UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CAMPUS CURITIBA - CENTRO
CURSO DE ENGENHARIA DE COMPUTAÇÃO

RUBIA TERUMY ZAVADIL ISOBE

ANÁLISE DE DESEMPENHO DE MIDDLEWARES PARA APLICAÇÕES WEB

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2021

RUBIA TERUMY ZAVADIL ISOBE

ANÁLISE DE DESEMPENHO DE MIDDLEWARES PARA APLICAÇÕES WEB

Trabalho de Conclusão apresentado ao Curso de Engenharia de Computação como requisito para a obtenção do título de Bacharel em Engenharia de Computação.

Universidade Tecnológica Federal do Paraná

Orientador: Profª. Drª. Ana Cristina Barreiras Kochem Vendramin

Coorientador: Prof. Dr. Daniel Fernando Pigatto

CURITIBA

2021

RUBIA TERUMY ZAVADIL ISOBE

ANÁLISE DE DESEMPENHO DE MIDDLEWARES PARA APLICAÇÕES WEB

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 18 de junho de 2021

ANA CRISTINA BARREIRAS KOCHEM VENDRAMIN
Doutorado em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná
Universidade Tecnológica Federal do Paraná (UTFPR)

LUIZ NACAMURA JÚNIOR
Doutorado em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná
Universidade Tecnológica Federal do Paraná (UTFPR)

PAULO ROBERTO BUENO
Doutorado em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná
Universidade Tecnológica Federal do Paraná (UTFPR)

CURITIBA

2021

RESUMO

Serviços web são a principal *middleware* utilizada no desenvolvimento de aplicações web, possibilitando a comunicação entre as aplicações e facilitando a disponibilização dos serviços na internet. SOAP (*Simple Object Access Protocol*) e REST (*Representational State Transfer*) são formas de implementação de serviços web muitos popularizadas nos últimos anos, empregando métodos padronizados de comunicação, como o XML (*Extensible Markup Language*). Este trabalho propõe analisar o desempenho das *middlewares* SOAP e REST sobre um *benchmark* de comércio virtual. Foi desenvolvida uma aplicação *benchmark* baseada em sistemas reais, utilizando a linguagem Java. O número de acessos de clientes foi variado de forma crescente. Foi analisado o desempenho das *middlewares* em relação à utilização de CPU, memória RAM, disco rígido, rede e tempo de execução de uma rotina de requisições. Esta análise identificou as situações nas quais cada *middleware* apresenta um melhor desempenho, podendo orientar a escolha entre SOAP e REST em futuras implementações.

Palavras-chave: Serviços Web; Análise de Desempenho; SOAP; REST.

ABSTRACT

Web services are the main middleware used in web application development, enabling communication between applications and easing the availability of services over the internet. SOAP (Simple Object Access Protocol) and REST (Representational State Transfer) are ways of implementing web services that have become very popular in recent years, using standardized methods of communication, such as XML (Extensible Markup Language). This paper proposes a performance analysis of SOAP and REST middlewares on an e-commerce benchmark. A benchmark application was developed based on real systems, using Java language. The number of customer accesses was gradually increased. The performance of the middlewares was analyzed considering the usage of CPU, RAM memory, disk, network, and execution time of a request routine. This analysis identified the situations in which each middleware performs better, being able to guide the choice between SOAP and REST in future implementations.

Keywords: Web Services; Performance Analysis; SOAP; REST.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

SUMÁRIO

1 INTRODUÇÃO

O estudo e desenvolvimento de sistemas distribuídos é de extrema importância para a computação moderna. Segundo pesquisas, o crescimento do comércio virtual, ou *e-commerce*, aumenta rapidamente a cada ano. Em 2020, em torno de 2 bilhões de pessoas compraram itens ou serviços online, com o total de vendas ultrapassando 4,2 trilhões de dólares ao redor do mundo (STATISTA, 2021). Só no Brasil, o faturamento das vendas em *e-commerce* cresceu 41% em 2020 (INFOMONEY, 2021).

No entanto, existem muitos desafios no desenvolvimento de aplicações distribuídas. Entre eles, existe a heterogeneidade das aplicações e sistemas. Afinal, existem diversas linguagens de programação, sistemas operacionais e configurações de rede e hardware. Uma forma de abstrair a heterogeneidade é com o emprego de *middlewares*. Uma *middleware* é uma camada de software que provê um modelo computacional uniforme para uso pelos programadores de aplicações distribuídas (COULOURIS et al., 2012). Todas as *middlewares* lidam com diferenças de sistemas operacionais e hardware. A maioria das *middlewares* é implementada sobre protocolos Internet, os quais isolam diferenças de rede. Porém, apenas algumas *middlewares*, como os serviços web, suportam diferentes linguagens de programação.

Serviços web são a principal *middleware* utilizada no desenvolvimento de aplicações web, possibilitando a comunicação entre as aplicações e facilitando a disponibilização dos serviços na internet. SOAP e REST são exemplos de *middlewares* popularmente utilizados em aplicações web, empregando métodos padronizados de comunicação como XML (*Extensible Markup Language*) e JSON (*JavaScript Object Notation*) no caso do REST.

Ao iniciar o projeto de uma aplicação distribuída é preciso considerar as especificações do projeto, como protocolo de comunicação, formato de dados e disponibilização dos serviços. Além disso, é preciso levar em conta os recursos disponíveis. SOAP e REST são *middlewares* com conceitos de funcionamento muito distintos, o que também reflete na utilização dos recursos por essas tecnologias.

Diante desse cenário, este trabalho tem como objetivo realizar uma análise de desempenho das *middlewares* SOAP e REST em uma aplicação *benchmark* de *e-commerce*. Esta análise visa comparar cada tecnologia, em diferentes contextos de acesso de clientes, e identificar em quais situações cada *middleware* apresenta um melhor desempenho, podendo orientar a escolha em futuras implementações.

1.1 OBJETIVOS

O objetivo geral do trabalho é analisar o desempenho das *middlewares* SOAP e REST, considerando como métricas a utilização de CPU, memória RAM, disco rígido, rede e tempo

de execução. Para isso, foram definidos os seguintes objetivos específicos:

- Apresentar os conceitos de sistemas distribuídos e serviços web, incluindo a caracterização das *middlewares* para serviços web SOAP e REST;
- Estudar sobre análise de desempenho de sistemas, incluindo a implementação de *benchmarks*, ferramentas de coletas de dados e emulador de clientes;
- Definir, projetar e desenvolver duas aplicações web, empregando as *middlewares* SOAP e REST, com base em um *benchmark* de *e-commerce*;
- Analisar o desempenho das aplicações utilizando diferentes volumes de acesso, considerando as métricas de desempenho: uso de CPU, memória RAM, disco, rede e tempo de execução;
- Comparar os dados obtidos através de gráficos, buscando identificar os casos em que uma *middlware* para serviço web é mais vantajosa em relação à outra.

1.2 ORGANIZAÇÃO DO DOCUMENTO

O presente trabalho está organizado em seis capítulos. O capítulo 1 contextualiza a pesquisa e estabelece os objetivos. O capítulo 2 compreende os conceitos envolvidos na análise de desempenho e nas tecnologias avaliadas. O capítulo 3 define o projeto dos serviços web, incluindo os diagramas de casos de uso, diagrama de classes e diagrama de fluxo, e o projeto do simulador de clientes. O capítulo 4 descreve a realização do experimento que inclui as etapas de preparação, execução e avaliação. O capítulo 5 apresenta os resultados alcançados nos experimentos. O capítulo 6 apresenta as conclusões e alguns trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica sobre sistemas distribuídos e a *middleware* serviços web, define os métodos e os conceitos para análise de desempenho de sistemas e descreve alguns trabalhos relacionados.

2.1 SISTEMAS DISTRIBUÍDOS

Segundo Coulouris et al. (2012), um sistema distribuído é aquele no qual componentes de software e hardware localizados em computadores conectados entre si através da rede se comunicam e coordenam suas ações utilizando troca de mensagens. Esses computadores podem estar fisicamente separados por qualquer distância.

Como características de um sistema distribuído, tem-se a concorrência entre seus componentes, a ausência de relógio global e as falhas de componentes independentes (COULOURIS et al., 2012). A execução concorrente de componentes significa que recursos podem ser acessados por diferentes computadores de forma transparente, ou seja, sem afetar um ao outro. A coordenação de diferentes computadores é feita apenas através da troca de mensagens. Não sendo possível ter uma noção global única de tempo correto, há limites de precisão para a sincronização de relógios em uma rede. Outra característica importante de um sistema distribuído é a possibilidade de isolamento das falhas, pois cada componente pode falhar independentemente, permitindo que outros continuem funcionando.

A motivação para a criação e utilização de sistemas distribuídos é o compartilhamento de recursos. Um recurso pode ser qualquer elemento que possa ser compartilhado em uma rede e que traga alguma utilidade para as partes envolvidas. As páginas web, banco de dados, vídeos e aúdios são exemplos de recursos de software que podem ser compartilhados. No caso de recursos de hardware, pode-se citar os discos e as impressoras.

O termo serviço é utilizado para a parte do sistema que gerencia uma coleção de recursos e disponibiliza suas funcionalidades para usuários e aplicações (COULOURIS et al., 2012). O papel de um serviço é limitar os recursos a ações, ou operações, específicas e bem definidas. Por exemplo, um serviço de arquivo disponibiliza operações como visualizar, criar e deletar arquivos. Servidores são programas, ou processos, dentro de um computador conectado à rede que recebem requisições de outros computadores, os clientes, para executar serviços (COULOURIS et al., 2012). Esta abordagem é chamada de cliente-servidor. Clientes possuem um papel "ativo", realizando requisições, enquanto servidores são "passivos", aguardando continuamente requisições.

A aplicação de sistemas distribuídos é ampla e ocorre em diversas esferas de negócio. Como exemplo, pode-se mencionar os setores: (a) finanças e comércio, ou *e-commerce*:

companhias como Amazon e Ebay para venda de produtos, a plataforma de pagamentos PayPal e bancos online (*online banking*); (b) informação: crescimento da *Word Wide Web* como um grande repositório, ferramentas de busca como Google, criação e disponibilização de conteúdos com YouTube, Facebook, Instagram, Google Drive e Dropbox; (c) entretenimento: jogos online e plataformas de *streaming* como Netflix, Amazon Prime, Disney+ e Spotify; (d) educação: plataformas de ensino remoto e suporte ao ensino a longas distâncias; (e) transporte: tecnologias de localização como GPS (*Global Positioning System*), Google Maps e Waze; entre outros.

O estudo e desenvolvimento de sistemas distribuídos é de extrema importância para a computação moderna. No entanto, existem muitos desafios no desenvolvimento de aplicações distribuídas. Um desses desafios é a heterogeneidade, ou seja, as diferenças entre sistemas em diversas camadas como linguagem de programação, sistema operacional, rede e hardware. Outro desafio é a escalabilidade, a capacidade de um sistema distribuído se manter eficiente com o aumento do número de usuários. O gerenciamento de falhas também se torna mais complexo, apesar da comunicação continuar funcionando mesmo com falhas de computadores independentes. Com recursos sendo disponibilizados através da rede, manter a segurança destes recursos também se torna mais desafiador. Concorrência e transparências de acesso e localização são exemplos de outros desafios em sistemas distribuídos.

De forma a abstrair do programador os aspectos de heterogeineidade, surgiram as *middlewares*. A *middleware* é uma camada de software que provê um modelo computacional uniforme para uso pelos programadores de aplicações distribuídas (COULOURIS et al., 2012). Dentre os modelos possíveis, tem-se: invocação de objeto remoto, serviço de nomes, Segurança, notificação de evento remoto, processamento de transação distribuída, entre outros. Como exemplos de *middlewares*, pode-se citar: o Java RMI (CORPORATION, 2021) ou *Remote Method Invocation*, PyRO (JONG, 2021) ou *Python Remote Object*, CORBA (INC., 2021b) ou *Common Object Request Broker*, RPC ou *Remote Procedure Call*, serviços web, gRPC (AUTHORS, 2021), entre outros. Todas as *middlewares* lidam com diferenças de sistemas operacionais e hardware. A maioria das *middlewares* é implementada sobre protocolos Internet, os quais mascaram diferenças de rede. Algumas *middlewares* como o Java RMI e PyRO suportam apenas uma única linguagem de programação. Outras, como o CORBA, gRPC e os serviços web suportam diferentes linguagens de programação. O objetivo deste trabalho é analisar o desempenho de dois serviços web que estão descritos a seguir.

2.1.1 Serviços web

Serviços Web (do inglês *Web Services*), de forma geral, consistem de uma coleção de operações que são utilizadas por clientes pela internet (COULOURIS et al., 2012). Serviços web são *middlewares* voltadas para aplicações web distribuídas, com a função de abstrair a complexidade do desenvolvimento destas aplicações.

Uma das maiores vantagens dos serviços web é a interoperabilidade, ou seja, a capacidade de comunicação de forma transparente com outros sistemas semelhantes ou não. Além disso, pode-se citar a alta capacidade de integração de sistemas, especialmente entre empresas (*business-to-business* ou B2B), considerando que os ambientes e tecnologias costumam ser distintos e a localização entre eles distante. Outra vantagem é o fraco acoplamento (*loose coupling*) entre os serviços web, que se refere à minimização de dependências entre si. Isso resulta em uma arquitetura mais flexível e com risco reduzido (COULOURIS et al., 2012).

A comunicação com serviços web é baseada em requisições e respostas, geralmente formatadas em XML, que são transmitidas utilizando algum protocolo de comunicação, como o HTTP.

O *Extensible Markup Language* (XML) é um modelo de representação de dados estruturado e padronizado. Com XML, um dado é associado com seu significado, ou seja, é possível representar qualquer estrutura de dados e objetos. Devido à sua grande flexiblidade, é preciso garantir que todas as definições sejam padronizadas utilizando um esquema (*schema*) e este compartilhado entre todos os sistemas ou aplicações envolvidas (NEWCOMER, 2002). Por ser uma linguagem textual, a comunicação torna-se mais legível para as pessoas, mas ocasiona em um custo maior de espaço de armazenamento e, por consequência, mais tempo de processamento. Atualmente, existem outras formas de representação de dados, como o *JavaScript Object Notation* (JSON), que podem ser empregadas por serviços web. O JSON utiliza uma estrutura mais compacta de representação, tornando o pacote de dados transportado muito menor e a tradução desses dados dentro da aplicação (*parsing*) mais rápida (JSON, 2021).

Em conjunto com os serviços web, são empregados protocolos de comunicação para transportar mensagens e abstrair aspectos da rede. Alguns exemplos de protocolos são HTTP e *Simple Mail Transfer Protocol* (SMTP). O *HyperText Transfer Protocol* (HTTP) é um protocolo do tipo requisição-resposta (*request-reply*). Isto é, clientes enviam uma requisição para o servidor, contendo o endereço do recurso, o servidor consulta o endereço e retorna uma resposta com o conteúdo do recurso solicitado. O HTTP define alguns métodos, ou operações, para gerenciamento de recursos web: (a) GET: acessar as informações de determinado recurso; (b) POST: fornecer informações a determinado recurso; (c) PUT: editar um recurso; e (d) DELETE: deletar um recurso. O endereço, definido como *Uniform Resource Locators* (URL), é utilizado para identificar os recursos e seu acesso causa a execução de algum processo pelo servidor (COULOURIS et al., 2012). De forma geral, o *Unique Resource Identifier* (URI) é o identificador, enquanto o URL, além de identificador, possui informações de como acessar o recurso.

Os dois estilos principais de interação com serviços web são: chamada de procedimento remoto (RPC ou *Remote Procedure Call*) e orientação a documento. A primeira forma de interação acontece na forma de invocação de método ou procedimento associado a parâmetros

de entrada e saída. Ou seja, as informações são mapeadas facilitando a transmissão de mensagens curtas com informações breves e objetivas. Por outro lado, a interação orientada a documentos é mais recomendada para transmissão de uma grande quantidade de informação, sendo o documento transmitido por completo em XML (NEWCOMER, 2002).

É preciso diferenciar servidores web e serviços web. Segundo Coulouris et al. (2012), servidores web disponibilizam um serviço HTTP básico, com seu conjunto de métodos, enquanto os serviços web disponibilizam serviços baseados em operações que podem estar definidas em uma interface. Um serviço web pode ser gerenciado por um servidor web, junto com páginas web, ou pode ser um serviço totalmente separado.

Existem duas formas de permitir a comunicação com serviços web. Uma das abordagens é utilizando o SOAP, na qual as mensagens formatadas em XML são encapsuladas como envelopes e existe uma interface com as definições das operações disponíveis para gerenciamento dos recursos. Alternativamente, pode ser utilizado o REST, cuja ênfase está na manipulação dos recursos através das próprias operações HTTP disponíveis, sem utilizar uma interface. O REST permite o uso tanto de XML como de JSON para representar as mensagens. O objetivo deste trabalho é comparar o desempenho dessas duas tecnologias que estão detalhadas na sequência.

2.1.2 SOAP

O *Simple Object Access Protocol* (SOAP) é um protocolo que especifica regras para o uso de XML para formatar mensagens e transmiti-las através da rede utilizando HTTP ou outros protocolos, como SMTP (COULOURIS et al., 2012). A primeira versão do SOAP 1.0 foi criada em 1999 pela *World Wide Web Consortium* (W3C). Atualmente, a versão mais recente é a 1.2, lançada em 2007 (W3C, 2021). A especificação SOAP suporta a troca de mensagens do tipo requisição-resposta (*request-reply*), característico de chamadas de procedimentos remotos, assim como o transporte de documentos completos formatados em XML (NEWCOMER, 2002).

A estrutura XML de uma mensagem SOAP é composta de um envelope, que contém outros dois elementos: um cabeçalho e um corpo. O cabeçalho (*header*) é opcional e contém informações necessárias para o processamento da mensagem, que está contida no corpo (*body*). Um exemplo de mensagem SOAP está contido na Figura 1. O envelope é chamado de *Envelope* e o elemento *xmlns:soapenv* indica o *namespace* que define um envelope SOAP. A mensagem não possui cabeçalho (*Header*), como pode ser observado pelo elemento XML vazio. O corpo (*Body*) contém a descrição do serviço (*home*) a ser executado, junto com os parâmetros (*token*). A definição completa do serviço está definida no esquema XML e seu endereço é referenciado pelo elemento *xmlns:soap*.

O envelope SOAP é inteiramente encapsulado por uma requisição, por exemplo HTTP. Portanto, a mensagem e seu conteúdo são independentes da escolha do protocolo de comunicação. De forma análoga, a requisição HTTP também contém cabeçalho e corpo. O cabeçalho

Figura 1: Exemplo de mensagem SOAP.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soap="http://soap/">
   <soapenv:Header/>
   <soapenv:Body>
      <soap:home>
        <token>1000</token>
      </soap:home>
   </soapenv:Body>
</soapenv:Envelope>
```

Fonte: Fonte própria

contém informações como: o método HTTP, sendo GET e POST as escolhas mais comuns; a ação ou operação a ser executada, contendo o endereço do recurso; e o tipo de conteúdo esperado da resposta.

Para permitir a comunicação entre sistemas distintos, é preciso garantir a padronização dos elementos compartilhados. O SOAP utiliza interfaces para fornecer uma descrição completa das operações disponíveis pelo serviço web. O *Web Services Description Language* (WSDL) é um documento XML que permite que clientes interajam de forma correta com os serviços web de forma independente da plataforma e da linguagem de programação. Segundo Curbera (2002), o WSDL é composto de duas partes. Uma delas é a descrição abstrata, que contém a definição das estruturas e tipos de dados e a descrição de como deve ser a interação através de requisições e respostas, incluindo a descrição das mensagens envolvidas nessa troca. A outra é uma decrição concreta (também chamada de *service bindings*), que contém o protocolo de comunicação utilizado, assim como o endereço de acesso às operações disponíveis. O WSDL pode ser utilizado em ambos os estilos de serviços web, tanto com chamadas de procedimento remoto quanto interações baseadas em documento.

Uma vez que os serviços são devidamente descritos, estas especificações devem ser disponibilizadas para integrar com outras aplicações e sistemas. *Universal Description, Discovery and Integration* (UDDI) é um serviço de diretório responsável pela disponibilização dessas especificações. O UDDI fornece operações para registrar (publicar) e buscar (descobrir) informações sobre serviços web, como suas interfaces WSDL (ALONSO et al., 2004). Além da interface WSDL e sua localicação (URL), um registro de um serviço web também pode incluir informações como sua localização geográfica, companhia e negócios, descrição dos produtos oferecidos, entre outras.

2.1.3 REST

Representational State Transfer (REST) é um estilo de arquitetura de serviços web que se tornou muito popular para disponibilização de serviços na web. Atualmente, é utilizado por grandes empresas como Amazon (AMAZON.COM, 2021b), Google (GOOGLE, 2021) e GitHub (GITHUB, 2021).

O REST surgiu através do estudo de arquiteturas de software baseadas em rede e possui alguns princípios para sua implementação, como a utilização apenas do URI para identificação e interações com recursos e, usualmente, o protocolo HTTP para a transferência de mensagens. Segundo Fielding (2000), autor deste estudo, e Neumann, Laranjeiro e Bernardinho (2018) os princípios para uso do REST são:

- Estar de acordo com o modelo cliente-servidor, onde o servidor detém os recursos e o cliente envia requisições para acessá-los;
- Não armazenamento de estado (*stateless*) do cliente entre requisições. A requisição do cliente deve conter todas as informações necessárias para a interação com o servidor;
- Respostas do servidor devem ser definidas como armazenável em memória *cache* (*cacheable*) ou não;
- Uniformização de sua interface que corresponde às propriedades: a) habilidade de endereçar recursos, onde cada recurso deve possuir um identificador (URL); b) manipulação de recursos através de representações; c) utilização de mensagens autodescritivas e d) as respostas do servidor fornecem links para recursos relacionados (*Hypermedia As The Engine Of Application State* (HATEOS)). Esses links de recursos fornecem aos clientes outras ações disponíveis com base no estado atual do cliente, o qual funciona como um sistema de navegação através da API;
- O princípio de sistema em camadas no qual um cliente não tem como saber se está conectado a um servidor final ou a um servidor intermediário. De modo a simplificar a arquitetura do sistema, essas camadas devem estar transparentes para o cliente;
- Disponibilizar código apenas por demanda. O servidor deve enviar código (por exemplo, códigos *JavaScript*) para o cliente apenas quando necessário que o mesmo seja executado localmente.

Apesar da existência de princípios bem definidos, o REST é apenas um estilo de arquitetura e não possui especificações obrigatórias para a sua implementação. Como resultado, muitas decisões de implementação devem ser tomadas por desenvolvedores para disponibilização de serviços através das APIs. Uma Web *Application Programming Interface* (API), termo utilizado em muitos trabalhos (MALESHKOVA, 2014) (NEUMANN; LARANJEIRO; BERNARDINHO, 2018), é a forma de disponibilizar os serviços web REST. Desenvolvedores de serviços web REST devem definir todos os aspectos de integração com clientes, resultando em uma grande diversidade de implementações, na maioria das vezes que divergem em alguns dos princípios citados acima (NEUMANN; LARANJEIRO; BERNARDINHO, 2018).

Diferente do SOAP, que suporta apenas mensagens XML, o REST permite diversos formatos de mensagens, como XML, JSON, *HyperText Markup Language* (HTML), *Portable*

Document Format (PDF), *Microsoft Excel Spreadsheet* (XLS), entre outros. O JSON é a alternativa mais utilizada hoje e sua popularidade pode ser explicada pelo crescimento da linguagem *Javascript* nas aplicações atuais (NEUMANN; LARANJEIRO; BERNARDINHO, 2018). Esta flexibilidade para tratamento de dados é uma vantagem do REST em comparação com o SOAP. Assim como com o SOAP, informações podem ser enviadas no corpo de uma requisição HTTP, por exemplo uma representação do recurso a ser editado, mas este não é obrigatório. Na maioria das requisições, apenas o URL é necessário para o entendimento da ação desejada pelo cliente sobre um recurso.

A documentação é um dos maiores desafios para utilização de serviços web REST. Sem a existência de padrões para a escrita e disponibilização da documentação das APIs, o uso dos serviços se torna muito limitado. Uma das soluções para este problema é a utilização de ferramentas de software, que além de gerar uma documentação da aplicação, pode auxiliar no planejamento e testes. Um exemplo muito popular de ferramenta com essas funcionalidades é o SwaggerUI (SOFTWARE, 2021). As documentações geradas podem ser consideradas como uma alternativa para as interfaces de descrição de serviços web, como o WSDL para o SOAP (NEUMANN; LARANJEIRO; BERNARDINHO, 2018).

2.2 ANÁLISE DE DESEMPENHO DE SISTEMAS

A análise de desempenho de sistemas é um interesse comum de usuários, administradores e desenvolvedores de sistemas com o objetivo de obter o melhor desempenho ao menor custo. A análise de desempenho é importante ao longo de todo ciclo de vida de uma aplicação ou sistema, pois mostra o quão bem está o seu comportamento e se existe alguma melhoria a ser feita.

Segundo Jain (1991), existe uma abordagem sistemática para executar uma análise de desempenho de um sistema:

1. Definir um objetivo e o sistema: o primeiro passo é definir o objetivo do estudo e o sistema a ser avaliado, incluindo seus limites;

2. Listar os serviços e seus retornos: uma aplicação possui um conjunto de serviços que podem responder ao usuário de forma desejada ou não;

3. Selecionar as métricas: podem ser relacionadas à velocidade, precisão e disponibilidade do sistema. Métricas são os critérios utilizados para avaliar o desempenho de um sitema. As métricas são selecionadas de acordo com a lista de variáveis de resposta resultantes da execução de alguma rotina no sistema. Se a rotina do sistema executa os serviços de forma correta, o desempenho é medido através do tempo de execução, da taxa na qual os serviços foram executados e da utilização dos recursos durante essa execução. Essas medidas fornecem informações sobre a responsividade, a produtividade e a utilização,

respectivamente (JAIN, 1991). A avaliação das métricas ajuda na tomada de decisão por parte de desenvolvedores e arquitetos de software com o objetivo de obter a maior eficiência do sistema ao menor custo. Por exemplo, ao avaliar a utilização de algum recurso é possível escolher a opção que demanda menos para realizar uma determinada ação ou rotina;

4. Listar os parâmetros: parâmetros afetam o desempenho do sistema e são divididos naqueles que variam e naqueles que permanecem constantes durante a avaliação;

5. Selecionar os fatores: são os parâmetros que variam e seus valores são chamados de níveis;

6. Selecionar a técnica: existem três abordagens de avaliação: modelagem analítica, simulação e aferição em protótipo. A escolha da técnica depende do tempo e dos recursos disponíveis, assim como a precisão desejada para os resultados. A principal consideração para definir a técnica utilizada é a etapa do ciclo de vida na qual o sistema a ser avaliado se encontra. Caso o sistema, ou um protótipo dele, já exista, a aferição é a mais indicada (JAIN, 1991). Esta técnica baseia-se na coleta de dados e medidas do próprio sistema. A modelagem analítica e simulação consistem em utilizar um modelo do sistema e podem ser realizadas em qualquer fase do projeto. A modelagem analítica consiste na obtenção da solução para um problema matemático por meio de algoritmos. Já a simulação é feita através da execução do modelo, produzindo amostras e medidas de desempenho do sistema.

7. Selecionar a carga de trabalho: deve ser uma representação da utilização do sistema na vida real. A carga de trabalho é a quantidade de requisições feitas pelos usuários, incluindo todas as ações necessárias para sua execução pelo sistema em avaliação. Além disso, a carga de trabalho é a mesma para todas as execuções em um mesmo experimento. Um gerador de carga (*load generator*) pode ser utilizado como ferramenta para gerar a carga de trabalho;

8. Projetar os experimentos: consiste em definir a sequência de experimentos que oferece o máximo de informação com o mínimo de execuções;

9. Analisar e interpretar os dados: o resultado de cada experimento é diferente a cada repetição. Isto é chamado de variabilidade de resultados;

10. Apresentar os resultados: consiste em comunicar os resultados de forma que sejam facilmente entendidos e, preferencialmente, de forma gráfica.

2.2.1 Especificação de *Benchmark*

Para guiar o desenvolvimento da aplicação *benchmark*, foi considerada a especificação TPC-W disponibilizada pelo *Transactional Processing Performance Council*, uma corporação

sem fins lucrativos fundada com o objetivo de definir *benchmarks*, assim como o processo para revisão e monitoramento desses *benchmarks* (COUNCIL, 2021). A norma TPC-W (COUNCIL, 2003) especifica uma carga de trabalho de *e-commerce*, ou comércio eletrônico, que simula atividades de uma loja de varejo. Usuários simulados podem navegar e comprar produtos na aplicação desenvolvida utilizando o TPC-W, e também é possível acessá-la através de um navegador (*browser*) real.

Em um *e-commerce*, clientes interagem com um site através de sessões, as quais são sequências de requisições que executam funções (como pesquisar, adicionar ao carrinho, pagar, entre outras) durante uma visita única. Uma das formas de capturar o padrão de navegação dentro de uma sessão é através de um grafo de comportamento do cliente (*Customer Behavioral Model Graph* (CBMG)) que descreve quais ações são executadas e sua frequência. É possível detectar diversos padrões de comportamento, como visitas de compradores ocasionais, que podem passar mais tempo navegando pelos produtos, e visitas de compradores frequentes, que em poucos cliques realizam a compra de diversos itens. Diferentes comportamentos de usuário geram diferentes cargas de trabalho sobre os recursos do sistema (MENASCé, 2002).

A especificação TPC-W descreve em detalhes 14 páginas web diferentes. A navegação entre as páginas ocorre através de uma distribuição chamada de mix de interações web (*Web Interaction Mix*), na qual existe um conjunto de opções disponíveis a partir de cada página web. As páginas web representam as seguintes funcionalidades específicas de um negócio de venda de livros, particularidade da TPC-W (COUNCIL, 2003):

- Acessar página inicial (*home*): fornece uma lista dos livros lançados recentemente e uma lista dos livros com maior número de vendas;
- Adicionar ao carrinho: além de fornecer os detalhes dos itens já adicionados no carrinho e suas quantidades, permite adicionar um ou mais livros ao carrinho de compras;
- Registrar cliente: permite que o cliente se conecte com suas informações de usuário (nome de usuário e senha) ou envie as informações necessárias para ser cadastrado como um novo usuário;
- Requisitar compra: o cliente demonstra sua intenção de compra e a aplicação fornece seus dados de cadastro, assim como as informações detalhadas dos itens de seu carrinho. Também são disponibilizadas opções de método de pagamento e envio.
- Confirmar compra: a compra é realizada e suas informações são fornecidas ao cliente. Uma simulação de pagamento é feita e o carrinho é esvaziado;
- Consultar pedidos: é o primeiro passo para a visualização da última compra realizada pelo cliente, solicitando ao usuário a confirmação de seus dados cadastrais;

- Visualizar a última compra: fornece ao usuário todas as informações referentes à última compra realizada;
- Requisitar pesquisa: fornece critérios de busca de livros (autor, gênero literário e título) e suas opções;
- Pesquisar livros: permite especificar os critérios de busca e fornece uma lista dos livros disponíveis;
- Listar lançamentos de livros: fornece uma lista dos livros lançados recentemente;
- Listar livros mais vendidos: fornece uma lista dos livros mais vendidos;
- Exibir detalhes do livro: fornece as informações detalhadas de determinado livro, como título, autor, disponibilidade, preço, entre outros;
- Requisitar alteração em item: fornece informações sobre determinado livro para que seu preço seja editado em seguida. Esta ação é descrita como funcionalidade de administrador; Algumas funcionalidades, descritas na Seção 2.2.1, foram desenvolvidas de forma diferente do especificado na norma TPC-W. O serviço de comprar livros foi desenvolvido para integrar as funcionalidades de requisitar compra e confirmar compra. Visualizar detalhes da compra compreende as funcionalidades consultar pedidos e visualizar a última compra.

As funcionalidades são divididas em duas categorias: navegação (*browse*) e compra (*order*). As funcionalidades de navegação consistem de requisições com pouco processamento, enquanto funcionalidades de compra requerem maior processamento, principalmente em termos de acesso ao banco de dados. As funcionalidades da categoria de navegação são: acessar página inicial, visualizar lançamentos, visualizar mais vendidos, exibir detalhes do item, requisitar busca e buscar itens. As funcionalidades de compra são: adicionar ao carrinho, registrar cliente, requisitar compra, confirmar compra, consultar pedidos, visualizar a última compra, requisitar alteração em item e confirmar alteração em item. A TPC-W define três distribuições diferentes de tipos de interações web com base na relação de atividades de navegação e compra: *shopping mix* (80% de navegação e 20% de compra), *browsing mix* (95% de navegação e 5% de compra) e *ordering mix* (50% de navegação e 50% de compra).

A estrutura de banco de dados definida pela TPC-W consiste de um mínimo de oito tabelas para armazenar dados de cliente, endereço, país, pedidos, itens, relação de pedidos e itens, transações por cartão de crédito e autores. Cada uma dessas tabelas é definida detalhadamente na especificação em termos de colunas, tipos de dados e registros armazenados. A TPC-W determina que todas as interações com o banco de dados devem ser feitas através de transações com suporte completo das propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) (COUNCIL, 2003).

Existem algumas limitações relacionadas ao uso da especificação TPC-W. A norma foi desenvolvida com foco na relação de consumidor e negócio, ou seja, não modela transações entre negócios (*business-to-business*). A especificação também ignora a possível influência de robôs na carga de trabalho. Além disso, os resultados são obtidos em um ambiente controlado, portanto não considera usuários com conexão de baixa velocidade (MENASCé, 2002).

2.3 TRABALHOS RELACIONADOS

Diversos trabalhos abordam análise de desempenho envolvendo serviços web, empregando diferentes ferramentas e estratégias de comparação.

Amza et al. (2002) busca avaliar o desempenho de páginas web com conteúdo dinâmico empregando três *benchmarks*, uma livraria online (*online bookstore*, a exemplo Inc. (2021a)), uma página de leilão (*auction site*, a exemplo Amazon.com (2021a)), e um quadro de avisos (*bulletin board*). Diferentes cargas de trabalho são avaliadas (*workload mix*): *browsing mix*, *shopping mix* e *ordering mix*. A diferença entre essas cargas de trabalho é a proporção de requisições de apenas leitura (*read-only*) e leitura-escrita (*read-write*). As métricas analisadas são utilização de CPU, memória, disco, rede e tempo de resposta.

Muito semelhante ao trabalho anterior, Cecchet et al. (2003) utiliza os mesmos modelos de negócio para implementar os *benchmarks* com diferentes arquiteturas. PHP, Java Servlets e EJB (*Enterprise Java Beans*) representam três arquiteturas para geração de conteúdo dinâmico e seus desempenhos são comparados. Foi medido e analisado o comportamento dos *benchmarks* para cada carga de trabalho com número crescente de usuários, até atingir o pico de utilização de CPU, para identificar limitações e gargalos.

??) aborda de forma mais detalhada e prática a especificação TPC-W, especificação de *benchmark* de *e-commerce* para serviço web e banco de dados proposto pelo *Transactional Processing Performance Council* e sua aplicação em uma análise de desempenho. Neste trabalho, é definido o comportamento do usuário de *e-commerce* e caracterizadas as diferentes cargas de trabalhos (*browsing, shopping e ordering mix*). Além disso, é apresentado o conceito de *think time*, o tempo entre cada interaçao do usuário dentro da rotina do experimento.

No trabalho de Voigt e Junior (2017) é realizada uma análise de desempenho entre SOAP e REST, com o objetivo de comparar o desvio padrão entre as amostras, o tempo de resposta, o volume de transferência de dados por segundo e o tamanho de pacote por requisição. Foram aplicadas apenas três cargas de trabalho: 1, 10 e 100 usuários simultâneos. Não foi empregado nenhum *benchmark*. Foi implementado apenas um método de consulta para cada serviço web. O experimento foi realizado em dois ambientes: em um servidor local e em um servidor na nuvem. Como resultado, o REST se mostrou mais eficiente que o SOAP nas métricas tempo médio, máximo e mínimo de resposta, desvio padrão entre as amostras e tamanho de pacotes para todas as cargas de usuários em ambos os ambientes.

Moro, Dorneles e Rebonatto (2011) apresentam um estudo comparativo entre os serviços web SOAP e REST. Além de explicar diversos conceitos envolvendo as duas *middlewares*, foram desenvolvidos dois *benchmarks* com quatro métodos de consulta. O gerenciamento de estado da aplicação, com armazenamento (*stateful*) e sem armazenamento (*stateless*) do estado, e o formato de dados das requisições eram diferentes entre as versões SOAP e REST. O formato XML no retorno das chamadas foi utilizado nas duas *middlewares*. Como métrica foi analisado o tamanho dos pacotes de dados (em *bytes*) transportados em cada requisição. Para os quatro métodos de consulta, o REST foi mais vantajoso, pois o número de *bytes* transportados foi menor em todos os cenários.

No estudo de caso de Claro1 et al. (2016), é feita uma análise de desempenho entre servicos web SOAP e REST em dispositivos de internet das coisas (IoT - *Internet of Things*). Foi desenvolvida uma aplicação e esta foi executada em um computador e em uma placa *Raspberry Pi*. Como métrica foi analisado o tempo de resposta das requisições. Os resultados deste estudo demonstraram que ambos os serviços não têm diferenças significativas em relação ao tempo de resposta, porém a quantidade de dados trafegados com SOAP é muito maior, sendo o REST mais eficiente nessa métrica.

O presente trabalho propõe realizar uma análise de desempenho entre os serviços web SOAP e REST, utilizando *benchmarks* de *e-commerce* baseados na especificação TPC-W a fim de analisar o comportamento de CPU, memória RAM, disco e rede com diferentes cargas de usuário.

No próximo capítulo são explicadas as tecnologias empregadas para o desenvolvimento dos *benchmarks*, emulador de clientes e execução e monitoramento do sistema.

3 PROJETO DE SOFTWARE

Este capítulo apresenta o projeto dos serviços web, incluindo os diagramas de casos de uso e diagrama de classes, e descreve como é realizado o acesso dos clientes a esses serviços web através de um roteiro das requisições.

3.1 SERVIÇOS WEB

Foram desenvolvidas duas aplicações *benchmark* com o objetivo de representar os serviços web de *e-commerce* reais. As duas aplicações diferem apenas na implementação das *middlewares* para serviços web SOAP e REST. A especificação TPC-W, apresentada na Seção 2.2.1, foi utilizada para definir os serviços disponibilizados em ambas as aplicações.

Para o desenvolvimento dos *benchmarks* foi utilizada a linguagem de programação Java na versão 11. No caso do *benchmark* que emprega a *middleware* SOAP foi utilizado o JAX-WS[1] (*Java API for XML Web Services*) versão 2.0.3. Para o *benchmark* do serviço web REST foi utilizada o JAX-RS[2] (*Java API for RESTful Web Services*). Ambas as *middlewares* estão integradas na plataforma Java EE e disponibilizam anotações (*annotations*) para implementar as diversas funções dos serviços web de acordo com cada abordagem. Para tornar a comparação mais justa, o XML foi utilizado como formato de dados de mensagens no SOAP e no REST.

Como servidor de aplicações foi utilizado o WildFly[3] 18, um software de código aberto que pode ser usado em qualquer sistema operacional que tenha suporte à linguagem Java. Por consequência, foi utilizado o Hibernate[4] como *framework* para a persistência de dados. O Hibernate implementa o JPA (*Java Persistence API*) que é a especificação empregada para tratamento e persistência de dados e classes. Neste projeto foi utilizado o JPA versão 2.1.

Todos os dados manipulados nas aplicações são armazenados em um banco de dados. Foi utilizado o MySQL[5] versão 8.0.22 como sistema de gerenciamento do banco de dados.

Também foi empregada a ferramenta Apache Maven[6] versão 3.6.3 para gerenciamento de dependências. As aplicações foram desenvolvidas utilizando a IDE Eclipse 2019-09.

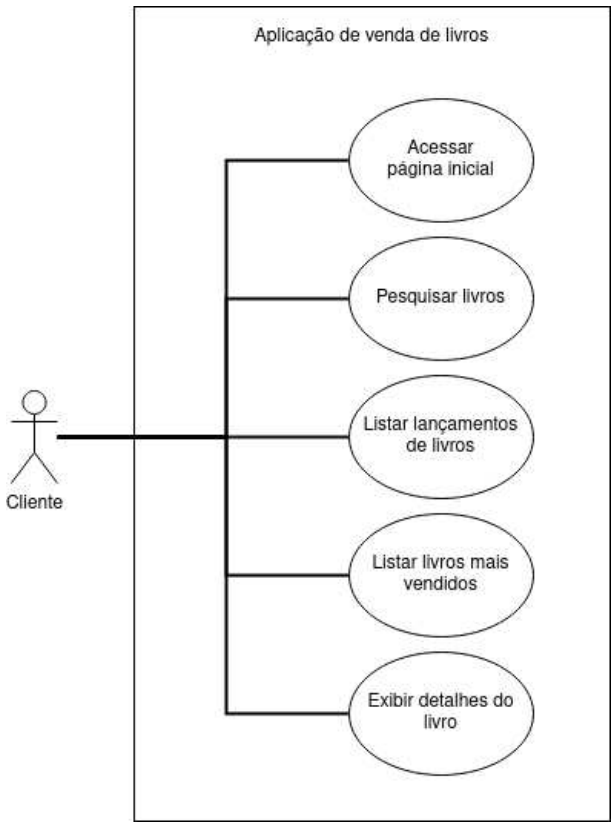
3.1.1 Diagramas de Casos de Uso

A partir da especificação TPC-W, foi possível descrever os casos de uso da aplicação. Considerando que esta é uma aplicação para venda de livros, é possível determinar dois tipos

1 <https://javaee.github.io/metro-jax-ws/>
2 <https://javaee.github.io/jsr311/>
3 <https://www.wildfly.org/>
4 <https://hibernate.org/>
5 <https://www.mysql.com/>
6 <https://maven.apache.org/>

de interações com a plataforma, navegação e compra, como mencionado anteriormente na Seção 2.2.1.

Figura 2: Diagrama de casos de uso das interações de navegação.

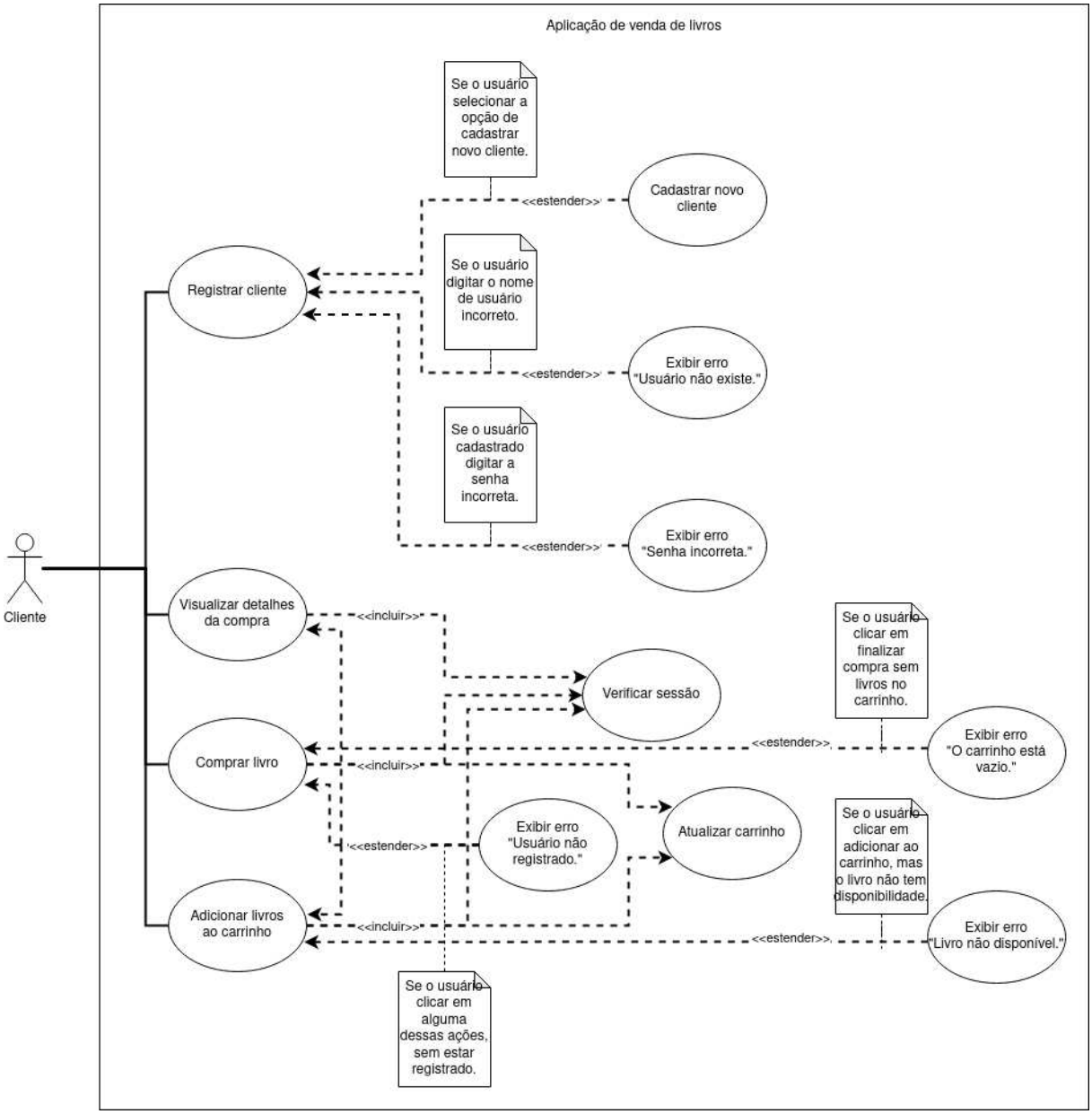


Fonte: Autoria própria

A Figura 2 representa o diagrama de casos de uso das interações de navegação da aplicação. Os serviços de navegação consistem de consultas de livros na plataforma, sendo métodos de leitura (*read*) ao banco de dados. Esses métodos podem utilizar parâmetros fornecidos pelo cliente como filtro de busca. As cinco ações de navegação são: a) acessar página inicial; b) pesquisar livros; c) listar lançamentos de livros; d) listar livros mais vendidos; e f) exibir detalhes do livro. Todas as interações podem ser realizadas pelo cliente de forma independente. Além disso, as ações de navegação podem ser realizadas de forma anônima, ou seja, sem a necessidade de identificação do cliente.

A Figura 3 representa o diagrama de casos de uso das interações de compra do sistema. Os serviços de compra consistem da atualização de registros na plataforma, sendo métodos de leitura e escrita (*read-write*) no banco de dados. As quatro ações de compra são: a) registrar cliente; b) visualizar detalhes da compra; c) comprar livro; e d) adicionar livros ao carrinho. Essas ações possuem uma ordenação determinada. Para comprar um livro é preciso adicioná-lo ao carrinho primeiro e para visualizar os detalhes da compra é preciso ter comprado um livro ao menos uma vez. A ação de registrar um cliente permite três ações subsequentes: cadastrar um novo cliente, caso seja selecionada essa opção; e exibir o erro "Usuário não existe"ou "Senha

Figura 3: Diagrama de casos de uso das interações de compra.



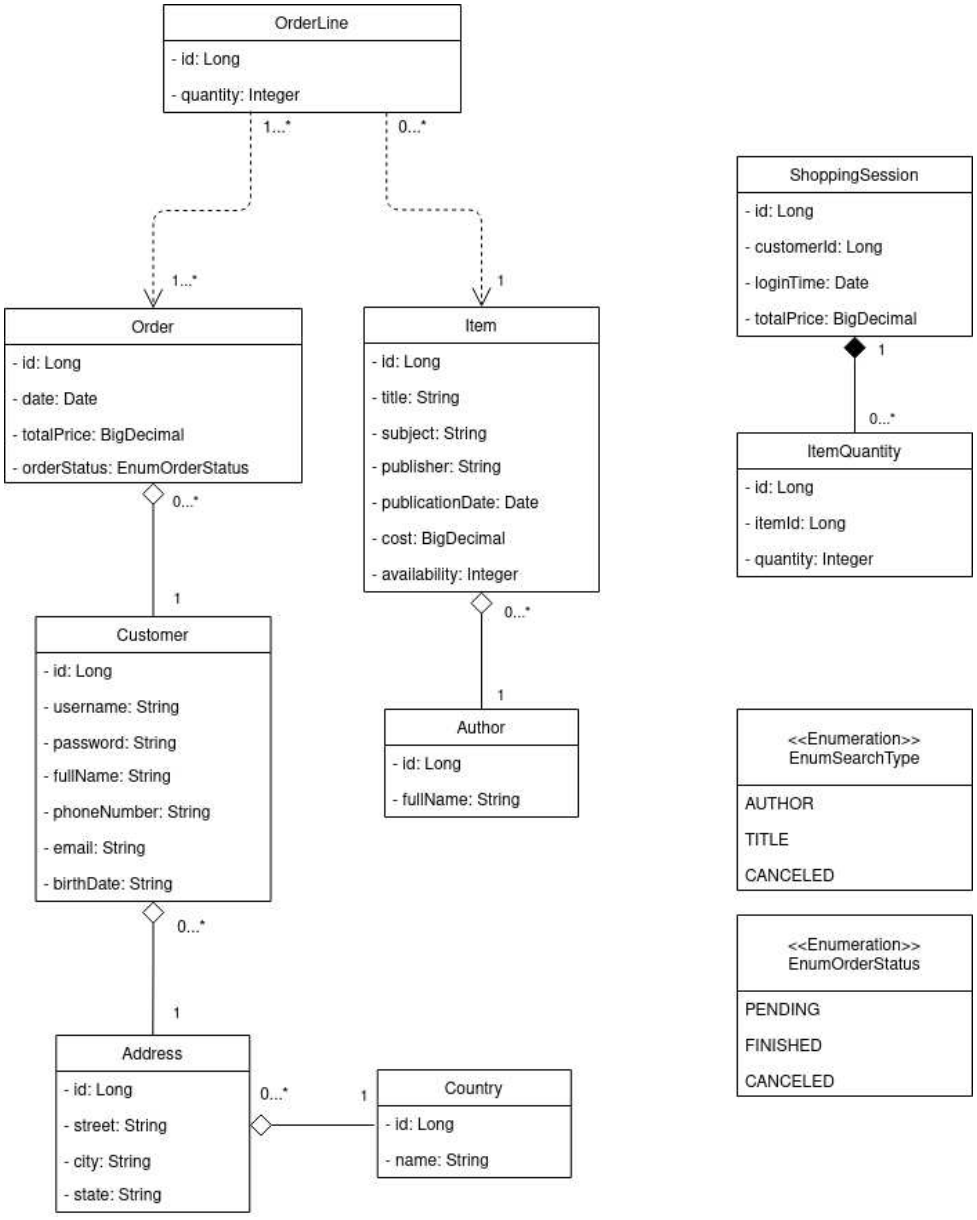
Fonte: Autoria própria

incorreta", caso sejam digitadas as credenciais incorretas para um cliente existente. As demais ações b), c) e d) sempre realizam a ação de verificar a sessão, caracterizada pela identificação do cliente dentro do sistema. Nesses casos, o registro do cliente é obrigatório e ocorre através da ação a). Caso esta ação não tenha sido realizada, é exibido o erro "Usuário não registrado". As ações c) e d) sempre realizam a ação de atualizar o carrinho de compras, podendo ocorrer através da adição ou remoção de livros. Além disso, caso seja realizada a ação c) e não haja livros adicionados no carrinho de compras ocorre a exibição do erro "O carrinho está vazio". Ao realizar a ação d) quando um livro não possui disponibilidade, será exibido o erro "Livro não disponível".

3.1.2 Diagrama de Classes

Para realizar a implementação das aplicações é preciso definir as entidades envolvidas no projeto e como essas classes se relacionam entre si.

Figura 4: Diagrama de classes.



Fonte: Autoria própria

A Figura 4 corresponde ao diagrama de classes dos *benchmarks*. As aplicações são compostas por nove classes: *Customer*, *Address*, *Country*, *Item*, *Author*, *OrderLine*, *ShoppingSession* e *ItemQuantity*. Foram definidas também as enumerações: *EnumSearchType* e *EnumOrderStatus*. O diagrama define a relação entre as classes, incluindo a cardinalidade e multiplicidade, assim como os atributos que compõem cada uma delas.

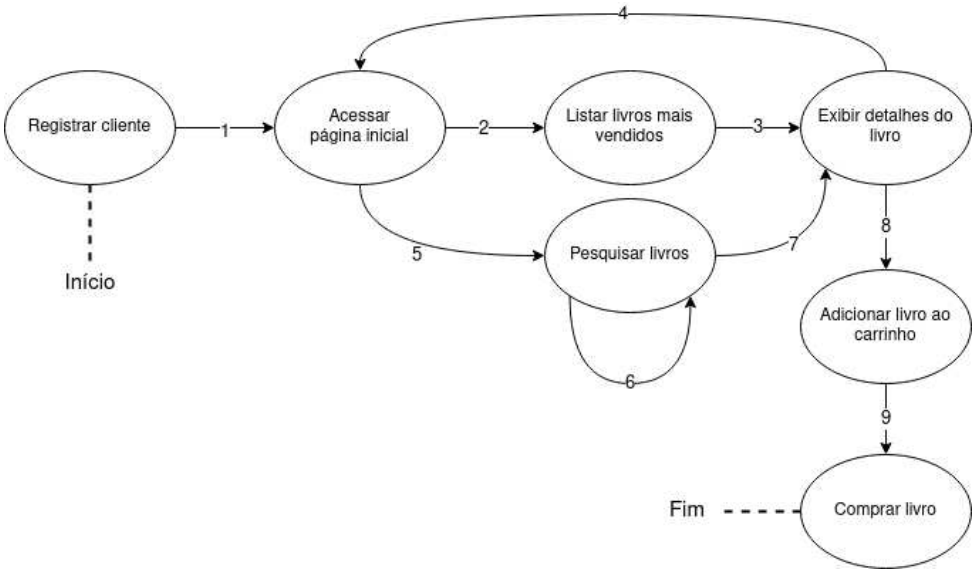
3.2 CLIENTES

Para simular o acesso simultâneo de vários usuários aos serviços web foi necessário utilizar um emulador de clientes. O SoapUI [7] é uma ferramenta de testes de código aberto que possui diversos recursos para realizar experimentos envolvendo requisições SOAP e REST. O software permite criar um roteiro de testes e possibilita a personalização de diversos fatores como número de *threads* simultâneas, tipo de estratégia e tempo entre requisições. A versão do SoapUI utilizada foi a 5.6.0.

3.2.1 Roteiro de requisições

As requisições realizadas pelos clientes são definidas de acordo com a distribuição *shopping mix*, descrita na Seção 2.2.1. A ordem e o intervalo de tempo das ações foram decididos experimentalmente, com o objetivo de simular um acesso próximo da realidade de um cliente de *e-commerce*.

Figura 5: Diagrama de fluxo das interações.



Fonte: Autoria própria

A Figura 5 representa o fluxo das interações realizadas por um cliente. A interação inicia com o registro do cliente, no qual é feito um novo cadastro na plataforma. Em seguida, são realizadas as ações de acessar página inicial, listar livros mais vendidos e depois exibir detalhes de determinado livro. A ação de acessar página inicial é executada novamente, desta vez seguida da ação de pesquisar livros. Esta ação de pesquisar livros é realizada mais uma vez pelo cliente para, por exemplo, pesquisar outro título. Depois, são realizadas as ações de exibir os detalhes do livro e adicionar o livro ao carrinho. Por fim, é feita a compra do livro.

[7] <https://www.soapui.org>

O fluxo contém 7 interações diferentes constituindo 10 passos de execução. O percentual de interações de navegação e compra é de 70% e 30%, respectivamente. O fluxo é realizado apenas uma vez por cliente e o intervalo de tempo entre cada interação é de 1 segundo.

3.2.2 Tipo de estratégia

O tipo de estratégia define como será o comportamento de um número de *threads* ao longo do tempo do experimento. Cada cliente é representado por uma *thread*. O SoapUI disponibiliza quatro tipos de estratégias (SMARTBEAR, 2021):

- *Simple*: executa um número constante de *threads* com um intervalo de tempo entre si. Esta estratégia permite definir o valor deste intervalo de tempo;
- *Burst*: executa um número constante de *threads* por um período determinado e pára completamente sua execução durante o intervalo entre as execuções. Esta estratégia é indicada para testar a capacidade de recuperação do sistema;
- *Thread*: executa um número de *threads* com crescimento linear, permitindo definir seu valor inicial e final;
- *Variance*: executa um número de *threads* com uma variação na forma de "dente de serra", ou seja, intercalando intervalos de crescimento linear e queda brusca no número de *threads*.

Foi escolhido o tipo de estratégia *Simple*, pois não é necessário um cenário de execução complexo para os experimentos desta análise. Esta estratégia é a mais simples e representa bem o acesso de múltiplos clientes a uma plataforma de *e-commerce*.

O próximo capítulo apresenta as características do experimento utilizado para analisar o desempenho das *middlewares* SOAP e REST em uma aplicação *benchmark* de *e-commerce*.

4 EXPERIMENTO

Este capítulo descreve a realização do experimento que inclui as etapas de preparação, execução e avaliação.

4.1 PREPARAÇÃO

Duas máquinas distintas foram usadas para representar os clientes e o servidor.

A aplicação *benchmark* foi executada dentro de uma máquina virtual hospedada com o Oracle VM Virtualbox[1] versão 6.1. O sistema operacional escolhido foi o Linux Ubuntu 18.04, sem interface gráfica. A máquina foi configurada com 4GB de memória RAM, 6,3GB de capacidade de armazenamento em disco e apenas um núcleo de processamento (CPU). A Tabela 1 resume as configurações adotadas para a máquina servidor.

Tabela 1: Configurações de ambiente para a máquina servidor.

Parâmetro	Configuração
Sistema Operacional	Linux Ubuntu 18.04
Uso de interface gráfica	Não
Memória RAM	4 GB
Memória em disco	6,3 GB
Número de núcleos (CPU)	1
Capacidade da interface de rede	1000 Mbps

Fonte: Autoria própria

Os clientes foram executados utilizando a máquina local. Não é necessário ter controle deste ambiente, já que a única função de cada cliente é realizar as requisições aos serviços web através da rede.

4.2 EXECUÇÃO

Foram realizados dois experimentos separados: um para a aplicação *benchmark* com a *middleware* SOAP e outro utilizando a *middleware* REST. Durante cada experimento, foi executada a rotina de requisições dos clientes descrita na Seção 3.2.1 com diferentes números de *threads* a cada execução. O tipo de estratégia utilizada para definir o comportamento das *threads* foi a *simple*, descrita na Seção 3.2.2. Os números de *threads* utilizados foram 20, 40, 60, 80, 100 e 120. O intervalo de tempo entre as *threads* foi definido como zero, resultando na execução simultânea de todas as *threads*. Cada uma das execuções foi repetida 10 vezes.

[1] <https://www.virtualbox.org/>

Para proporcionar a execução da rotina de interações de forma equivalente foi necessário garantir que o ambiente de software fosse o mesmo a cada repetição. Os seguintes passos foram seguidos para a realização do experimento:

1. Inicialização da máquina virtual;
2. Restauração do banco de dados;
3. Inicialização do servidor da aplicação;
4. Espera de 1 minuto;
5. Inicialização da ferramenta de aferição das métricas do sistema;
6. Execução da rotina de requisições pelos clientes;
7. Finalização da aferição das métricas;
8. Término do servidor;
9. Término da máquina virtual.

Este procedimento foi definido de forma experimental. A reinicialização da máquina virtual minimiza a influência das execuções anteriores sobre a execução atual. A restauração do banco de dados garante que o estado dos dados será o mesmo a cada execução. O tempo de espera de 1 minuto foi definido empiricamente para analisar a utilização da CPU até atingir um percentual constante.

4.3 AVALIAÇÃO

Para analisar o desempenho das *middlewares* foi utilizada a abordagem sistemática apresentada na Seção 2.2. A técnica de avaliação utilizada foi a aferição em protótipo.

As métricas selecionadas para este trabalho foram o percentual de utilização de CPU, memória RAM, disco e rede e o tempo de execução (em segundos).

Para o monitoramento das métricas foi utilizado o sysstat[2], uma coleção de ferramentas de monitoramento de desempenho de sistemas Unix. Atráves dessas ferramentas foi possível medir a utilização de CPU, memória RAM, disco e interface de rede com controle da frequência e tempo de aferição. O valores aferidos são salvos como arquivo de texto, separados por métrica analisada. O software também possui seu código aberto e disponível no GitHub[3]. A frequência de aferição definida para todas as execuções foi de 1 segundo e o tempo de aferição foi variado de acordo com o tempo total da execução, obtido experimentalmente a cada rodada.

[2] <http://sebastien.godard.pagesperso-orange.fr/documentation.html>
[3] <https://github.com/sysstat/sysstat>

Além da aferição das métricas, foi necessário desenvolver um programa para tratamento dos dados obtidos com o sysstat e para geração dos gráficos. Esse programa foi desenvolvido na linguagem Python 2.7.18 em conjunto com o editor Visual Studio Code 1.47.3 da Microsoft. Foram utilizadas as bibliotecas matplotlib[4], numpy[5] e scipy[6] para plotagem de gráficos, manipulação de vetores e cálculos estatísticos.

[4] <https://matplotlib.org/>
[5] <https://numpy.org/>
[6] <https://www.scipy.org/>

5 RESULTADOS

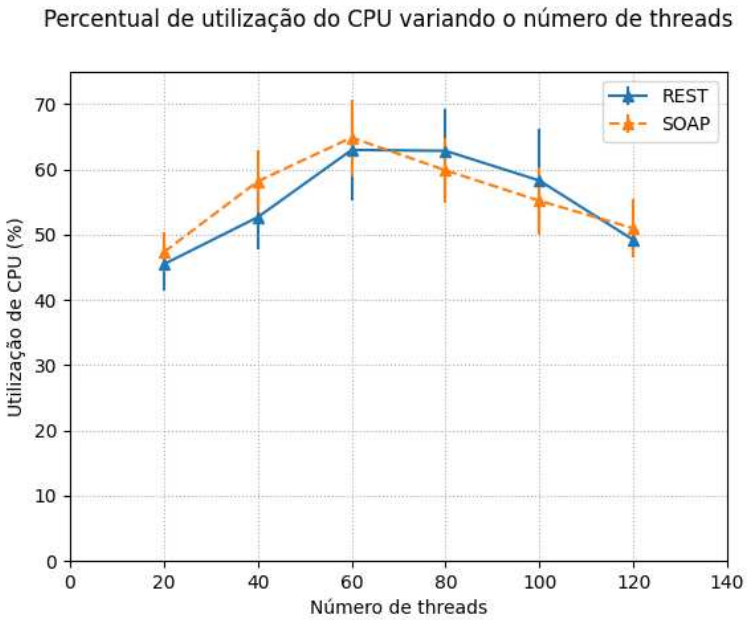
Este capítulo mostra os resultados obtidos nos experimentos envolvendo as aplicações *e-commerce* utilizando as *middlewares* REST e SOAP. Cada gráfico apresenta a média dos resultados obtidos através de 10 execuções para uma determinada métrica do sistema, com diferentes quantidades de clientes realizando requisições simultâneas.

Para avaliar se as diferenças de desempenho entre as *middlewares* (grupos em um contexto estatístico) são estatisticamente significativas considerando as métricas analisadas (fator em um contexto estatístico), é calculado o valor-p (*p-value*) dos resultados. Um $valor - p > 0,05$ indica (com 95% de confiança) que não há diferença significativa entre os grupos e um $valor - p \leq 0,05$, se houver diferença estatisticamente significativa.

5.1 CPU

É relevante analisar o percentual de CPU utilizado para entender qual *middleware* demanda menos processamento para realizar a mesma ação. Quanto menor a utilização de recursos, maior é a eficiência da *middleware*. A Figura 6 mostra o comportamento das aplicações quanto à utilização de CPU. A Tabela 2 contém os valores referentes ao gráfico.

Figura 6: Percentual de utilização de CPU para REST e SOAP variando o número de *threads*.



Fonte: Autoria própria

É possível observar que a utilização de CPU varia de forma crescente até a máxima de aproximadamente 63%, com 60 *threads*, e decresce depois deste ponto até aproximadamente

Tabela 2: Percentual de utilização de CPU para REST e SOAP de acordo com o número de *threads*.

Número de *threads*	Valor aferido para o REST (%)	Valor aferido para o SOAP (%)
20	45,456 ±4,01	47,31 ±3,023
40	52,662 ±5,017	58,147 ±4,751
60	62,964 ±7,633	64,857 ±5,891
80	62,825 ±6,311	59,841 ±4,948
100	58,266 ±7,923	55,172 ±5,164
120	49,191 ±2,611	50,971 ±4,451

Fonte: Autoria própria

50%. Ao realizar o cálculo do valor-p, foi determinado que para os números de *threads* 20, 60, 80, 100 e 120, o desempenho das *middlewares* em relação ao uso de CPU não é significativamente diferente visto que seus intervalos se sobrepõem. Na execução com 40 *threads*, o desempenho é considerado estatisticamente diferente e a *middleware* SOAP apresenta utilização de CPU aproxidamente 10,4% maior que o REST.

5.2 MEMÓRIA RAM

A análise de utilização de memória RAM indica qual *middleware* necessita de menos memória temporária para armazenamento. Portanto, uma solução com menor demanda é preferível. A Figura 7 mostra o comportamento das *middlewares* quanto à utilização de memória RAM. A Tabela 3 contém os valores referentes ao gráfico.

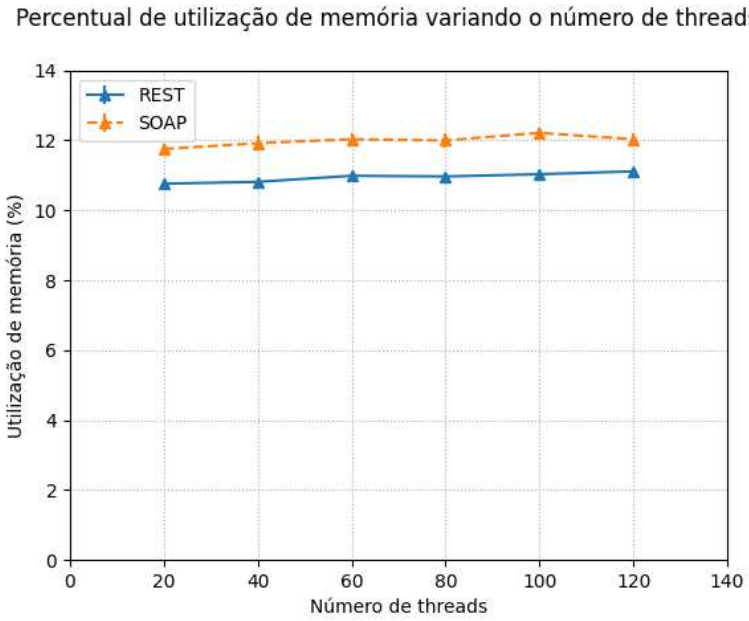
Tabela 3: Percentual de utilização de memória RAM para REST e SOAP de acordo com o número de *threads*.

Número de *threads*	Valor aferido para REST (%)	Valor aferido para SOAP (%)
20	10,758 ±0,103	11,752 ±0,136
40	10,812 ±0,107	11,916 ±0,197
60	10,985 ±0,117	12,03 ±0,168
80	10,967 ±0,091	11,998 ±0,221
100	11,029 ±0,11	12,212 ±0,139
120	11,111 ±0,109	12,03 ±0,188

Fonte: Autoria própria

Pode-se observar que a utilização de memória RAM é independente do número de *threads* para ambas as *middlewares* e que os resultados apresentados são significativamente diferentes. A aplicação com a *middleware* REST possui uma média de utilização de memória RAM de 10,94%, enquanto que no SOAP essa média é de 11,99%. A diferença percentual das duas *middlewares* foi de aproximadamente 9,56%.

Figura 7: Percentual de utilização de memória RAM para REST e SOAP variando o número de *threads*.



Fonte: Autoria própria

5.3 DISCO

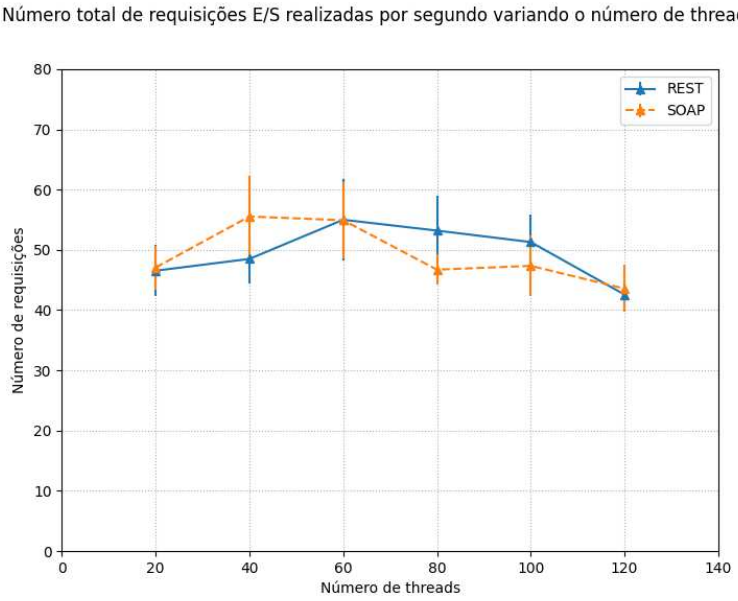
A análise de utilização de disco permite verificar o armazenamento de dados de forma não-volátil, e suas operações de leitura e escrita ajudam a escolher qual *middleware* exige menos capacidade de armazenamento e é mais eficiente para o processamento de operações em disco. A unidade de medida escolhida para avaliar essa métrica foi o número total de transferências por segundo. Segundo a documentação da ferramenta *sysstat* (GODARD, 2021), uma transferência é uma requisição E/S para um dispositivo físico. Múltiplas requisições lógicas podem ser combinadas em uma única requisição de E/S. Além disso, uma transferência não possui tamanho determinado. A Figura 8 mostra o comportamento das *middlewares* quanto à utilização de memória RAM. A Tabela 4 contém os valores referentes ao gráfico.

Tabela 4: Número de requisições E/S realizadas por segundo para REST e SOAP de acordo com o número de *threads*.

Número de *threads*	Valor aferido para REST	Valor aferido para SOAP
20	46,542 ±4,25	47,071 ±3,677
40	48,5 ±4,024	55,54 ±6,668
60	54,996 ±6,769	54,917 ±6,263
80	53,204 ±5,807	46,72 ±2,477
100	51,295 ±4,606	47,348 ±4,967
120	42,58 ±2,178	43,572 ±3,89

Fonte: Autoria própria

Figura 8: Número de requisições E/S realizadas por segundo para REST e SOAP variando o número de *threads*.



Fonte: Autoria própria

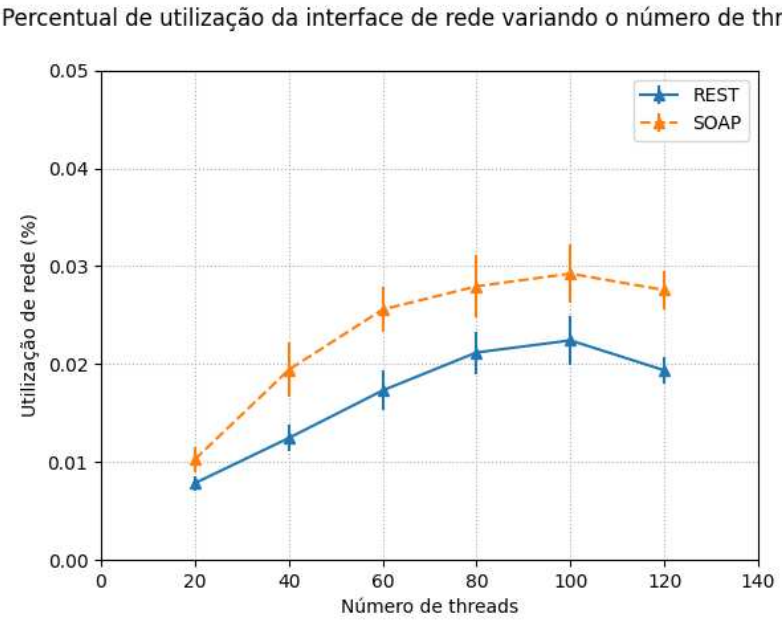
Através da análise do gráfico, da tabela e do valor-p, pode-se considerar que os resultados quanto à utilização de disco não são estatisticamente diferentes nos pontos de 20, 60 e 120 *threads* para as duas *middlewares*. Já nos pontos de 40, 80 e 100 *threads* os resultados são estatisticamente diferentes. Com 40 *threads*, o SOAP utilizou mais disco. Já com um número maior de *threads* (80 e 100), o REST utilizou mais disco do que o SOAP. De forma geral, os comportamentos não apresentam grandes variações com o aumento do número de *threads*. O número total de requisições de E/S realizadas por segundo durante os experimentos oscilou entre 46,54 e 55,54.

5.4 REDE

O percentual de utilização da rede é outra característica relevante para determinar qual *middleware* demanda mais largura de banda para transportar os dados. O cálculo considera o volume total de dados (em kilobytes) recebidos ou transmitidos por segundo em relação a capacidade total de transmissão. Neste caso, a capacidade da interface de rede é de 1000 Mbps. A Figura 9 mostra o comportamento das *middlewares* quanto à utilização de memória RAM. A Tabela 5 contém os valores referentes ao gráfico.

Pode-se verificar que a aplicação que emprega SOAP tem um percentual de utilização de rede maior do que o REST para um número de *threads* entre 20 e 120. Isto pode ser explicado pela forma de encapsular os dados enviados em requisições. O SOAP requer a utilização de envelopes XML formatados de acordo com a especificação WSDL, enquanto o

Figura 9: Percentual de utilização da interface de rede para REST e SOAP variando o número de *threads*.



Fonte: Autoria própria

Tabela 5: Percentual de utilização da interface de rede para REST e SOAP de acordo com o número de *threads*.

Número de *threads*	Valor aferido para REST (%)	Valor aferido para SOAP (%)
20	0,008 ±0,001	0,01 ±0,001
40	0,012 ±0,001	0,019 ±0,003
60	0,017 ±0,002	0,026 ±0,002
80	0,021 ±0,002	0,028 ±0,003
100	0,022 ±0,003	0,029 ±0,003
120	0,019 ±0,001	0,028 ±0,002

Fonte: Autoria própria

REST não requer nenhum tipo de padronização além do formato empregado. A mensagem transportada com SOAP é mais extensa e, consequentemente, requer um pacote maior de dados. A média do percentual de ganho do REST em relação ao SOAP foi de aproximadamente 41,5%.

5.5 TEMPO

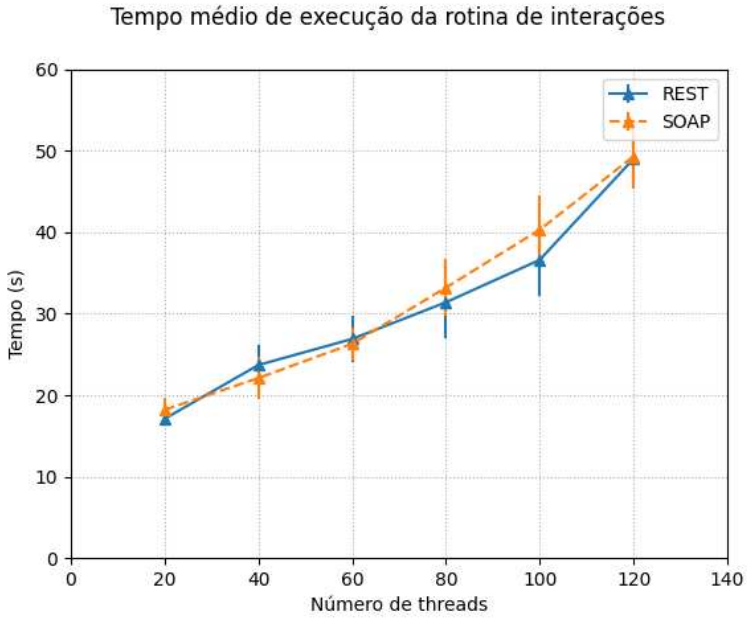
O tempo de execução do experimento também foi considerado, pois é relevante saber qual *middleware* executa as mesmas ações em menos tempo.

Tabela 6: Tempo total de execução da rotina de requisições, em segundos, para REST e SOAP de acordo com o número de *threads*.

Número de *threads*	Valor aferido para REST (s)	Valor aferido para SOAP (s)
20	17,1 ±0,7	18,2 ±1,4
40	23,7 ±2,41	22,1 ±2,663
60	26,9 ±2,844	26,3 ±2,002
80	31,4 ±4,477	33,2 ±3,458
100	36,6 ±4,409	40,3 ±4,148
120	49,0 ±3,066	49,3 ±4,026

Fonte: Autoria própria

Figura 10: Tempo total de execução da rotina de requisições, em segundos, para REST e SOAP variando o número de *threads*.



Fonte: Autoria própria

O tempo de execução é o intervalo de tempo, em segundos, entre o início da execução da rotina de interações, descrita na Seção 3.2.1, até o final de sua execução realizada pelo emulador de clientes. O final da execução é determinado pelo momento de recebimento no cliente da resposta da última requisição ao servidor. O valor-p mostrou que os resultados não são significativamente diferentes. O tempo total foi de aproximadamente 17 segundos com 20 *threads* e 49 segundos com 120 *threads*.

6 CONCLUSÃO

Sistemas distribuídos estão presentes em diversos setores de negócio e sua utilização é ampla por toda a internet. A escolha da *middleware* no desenvolvimento de um projeto de software para um sistema distribuído deve considerar a eficiência de cada tecnologia, além dos requisitos do sistema e recursos disponíveis. As *middlewares* para serviços web mais populares, SOAP e REST, possuem características muitos distintas em relação ao seu surgimento, arquitetura e funcionamento. Essas características distintas resultam em diferenças no desempenho de cada *middleware*, inclusive em diferentes cenários. O contexto de aplicações *e-commerce* é relevante, dado o surgimento crescente das lojas virtuais e plataformas de compra como eBay, Amazon, Mercado Livre e AliExpress.

No presente trabalho realizou-se uma análise de desempenho das *middlewares* SOAP e REST em um cenário de sistema *e-commerce*. Para isso, as aplicações de *e-commerce* foram desenvolvidas empregando as duas *middlewares*. As métricas analisadas foram utilização de CPU, memória RAM, disco, rede e tempo de execução de uma rotina de interações entre clientes e serviço web.

No caso da utilização de CPU, não foi verificada diferença de desempenho entre as *middlewares* com um número de clientes variando de 20 a 120.

Para a memória RAM, a *middleware* SOAP demonstrou uma utilização maior do recurso, sendo assim menos eficiente que o REST para qualquer número de clientes. Também foi possível observar que a utilização de memória RAM independe da quantidade de clientes, apresentando um comportamento constante durante o experimento.

Acerca dos resultados da utilização de disco, ao utilizar 20, 60 e 120 clientes o desempenho das *middlewares* não são estatisticamente diferentes. Porém, com 40 clientes, o SOAP utilizou mais disco. Já com um número maior de clientes (80 e 100) o REST utilizou mais disco do que o SOAP.

A análise da utilização de rede mostrou que a *middleware* REST é mais eficiente que o SOAP, mesmo com a utilização do XML como formato de dados em ambas as aplicações. O tempo total de execução aferido foi equivalente para as duas *middlewares*. Este tempo foi crescente de acordo com o número de acessos simultâneos de clientes.

Em resumo, pode-se concluir que a *middleware* REST mostrou-se mais eficiente que SOAP quanto à utilização de memória RAM e rede, sendo que o percentual de ganho foi de 9,6% e 41,5%, respectivamente.

Quanto à utilização de CPU e tempo de execução, ambas podem ser consideradas equivalentes. Não foi possível determinar qual das *middlewares* é mais eficiente em relação à

utilização de disco. Esses resultados devem ser considerados dentro do escopo de sistemas de *e-commerce* e com número de clientes limitado entre 20 e 120.

6.1 TRABALHOS FUTUROS

Como trabalhos futuros pretende-se avaliar as *middlewares* SOAP e REST com outras aplicações *benchmark*, incluindo outras linhas de negócio além de *e-commerce*. Além disso, pode-se comparar o desempenho delas com outras *middlewares* como o gRPC.

REFERÊNCIAS

ALONSO, Gustavo et al. **Web Services: Concepts, Architectures and Applications**. 1. ed. [S.l.]: Springer, 2004. ISBN 9783642078880. Citado na página 23.

AMAZON.COM, Inc. **Amazon**. 2021. Disponível em: <https://www.amazon.com/>. Acesso em: 15 mar. 2021. Citado na página 29.

AMAZON.COM, Inc. **Amazon.com - Marketplace for Web Service**. 2021. Disponível em: <https://developer.amazonservices.com/>. Acesso em: 10 abr. 2021. Citado na página 23.

AMZA, Cristiana et al. Specification and implementation of dynamic web service benchmarks. IEEE, 2002. Citado na página 29.

AUTHORS gRPC. **gRPC - A high performance, open source universal RPC framework**. 2021. Disponível em: <https://grpc.io/>. Acesso em: 6 abr. 2021. Citado na página 20.

CECCHET, Emmanuel et al. Performance comparison of middleware architectures for generating dynamic web content. Springer, Berlin, Heidelberg, 2003. Citado na página 29.

CLARO1, Daniela Barreiro et al. Avaliação de desempenho de serviços web soap e restful: estudos de caso para embarcar em dispositivos na internet das coisas. 2016. Citado na página 30.

CORPORATION, Oracle. **Remote Method Invocation Home**. 2021. Disponível em: <https://www.oracle.com/java/technologies/javase/remote-method-invocation-home.html>. Acesso em: 6 abr. 2021. Citado na página 20.

COULOURIS, George et al. **Distributed Systems: Concepts and Design**. 5. ed. [S.l.]: Pearson, 2012. ISBN 9780132143011. Citado 5 vezes nas páginas 17, 19, 20, 21 e 22.

COUNCIL, Transaction Processing Performance. Tpc benchmark w (web commerce) - especification. 2003. Citado 2 vezes nas páginas 27 e 28.

COUNCIL, Transactional Processing Performance. **TPC Homepage**. 2021. Disponível em: <http://tpc.org/default5.asp>. Acesso em: 14 abr. 2021. Citado na página 27.

CURBERA, Francisco. Unraveling the web services web: An introduction to soap, wsdl, and uddi. 2002. Citado na página 23.

FIELDING, Roy Thomas. Architectural styles and design of network-based software architecture. 2000. Citado na página 24.

GITHUB, Inc. **GitHub REST API - GitHub Docs**. 2021. Disponível em: <https://docs.github.com/en/rest>. Acesso em: 10 abr. 2021. Citado na página 23.

GODARD, Sebastien. **sar Manual Page**. 2021. Disponível em: <http://sebastien.godard.pagesperso-orange.fr/man_sar.html>. Acesso em: 29 abr. 2021. Citado na página 43.

GOOGLE. **Google Fit - Google developers**. 2021. Disponível em: <https://developers.google.com/fit>. Acesso em: 10 abr. 2021. Citado na página 23.

INC. eBay. **eBay**. 2021. Disponível em: <https://www.ebay.com/>. Acesso em: 15 mar. 2021. Citado na página 29.

INC., Object Management Group. **CORBA**. 2021. Disponível em: <https://www.corba.org/>. Acesso em: 6 abr. 2021. Citado na página 20.

INFOMONEY. **Faturamento das vendas online cresce 41% no Brasil em 2020; veja 5 tendências vencedoras**. 2021. Disponível em: <https://www.infomoney.com.br/negocios/faturamento-das-vendas-online-cresce-41-no-brasil-em-2020-veja-5-tendencias-vencedoras/>. Acesso em: 13 mai. 2021. Citado na página 17.

JAIN, Raj. **Art of Computer Systems Performance Analysis Techniques For Experimental Design Measurements Simulation And Modeling**. [S.l.]: Wiley Computer Publishing, 1991. ISBN 0471503363. Citado 2 vezes nas páginas 25 e 26.

JONG, Irmen de. **Pyro - Python Remote Objects**. 2021. Disponível em: <https://pyro5.readthedocs.io>. Acesso em: 6 abr. 2021. Citado na página 20.

JSON. **Introducing JSON**. 2021. Disponível em: <https://www.json.org>. Acesso em: 6 abr. 2021. Citado na página 21.

MALESHKOVA, Maria. Restful or restless current state of art of today's top web apis. 2014. Citado na página 24.

MENASCé, Daniel A. Tpc-w a benchmark for e-commerce. 2002. Citado 2 vezes nas páginas 27 e 29.

MORO, Tharcis Dal; DORNELES, Carina F.; REBONATTO, Marcelo Trindade. Web services ws-* versus web services rest. 2011. Citado na página 30.

NEUMANN, Andy; LARANJEIRO, Nuno; BERNARDINHO, Jorge. An analysis of public rest web service apis. IEEE Transactions on Services Computing, 2018. Citado 2 vezes nas páginas 24 e 25.

NEWCOMER, Eric. **Understanding Web Services: XML, WSDL, SOAP and UDDI**. 1. ed. [S.l.]: AddisonWesley Professional, 2002. ISBN 9780201750812. Citado 2 vezes nas páginas 21 e 22.

SMARTBEAR. **Simulating different types of Load**. 2021. Disponível em: <https://www.soapui.org/docs/load-testing/simulating-different-types-of-load/>. Acesso em: 25 abr. 2021. Citado na página 36.

SOFTWARE, SmartBear. **SwaggerUI - REST API Documentation Tool**. 2021. Disponível em: <https://swagger.io/tools/swagger-ui/>. Acesso em: 10 abr. 2021. Citado na página 25.

STATISTA. **E-commerce worldwide - Statistics & Facts**. 2021. Disponível em: <https://www.statista.com/topics/871/online-shopping/#dossierSummary>. Acesso em: 13 mai. 2021. Citado na página 17.

VOIGT, Ricardo; JUNIOR, Osmar de Oliveira Braz. Análise de desempenho de arquitetura soap e rest para comunicação entre sistemas. 2017. Citado na página 29.

W3C. **Soap Specifications**. 2021. Disponível em: <https://www.w3.org/TR/soap/>. Acesso em: 6 abr. 2021. Citado na página 22.