo**, enquanto o InfluxDB ficou abaixo das **250 linhas por segundo**.

Na mesma figura, também se percebeu um claro desgaste deste último sistema ao longo do tempo, com a taxa declinando severamente após 570 segundos de início do teste e se mantendo assim por horas. Esse comportamento atípico que levou ao encerramento pré-maturo do teste. Além disso, para os testes seguintes que dependiam desses dados inseridos, foi utilizada uma técnica de importação em lote, que é muito interessante, mas que **não reflete** o comportamento do NetMetric.

Sobre o PostgreSQL, entre as configurações escolhidas para ele, não se viu diferença significativa, com médias aproximadas de **1109 l/s** (B-TREE), de **1107 l/s** (BRIN128) e de **1104 linhas por segundo** (BRIN64).

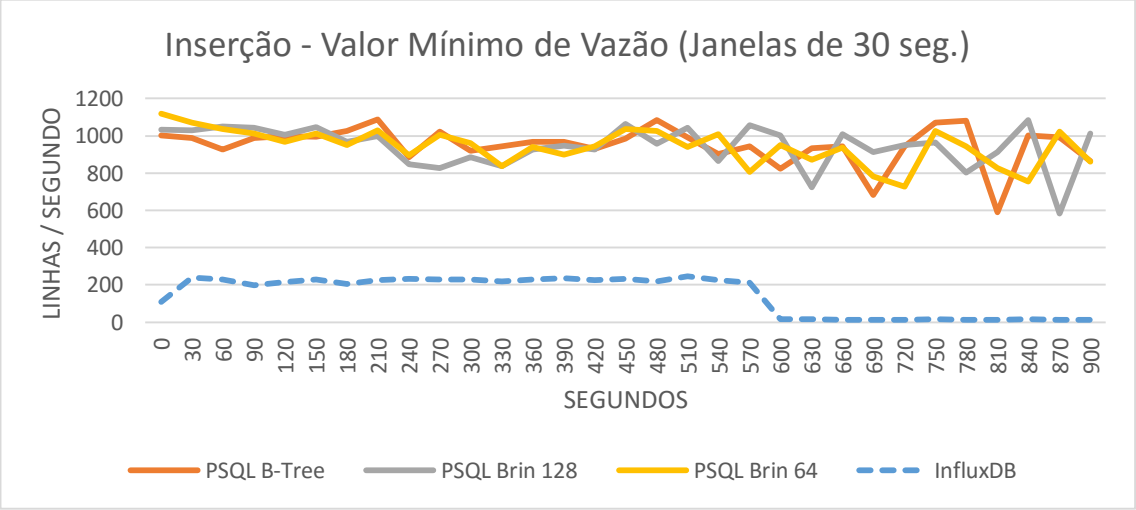
Figura 5.7 – Vazão média na inserção de dados



Fonte: (DLUGOKENSKI, 2016)

Durante a análise dos resultados, outro aspecto que se mostrou interessante é da variação do valor **mínimo** ao longo do tempo, que conforme após um determinado tempo após o início, mostrou uma grande oscilação, representado na Figura 5.8.

Figura 5.8 – Vazão mínima na inserção de dados



Fonte: (DLUGOKENSKI, 2016)

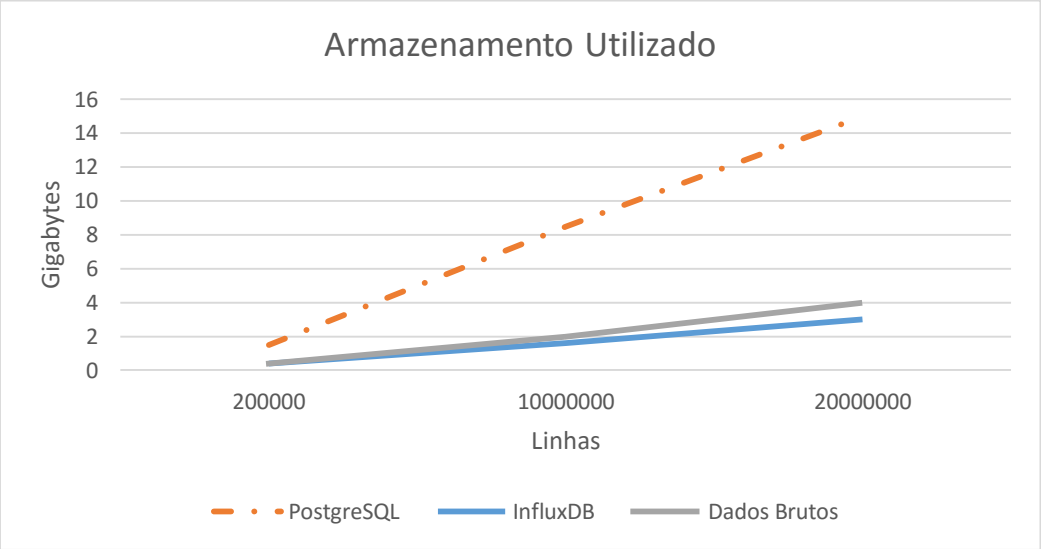
Esse é um aspecto importante na hora de se fazer decisões a respeito da quantidade de sondas no NetMetric que podem ser ativadas simultaneamente, já que uma falta de folga na transmissão ao banco de dados, como aplicado no teste, pode levar a uma oscilação forte, que possivelmente poderia ter um efeito negativo no *webservice* do NetMetric. Nesse caso teriam que se aprofundar os testes de integração entre ambos para verificar se o possível congestionamento não atrapalharia outras funcionalidades do *webservice*.

5.6 Armazenamento utilizado

Por fim, a última métrica avaliada é o armazenamento utilizado, onde o engenheiro que desenha o sistema deve decidir o quanto de armazenamento de dados ele vai precisar, em acordo com as necessidades correntes e futuras do sistema.

Para obter uma ideia geral do crescimento dessa utilização, foram feitos alguns testes com três diferentes tamanhos de conjunto de dados: 2 milhões de linhas, 10 milhões de linhas e 20 milhões de linhas. Na Figura 5.9 estão os resultados.

Figura 5.9 – Armazenamento utilizado pelos sistemas testados



Fonte: (DLUGOKENSKI, 2016)

Aqui, novamente temos dois resultados instigadores, o primeiro deles em relação ao InfluxDB, que ocupa menos espaço em relação aos dados brutos, conforme estes últimos crescem em número.

A explicação deste fato é que o InfluxDB, segundo (MCDONALD, 2016, p. 7,8), mantém os dados armazenados em formato de colunas, ao invés de linhas, que trás diversas oportunidades de compressão de dados similares. Neste trabalho verifica-se que os marcadores (*tags*), em especial, se repetem (nome da sonda, por exemplo), e isto pode ser armazenado de alguma maneira que não se repita diversas vezes em disco.

Na outra ponta temos o PostgreSQL (com índice B-TREE) que aumenta o espaço conforme o crescimento dos dados brutos. É possível que com alguma otimização manual (comando VACUUM, por exemplo) isso se reduza, mas otimizações manuais estão fora do escopo desse trabalho.

Logo, se existe uma grande preocupação com o espaço utilizado, o InfluxDB é uma clara escolha.

6 CONCLUSÕES

Uma das primeiras conclusões que se chegou durante o desenvolvimento desse trabalho, é que a pré-avaliação feita no capítulo 3 é muito importante. O número de sistemas gerenciais de bancos de dados disponíveis é muito grande, e os passos utilizados no dito capítulo foram cruciais para que esse trabalho fosse feito em tempo hábil.

Outra conclusão é que não existem muitas ferramentas de teste livres e facilmente extensíveis. As ferramentas da TPC (TPC - Benchmarks, 2015) apesar de variadas e numerosas, não tem seu software de teste alcançável ao grande público, e, por isso, não foram consideradas como referência.

Após isso, durante os testes, podemos ver que, para as necessidades do NetMetric – e de, possivelmente, outras ferramentas de rede com filosofia similar no quesito armazenamento – o **PostgreSQL**, com o tradicional índice **B-TREE** ainda é a melhor opção; pois, como vemos durante todo capítulo 5, se existe a necessidade dos usuários em fazer consultas a todo instante, e não apenas a ingestão dos dados, claramente temos essa configuração como escolha natural, mesmo levando em conta possíveis limitações em relação ao armazenamento utilizado.

Agora, apenas considerando o campo ingestão de dados, não é possível descartar as outras opções, em especial o InfluxDB, que oferece várias otimizações para tal, que estavam fora do escopo deste trabalho, mas que em determinados momentos foram utilizadas para torná-lo possível ou por pura curiosidade.

Somado ao já dito, se foram consideradas como viáveis possíveis otimizações desses sistemas em futuros estudos, concluo que mais crucial do que fazer os testes neste estudo, foi **a apresentação de um método sólido de avaliação**, algo que foi feito no capítulo 4, baseado em conclusões similares dessa necessidade por (COOPER, SILBERSTEIN, *et al.*, 2010) e (HAN, JIA, *et al.*, 2015).

Assim, automatizando cada vez mais os métodos de teste mostrado neste e nos supracitados trabalhos, espera-se que um número maior de configurações e otimizações se torne fácil de avaliar, e que a especulação seja mais facilmente substituída por dados.

Voltando à questão inicial do trabalho, de qual dos sistemas disponíveis é suficiente ou adequado para o contexto, verifica-se que, como vemos em todo nosso estudo na graduação de Ciência da Computação – e neste trabalho – não existe uma solução que seja simultaneamente simples e eficiente para uma classe de problemas, e que compromissos devem ser assumidos pelo arquiteto do sistema.

Todavia, fica a experiência de que o *benchmarking* é a uma forte ferramenta aliada nessa decisão, e que essas técnicas devem ser valorizadas, pois se dependesse somente do *marketing* que cada uma faz, todas seriam ótimas para tudo, premissa que não é realidade.

REFERÊNCIAS

ANDRÉ-JÖNSSON, H. **Indexing for Time Series Data**. Linköpings Universitet. Linköping, Suécia. 2002. (91-7373-346-6).

BAILIS, P.; GHODSI, A. Eventual Consistency Today: Limitations, Extensions, and Beyond. **ACM Queue**, Nova Iorque, EUA, v. 11, n. 3, Março 2013. ISSN 1542-7730. Disponivel em: <http://dl.acm.org/citation.cfm?doid=2460276.2462076>.

BRADEN, R. **RFC 1122: Requirements for Internet Hosts -- Communication Layers**. Internet Engineering Task Force. [S.l.]. 1989.

BRADEN, R. **RFC 1123: Requirements for Internet Hosts -- Application and Support**. Internet Engineering Task Force. [S.l.], p. 97. 1989.

BRAY, T. **The JavaScript Object Notation (JSON) Data Interchange Format**. Internet Engineering Task Force (IETF). [S.l.]. 2014. (RFC 7159).

BREWER, E. **Towards robust distributed systems**. Principles of Distributed Computing. Portland, EUA: ACM. 2000.

BREWER, E. CAP twelve years later: How the "rules" have changed. **Computer**, Berkeley, CA, EUA, v. 45, n. 2, Fevereiro 2012. Disponivel em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6133253&tag=1>.

CHANAN, G. Apache HBase Internals: Locking and Multiversion Concurrency Control. **Apache's Foundation "Blogging in Action."**, 11 jan. 2013. Disponivel em: <https://blogs.apache.org/hbase/entry/apache_hbase_internals_locking_and>. Acesso em: 10 jan. 2016.

CHANG, F. et al. Bigtable: A Distributed Storage System for Structured Data. **ACM Transactions on Computer Systems**, Nova Iorque, EUA, v. 26, n. 2, Junho 2008. ISSN 0734-2071.

CODD, E. F. A relational model of data for large shared data banks. **Communications of the ACM**, Nova Iorque, EUA, 13, n. 6, 1970. 377-387. Disponivel em: <http://dl.acm.org/citation.cfm?id=362685>.

COOPER, B. F. et al. **Benchmarking Cloud Serving Systems with YCSB**. Proceedings of the 1st ACM symposium on Cloud computing. Indianapolis, EUA: ACM. 2010. p. 143-154.

DIX, P. InfluxDB Clustering Design – neither strictly CP or AP. **Blog InfluxData**, 3 Junho 2015. Disponivel em: <https://influxdata.com/blog/influxdb-clustering-design-neither-strictly-cp-or-ap/>. Acesso em: 04 jan. 2016.

DUNNING, T.; FRIEDMAN, E. **Time Series Databases:** New Ways to Store and Access Data. 1ª. ed. Sebastopol, EUA: O'Reilly Media, Inc., 2015.

EKELIN, S. et al. **Real-Time Measurement of End-to-End Available Bandwidth using Kalman Filtering**. Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP. Vancouver, Canadá: IEEE. 2006. p. 73-84.

FEINBERG, D. et al. Magic Quadrant for Operational Database Management Systems. **Site da Gartner, Inc.**, 12 Outubro 2015. Disponivel em: <http://www.gartner.com/technology/reprints.do?id=1-2PMFPEN&ct=151013&st=sb>. Acesso em: 29 Dezembro 2015.

FLANAGAN, D.; MATSUMOTO, Y. **The Ruby Programming Language**. 1ª. ed. Sebastopol, CA, EUA: O'Reilly Media Inc., 2008.

GOOGLE INC. Snappy: A fast compressor/decompressor. **Snappy by google**, 2016. Disponivel em: <http://google.github.io/snappy/>. Acesso em: 04 jan. 2016.

GRAPHITE PROJECT. Does Graphite use RRDtool? **Graphite FAQ**, 2015. ISSN Revision 793f11af. Disponivel em: <http://graphite.readthedocs.org/en/latest/faq.html#does-graphite-use-rrdtool>. Acesso em: 11 jan. 2016.

GROLINGER, K. et al. Data management in cloud environments: NoSQL and NewSQL data stores. **Journal of Cloud Computing: Advances, Systems and Applications**, v. 2, 2013. Disponivel em: <http://www.journalofcloudcomputing.com/content/2/1/22/abstract>.

HAN, R. et al. **Benchmarking Big Data Systems: State-of-the-Art and Future Directions**. Institute of Computing Technology. Beijing, China. 2015.

HSIEH, M.-Y. A. et al. **Towards the deployment of a mobile robot network with end-to-end performance guarantees**. Proceedings of the 2006 IEEE International Conference on Robotics and Automation. Orlando, EUA: IEEE. 2006. p. 2085-2090.

HYKES, S. Docker 0.9: introducing execution drivers and libcontainer. **Docker Blog**, 2014. Disponivel em: <http://blog.docker.com/2014/03/docker-0-9-introducing-execution-drivers-and-libcontainer/>. Acesso em: 01 maio 2016.

INFLUXDATA INC. Storage Engine. **InfluxDB Docs v0.9**, 2015. Disponivel em: <https://influxdb.com/docs/v0.9/concepts/storage_engine.html>. Acesso em: 11 jan. 2016.

INFLUXDATA INC. Comparison to SQL. **InfluxData | Documentation**, 2015. Disponivel em: <https://docs.influxdata.com/influxdb/v0.9/concepts/crosswalk/>. Acesso em: 04 jan. 2016.

INFLUXDATA INC. Line Protocol. **InfluxData | Documentation**, 2015. Disponivel em: <https://docs.influxdata.com/influxdb/v0.9/write_protocols/line/>. Acesso em: 01 abr. 2016.

INFLUXDATA INC. Design Insights and Tradeoffs in InfluxDB. **InfluxData | Documentation**, 2016. Disponivel em: <https://docs.influxdata.com/influxdb/v0.9/concepts/insights_tradeoffs/>. Acesso em: 04 jan. 2016.

KAPILA, A. Well-known Databases Use Different Approaches for MVCC. **Postgres Plus EDB Blog**, 2015. Disponivel em: <http://www.enterprisedb.com/postgres-plus-edb-blog/amit-kapila/well-known-databases-use-different-approaches-mvcc>. Acesso em: 10 jan. 2016.

KUROSE, J. F.; ROSS, K. W. **Computer Networking:** A Top-Down Approach. 6ª. ed. [S.l.]: Pearson, 2012. ISBN ISBN-13: 9780132856201.

LAI, K.; BAKER, M. **Measuring link bandwidths using a deterministic model of packet delay**. SIGCOMM'00 Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. Nova Iorque, EUA: ACM. 2000. p. 283-294.

MAIER, G.; KAUFMANN, A. The development of computer networks: First results from a microeconomic model. **Journal of Geographical Systems**, v. 3, n. 2, p. 155-166, 31 ago. 2001. ISSN 1435-5930.

MCDONALD, C. An In-Depth Look at the HBase Architecture. **MAPR Blog**, 7 jul. 2015. Disponivel em: <https://www.mapr.com/blog/in-depth-look-hbase-architecture>. Acesso em: 11 jan. 2016.

MCDONALD, R. **InfluxDB Performance Tuning Tips**. InfluxData Inc. [S.l.]. 2016.

MICROSOFT. Create Clustered Indexes. **Microsoft Developer Network**, 2016. Disponivel em: <https://msdn.microsoft.com/en-us/library/ms186342.aspx>. Acesso em: 11 jan. 2016.

OETIKER, T. rrdtutorial. **Site oficial do RRDTool**, 08 Setembro 2015. Disponivel em: <http://oss.oetiker.ch/rrdtool/tut/rrdtutorial.en.html>. Acesso em: 29 Dezembro 2015.

OPENTSDB. Writing Data. **OpenTSDB 2.1 documentation**, 2015. Disponivel em: <http://opentsdb.net/docs/build/html/user_guide/writing.html>. Acesso em: 05 Janeiro 2016.

ORACLE. Clustered and Secondary Indexes. **MySQL 5.7 Reference Manual**, 2015. Disponivel em: <https://dev.mysql.com/doc/refman/5.7/en/innodb-index-types.html>. Acesso em: 11 jan. 2016.

ORACLE. Overview of Indexed Clusters. **Database Online Documentation**, 2015. Disponivel em: <https://docs.oracle.com/database/121/CNCPT/tablecls.htm#CNCPT-GUID-CC31365B-83B0-4E09-A047-BF1B79AC887A>. Acesso em: 11 jan. 2016.

PÁSZTOR, A. **Accurate Active Measurement in the Internet**. University of Melbourne. Melbourne, Austrália. 2003.

POSTGRESQL WEB TEAM. What's new in PostgreSQL 9.5. **PostgreSQL Wiki**, 2016. Disponivel em: <https://wiki.postgresql.org/wiki/What's_new_in_PostgreSQL_9.5#BRIN_Indexes>. Acesso em: 11 jan. 2016.

PRITCHETT, D. BASE: An Acid Alternative. **ACM Queue**, Nova Iorque, EUA, v. 6, n. 3, p. 48-55, Maio/Junho 2008. ISSN 1542-7730.

SHASHA, D.; ZHU, Y. **High Performance Discovery In Time Series:** Techniques and Case Studies. 1ª. ed. Nova Iorque, EUA: Springer Science & Business Media, 2013. ISBN 978-1-4757-4046-2.

SMITH, J. E.; NAIR, R. The Architecture of Virtual Machines. **Computer**, v. 38, n. 5, p. 32-38, Maio 2005. ISSN 0018-9162.

SOLID IT GMBH. DB-Engines Ranking - popularity ranking of relational DBMS. **DB-Engines Ranking**, 2016. Disponivel em: <http://db-engines.com/en/ranking/relational+dbms>. Acesso em: 08 jan. 2016.

SOLID IT GMBH. DB-Engines Ranking - popularity ranking of time Series DBMS. **DB-Engines Ranking**, 2016. Disponivel em: <http://db-engines.com/en/ranking/time+series+dbms>. Acesso em: 08 jan. 2016.

SUMATHI, S.; ESAKKIRAJAN, S. **Fundamentals of Relational Database Management Systems**. Berlim, Alemanha: Springer, v. 47, 2006. ISBN ISBN-13 978-3-540-48397-7.

TAY, Y. C. **Data Generation for Application-Specific Benchmarking**. Challenges and Visions of International Conference on Very Large Databases. Seattle, EUA: VLDB Endowment. 2011. p. 1470-1473.

TPC - Benchmarks. **TPC - Benchmarks**, 2015. Disponivel em: <http://www.tpc.org/information/benchmarks.asp>. Acesso em: 15 out. 2015.

TRUBETSKOY, G. Time Series Accuracy - Graphite vs RRDTool. **Notes to self.**, 4 maio 2015. Disponivel em: <http://grisha.org/blog/2015/05/04/recording-time-series/>. Acesso em: 11 jan. 2016.

W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition). **Site oficial da W3C**, 26 Novembro 2008. Disponivel em: <http://www.w3.org/TR/2008/REC-xml-20081126/>. Acesso em: 04 jan. 2016.

WIKIPEDIA. SQL. **Wikipedia, the free encyclopedia**, 25 dez. 2015. Disponivel em: <https://en.wikipedia.org/w/index.php?title=SQL&oldid=696732436>. Acesso em: 04 jan. 2016.

APÊNDICE: SOFTWARE DE TESTE

Código criado para os testes que foi citado diversas vezes neste trabalho, encontra-se em:

- https://bitbucket.org/rdlu/rdlu-database-test

No local citado existe um LEIAME com informações úteis no provisionamento de máquinas de teste e procedimentos e comandos adotados, além do código desenvolvido pelo autor deste trabalho utilizando todas as premissas mostradas ao longo trabalho.

É requerido da pessoa interessada que ela saiba ou utilizar GIT, ou baixar manualmente a última versão na seção de *downloads* da localização supracitada.