

---

Marlon Henry Schweigert

*Análise de arquiteturas de microsserviços empregados a jogos  
MMORPG voltada a otimização do uso de recursos  
computacionais*

---

Joinville

2018

**UNIVERSIDADE DO ESTADO DE SANTA CATARINA**  
**BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO**

**Marlon Henry Schweigert**

**ANÁLISE DE ARQUITETURAS DE MICROSERVIÇOS  
EMPREGADOS A JOGOS MMORPG VOLTADA A  
OTIMIZAÇÃO DO USO DE RECURSOS  
COMPUTACIONAIS**

Trabalho de conclusão de curso submetido à Universidade do Estado de Santa Catarina como parte dos requisitos para a obtenção do grau de Bacharel em Ciência da Computação

**Charles Christian Miers**  
**Orientador**

Joinville, Dezembro de 2018

**ANÁLISE DE ARQUITETURAS DE MICROSSERVIÇOS  
EMPREGADOS A JOGOS MMORPG VOLTADA A  
OTIMIZAÇÃO DO USO DE RECURSOS  
COMPUTACIONAIS**

Marlon Henry Schweigert

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Bacharelado Ciência da Computação em turno Integral do CCT/UDESC.

Banca Examinadora

---

Charles Christian Miers - Doutor (orientador)

---

Débora Cabral Nazário - Doutora

---

Guilherme Piêgas Koslovski - Doutor

# Agradecimentos

Aos meus pais, pelo amor, incentivo e apoio incondicional durante toda a minha jornada.

Ao professor Doutor Charles Christian Miers, pela orientação e apoio para a elaboração deste trabalho.

Agradeço a todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação do processo de formação profissional. Portanto, que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender. A palavra mestre, nunca fará justiça aos professores dedicados aos quais, sem nominar, terão os meus eternos agradecimentos.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

*“É um erro acreditar que é possível  
resolver qualquer problema importante  
usando batatas.”*  
*(Douglas Adams)*

## Resumo

A crescente popularização de jogos *Massively Multiplayer Online Role-Playing Game* (MMORPG) demanda por novas abordagens tecnológicas, a fim de suprir as necessidades dos usuários com menor custo de recursos computacionais. Projetar essas arquiteturas, do ponto de vista da rede, é pertinente e impactante para o sucesso desses jogos. O objetivo deste trabalho é realizar uma análise voltada a identificar os recursos computacionais consumidos pelas arquiteturas de microsserviços Rudy, Salz e Willson, a qual são arquiteturas de microsserviços elaboradas para jogos MMORPG. Esse objetivo será atingido após realizar uma pesquisa referenciada, seguida de uma análise das principais arquiteturas e, preferencialmente, a execução de testes utilizando simulações de clientes sobre as arquiteturas implantadas em uma nuvem computacional para auxiliar na identificação de gargalos de recursos. Espera-se que os resultados obtidos possam auxiliar provedores de serviços MMORPG a reduzir gastos de manutenção e melhorar a qualidade de tais serviços.

**Palavras-chaves:** Arquitetura de Microsserviços, Desenvolvimento de Jogos, Rede de Jogos, Jogos Massivos, Otimização de Recursos, Nuvem de Computadores.

## Abstract

The increasing popularization of *Massively Multiplayer Online Role-Playing Game* (MMORPG) demands new technological approaches in order to supply the requirements of users with lower cost of computational resources. Designing these architectures, from the network point of view, is relevant and impacting to the success of these games. The objective of this work is to propose an analysis aimed at identifying approaches to optimize the computational resources consumed by the Rudy, Salz and Willson micro-services architectures. This objective will be achieved after conducting a referenced search, followed by an analysis of the main architectures and, respectively, the execution of clients simulations using MMORPG architectures in a computational cloud to help identifying resources bottlenecks. The results obtained will help MMORPG service providers to reduce maintenance costs and improve the quality of such services.

**Keywords:** Microservice Architecture, Game Development, Game Network, Massive Games, Resource Optimization, Cloud Computing.

# Sumário

<b>Lista de Figuras</b>	<b>8</b>
<b>Lista de Tabelas</b>	<b>11</b>
<b>Lista de Abreviaturas</b>	<b>12</b>
<b>1 Introdução</b>	<b>14</b>
<b>2 Fundamentação Teórica</b>	<b>17</b>
2.1 Jogos Eletrônicos . . . . .	18
2.1.1 Árvore de gêneros de jogos eletrônicos . . . . .	19
2.2 MMORPG . . . . .	23
2.3 Jogabilidade de jogos MMORPG . . . . .	24
2.4 Problemas em jogos MMORPG . . . . .	28
2.4.1 Arquitetura de Clientes MMORPG . . . . .	30
2.4.2 Arquitetura de Microsserviços . . . . .	33
2.4.3 Microsserviços para jogos MMORPG . . . . .	35
2.5 Arquiteturas MMORPG identificadas . . . . .	38
2.5.1 Arquitetura elaborada por Rudy . . . . .	39
2.5.2 Arquitetura elaborada por Salz . . . . .	42
2.5.3 Arquitetura elaborada por Willson . . . . .	45
2.6 Definição do Problema . . . . .	47
2.7 Considerações parciais . . . . .	49
<b>3 Trabalhos Relacionados</b>	<b>51</b>

3.1	Huang et al. (2004) . . . . .	51
3.2	Villamizar et al. (2016) . . . . .	53
3.3	Suznjevic e Matijasevic (2012) . . . . .	55
3.4	Análise dos trabalhos relacionados . . . . .	58
3.5	Considerações parciais . . . . .	59
<b>4</b>	<b>Proposta para análise de arquiteturas MMORPG</b>	<b>61</b>
4.1	Proposta . . . . .	62
4.2	Critérios de análise . . . . .	63
4.3	Plano de testes . . . . .	65
4.3.1	Cenário . . . . .	67
4.3.2	Simulação de Clientes . . . . .	70
4.3.3	Testes . . . . .	74
4.4	Considerações parciais . . . . .	74
<b>5</b>	<b>Desenvolvimento das arquiteturas propostas</b>	<b>76</b>
5.1	Topologia da Rede . . . . .	77
5.2	Sistema de coleta de informações de recursos . . . . .	79
5.3	Arquitetura Rudy . . . . .	81
5.4	Dados obtidos da arquitetura Rudy . . . . .	83
5.4.1	Consumo de CPU pelo serviço Rudy . . . . .	84
5.4.2	Consumo de Memória pelo serviço Rudy . . . . .	84
5.4.3	Consumo de Banda pelo serviço Rudy . . . . .	86
5.4.4	Tempo de Resposta pelo cliente Rudy . . . . .	87
<b>6</b>	<b>Considerações &amp; Próximos passos</b>	<b>91</b>
6.1	Cronograma . . . . .	93
6.2	Etapas realizadas . . . . .	93

6.3	Etapas a realizar . . . . .	94
6.4	Execução do cronograma . . . . .	94
	<b>Referências</b>	<b>95</b>

# Listas de Figuras

2.1	Árvore de gêneros de jogos eletrônicos simplificada.	19
2.2	Sistema de autenticação para jogos MMORPG.	25
2.3	Área de interesse com base na proximidade de um jogador.	26
2.4	Chat baseado em contexto de posicionamento, utilizando Distância Euclidiana.	27
2.5	Personagens e os seus pontos de destino.	27
2.6	Personagens, objetos e <i>Non-Playable Characters</i> (NPCs) no ambiente.	28
2.7	Exemplo de Cliente MMORPG (Sandbox-Interactive Albion).	30
2.8	<i>Scene tree view</i> no motor gráfico Godot.	31
2.9	Modelo de um cliente genérico.	32
2.10	Microsserviços podem ter diferentes tecnologias.	34
2.11	Microsserviços são escaláveis.	34
2.12	Cliente pode realizar requisições <i>Create Read Update Delete</i> (CRUD) ao serviço.	36
2.13	Diagrama de requisições entre serviço e cliente com operações CRUD e <i>Remote Procedure Call</i> (RPC) em uma arquitetura monolítica.	37
2.14	Diagrama de requisições entre serviço e cliente com operações CRUD e RPC em uma arquitetura de microsserviços.	37
2.15	Diagrama de integração entre Cliente e Serviço, considerando a <i>engine</i> Unity3D.	38
2.16	Arquitetura Rudy completa.	39
2.17	Arquitetura Rudy completa com <i>firewall</i> .	41
2.18	Modelo de processos <i>Thread Pool</i> .	41
2.19	Arquitetura Salz.	43

2.20	Modelo de paralelismo do serviço de jogo na arquitetura Salz.	44
2.21	Arquitetura Willson.	46
2.22	Arquitetura de um jogo MMORPG genérico.	48
3.1	Arquitetura distribuída utilizando proxy.	52
3.2	Número de conexões no serviço pelo tempo decorrido.	52
3.3	Regressão linear comparado ao consumo de banda real do servidor.	53
3.4	Arquitetura monolítica web implementada na <i>Amazon Web Services</i> (AWS).	54
3.5	Arquitetura de microserviços web implementada na AWS.	55
3.6	Arquitetura de microserviços web implementada na AWS utilizando a tecnologia <i>lambda</i> .	56
3.7	Custo por um milhão de requisições em dólares utilizando diferentes arquiteturas sobre a AWS.	56
3.8	Regressão levando em conta a complexidade das ações e contexto dos personagens.	57
4.1	Ambiente de testes definido para a coleta de dados.	66
4.2	Rede de execução dos testes.	68
4.3	Área de interesse da simulação com raio de quatro células.	72
4.4	Autômato de movimentação dos personagens simulados.	72
5.1	Topologia da rede no gestor de redes do OpenStack.	77
5.2	Fluxo dos valores obtidos no cliente e serviço.	80
5.3	Consumo de <i>Central Processing Unit</i> (CPU) no servidor utilizando a arquitetura Rudy	85
5.4	Registro de memória no servidor utilizando a arquitetura Rudy	86
5.5	Memória consumida no servidor utilizando a arquitetura Rudy	86
5.6	Consumo de Banda no servidor utilizando a arquitetura Rudy	87
5.7	Tempo de Resposta do cliente servidor utilizando a arquitetura Rudy	88

5.8	Tempo de Resposta de requisições <i>Web</i> do cliente servidor utilizando a arquitetura Rudy . . . . .	89
5.9	Tempo de Resposta de requisições RPC do cliente servidor utilizando a arquitetura Rudy . . . . .	89

## **Lista de Tabelas**

2.1	Tipos de comunicação e quantidade de jogadores impactados por ocorrências conforme o seu gênero. . . . .	23
3.1	Complexidade da interação com o ambiente, por contexto da interação. . . . .	55
3.2	Trabalhos relacionados por categoria. . . . .	58
3.3	Trabalhos relacionados por recurso analisado. . . . .	59
3.4	Arquiteturas analisadas. . . . .	59
4.1	Possíveis conjuntos para a análise. . . . .	64
4.2	Tabela de interdependência das sub-redes. . . . .	69
4.3	Limite de recursos por instância de cada rede. . . . .	70
4.4	Requisitos das funcionalidades e respectivo impacto de implementação. . . . .	71
4.5	Requisitos mínimos funcionais para a implementação da simulação descrita. . . . .	71
5.1	Instâncias das redes de teste . . . . .	78
5.2	Orquestradores utilizados no teste. . . . .	79
5.3	Protocolos dos microsserviços da arquitetura Rudy. . . . .	82
5.4	Média, mediana, máximo e mínimo obtidos pelas requisições HTTP. . . . .	90

# **Lista de Abreviaturas**

**API** *Application Programming Interface*

**ORM** *Object-Relational Mapping*

**AWS** *Amazon Web Services*

**C/S** Cliente/Servidor

**CPU** *Central Processing Unit*

**CRUD** *Create Read Update Delete*

**FIFO** *First In First out*

**FPS** *First-Person Shooter*

**HTTP** *Hypertext Transfer Protocol*

**IDE** *Integrated Development Environment*

**IP** *Internet Protocol*

**JSON** *JavaScript Object Notation*

**JWT** *JSON Web Token*

**LAN** *Local Area Network*

**MMO** *Massively Multiplayer Online*

**MMORPG** *Massively Multiplayer Online Role-Playing Game*

**MOBA** *Multiplayer Online Battle Arena*

**MVC** *Model-View-Controller*

**NoSQL** *Not Only SQL*

**NPC** *Non-Playable Character*

**OS** *Operating System*

**P2P** *Peer-to-Peer*

**PaaS** *Platform as a Service*

**POF** *Point of View*

**PvNPCs** *Player vs NPCs*

**PvP** *Player vs Player*

**REST** *Representational State Transfer*

**RPC** *Remote Procedure Call*

**RPG** *Role-Playing Game*

**RTS** *Real-Time Strategy*

**SQL** *Structured Query Language*

**TCP** *Transmission Control Protocol*

**TPS** *Third-person Shooter*

**UDP** *User Datagram Protocol*

**VCPU** *Virtual Central Process Unit*

**VM** *Virtual Machine*

**WAN** *Wide Area Network*

**XDR** *External Data Representation*

**XML** *Extensible Markup Language*

# 1 Introdução

Jogos *Massively Multiplayer Online Role-Playing Game* (MMORPG) são utilizados como negócio viável e lucrativo, sendo que, a experiência de jogabilidade na qual o usuário final será submetido é um fator crítico para o sucesso. Este gênero, especificado de interpretação de papéis de forma massiva em um ambiente compartilhado, tem como sua principal característica a comunicação e representação virtual de personagens em um mundo fantasia compartilhado. Neste mundo compartilhado cada jogador pode interagir com objetos ou tomar ações sobre outros jogadores em tempo real, tendo como principais objetivos a resolução de problemas conforme a sua regra de negócio de cada projeto (HANNA, 2015).

A maioria dos jogos MMORPG disponíveis no mercado estão implementados sobre uma arquitetura que executa o serviço sobre diversos servidores (CLARKE-WILLSON, 2017), nos quais o desempenho dos servidores e serviços influenciam tanto na experiência de jogabilidade do usuário final, quanto no custo de manutenção destes serviços (HUANG; YE; CHENG, 2004). Modelar um sistema de alto desempenho torna-se um trabalho essencial para a satisfação do usuário final neste cenário (HUANG; YE; CHENG, 2004). As ocorrências geradas por um sistema de baixo desempenho podem acarretar em frustração do usuário com o serviço e/ou aumento dos gastos com recurso computacional para manter o serviço. Uma ocorrência é qualquer tipo de mal funcionamento em uma aplicação, não necessariamente aparente ao usuário final (HUANG; YE; CHENG, 2004). Evitar ou eliminar as ocorrências durante o projeto e desenvolvimento das arquiteturas do serviço é um processo crítico para o bom funcionamento desses jogos.

Os avanços tecnológicos de sistemas distribuídos estão permitindo que as pessoas utilizem serviços com volumes massivos de dados para aplicações sensíveis a latência. Essa situação é propícia à área de jogos massivos, e tem atraído pesquisadores (KIM; KIM; PARK, 2008; HUANG; YE; CHENG, 2004; SALDANA et al., 2012; BARRI; GINE; ROIG, 2011). O principal objetivo destas pesquisas é reduzir a carga e o impacto da latência para o usuário final nesses serviços, aumentando a quantidade de jogadores simultâneos em um único serviço. Reduzir a carga e impacto da latência em serviços para jogos massivos resulta em uma melhor experiência de jogabilidade aos usuários finais, sendo este um dos fatores críticos para o sucesso destes serviços (HUANG; YE; CHENG,

2004).

Entende-se por arquitetura de microsserviços uma arquitetura com diversas aplicações menores na qual utilizam troca de mensagens pela rede para implementar uma regra de negócio complexa, não exibindo necessariamente se este serviço é implementado por um ou mais microsserviços. Em específico, este paradigma de desenvolvimento de arquiteturas herda características da filosofia UNIX, descrito pelo matemático Malcolm Doug McIlroy em 2003 pela seguinte passagem:

*Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.*

(RAYMOND, 2003)

Alguns estudos discorrem sobre a escalabilidade dos serviços baseados em microsserviços a qual em um todo com o seu foco para desenvolvimento *web* (Exit Games, 2017; JON, 2010; FRANCESCO, 2017). Entretanto, a estrutura de um serviço MMORPG baseado em microsserviços é mais abrangente, necessitando de maior desempenho comparado a serviços monolíticos *web* (Exit Games, 2017; JON, 2010). Dessa forma, torna-se interessante a análise de serviços MMORPG baseados em microsserviços.

Deste modo, o problema principal identificado está na análise de consumo de recursos para a manutenção de arquiteturas de microsserviços específicos a jogos MMORPG utilizando como base arquiteturas encontradas na literatura. Adicionalmente a este problema, será necessário o desenvolvimento de tais arquiteturas para submissão a experimentos e análise.

Este trabalho tem como objetivo analisar o consumo de recurso das principais arquiteturas disponíveis na literatura. Nesse sentido, os objetivos específicos do atual trabalho são:

- Identificar arquiteturas empregadas na categoria de jogos do presente trabalho.
- Identificar os protocolos utilizados nessas arquiteturas.
- Identificar os microsserviços dessas arquiteturas.
- Identificar e avaliar ferramentas de análise de métricas para armazenar valores dos testes.

- Analisar o comportamento das arquiteturas aplicadas, levantando questões de desempenho e recursos utilizados.
- Propor alternativas de otimização para os problemas encontrados nas devidas arquiteturas.

Para dar suporte a proposta deste trabalho, uma revisão da literatura é apresentada, a fim de esclarecer os conceitos principais referentes a temática de serviços MMORPG. Ao analisar os trabalhos relacionados identificados, houve uma dificuldade para encontrar os tópicos MMORPG e sistemas distribuídos baseado em microsserviços. A relevância desta análise contribui, diretamente para que outros trabalhos científicos possam utilizar da análise de arquiteturas para jogos MMORPG baseados em arquiteturas de microsserviços.

A análise é inicialmente realizada através de uma pesquisa referenciada sobre arquiteturas de microsserviços e arquitetura de serviços MMORPG. Depois, um estudo sobre a intersecção de ambos os temas. Também são identificados trabalhos relacionados na presente literatura que foquem tanto em microsserviços tal como serviços MMORPG. Em seguida é definida a proposta, junto a testes e cenários a fim de analisar tais arquiteturas.

Este trabalho de conclusão de curso possui natureza aplicada pois será necessário a implementação das arquiteturas descritas na literatura (TCC-II), utilizando as mesmas regras de negócio, viabilizando uma análise igualitária entre as arquiteturas propostas. A análise será abordada de maneira qualitativa, pois será feito um estudo a partir de valores gerados por experimentos utilizando as ferramentas, abordando as características de tais arquiteturas.

Este trabalho está organizado em três capítulos, que dão suporte a análise das arquiteturas específicas a jogos MMORPG proposta. No Capítulo 2 são apresentados os conceitos necessários para o entendimento desse trabalho, com a finalidade de apresentar o funcionamento básico de um cliente e serviço MMORPG, arquitetura de um serviço baseado em microsserviços e por fim em específico algumas arquiteturas de microsserviços MMORPG encontradas na literatura. O Capítulo 3 aborda trabalhos relacionados encontrados na literatura, tendo como objetivo principal destacar e comparar suas características, a fim de prover fundamentos para a análise dos serviços descritos na referência teórica. A proposta é definida no Capítulo 4, a qual define o plano de extra-

ção de valores, a interpretação de tais valores e cenário a qual as arquiteturas propostas serão implantados.

## 2 Fundamentação Teórica

O termo *jogos eletrônicos* é amplamente difundido, entretanto as especificações, características e histórico deste termo não são de conhecimento popular. A Seção 2.1 trata a definição de jogo eletrônico, um levante histórico e o impacto da evolução do hardware no desenvolvimento dos jogos. Este termo é tomado como introdução para o conceito de gênero de jogo, abordado na Subseção 2.1.1 e na qual são abordados os principais gêneros, características e tecnologias (do ponto de vista de rede de computadores) que são comuns em cada gênero. Esta introdução acerca dos gêneros e suas tecnologias de comunicação busca trazer a importância do desempenho das arquiteturas dos jogos MMORPG e proporção da comunidade impactada caso haja falhas de funcionamento nessas arquiteturas.

Após definir a categoria de jogo abordado, a Seção 2.2 aborda sobre uma introdução ao impacto de mercado desse gênero, uma definição simplista e a divisão das camadas de aplicação que permeiam uma arquitetura para um jogo MMORPG. Entretanto, antes de abordar sobre as camadas da infraestrutura de um arquitetura MMORPG, faz-se obrigatório o entendimento sobre jogabilidade (Seção 2.3) e problemas relativos a rede relevantes a este gênero (Seção 2.4).

Os conceitos de Cliente (Seção 2.4.1) e Serviço (Seção 2.4.2) são abordados a fim de introduzir conceitos de arquiteturas comuns nestes serviços. O objetivo destas seções é referenciar diversas tecnologias e técnicas utilizadas nesses sistemas a fim de permitir o desenvolvimento de uma arquitetura de microsserviços específica a jogos MMORPG. Por fim, torna-se obrigatório a apresentação de trabalhos relacionados (Seção 3) a arquitetura de jogos MMORPG desenvolvidos de forma distribuída ou sobre uma arquitetura de microsserviços. Esta seção em específico aborda exemplos de métodos e métricas utilizadas para mensurar o desempenho de tais arquiteturas, realizando por fim uma análise destes trabalhos (Subseção 3.4).

## 2.1 Jogos Eletrônicos

O primeiro sistema de entretenimento interativo foi construído em 1947, utilizando como base de exibição um tubo de raios catódicos. Essa criação foi patenteada em janeiro de 1948, datando então o início dos jogos eletrônicos (ADAMS, 2014; GOLDSMITH, 1947).

O jogo eletrônico, ou entretenimento interativo, é uma atividade intelectual que integra um sistema de regras, na qual utiliza tal sistema a fim de definir seus objetivos ou pontuação por meio de um computador, com o objetivo de despertar alguma emoção ao jogador (HANNA, 2015). Os jogos eletrônicos são aplicações convencionais, que executam sobre algum sistema operacional ou hardware apropriado a este fim. O sistema operacional, hardware ou base de execução da aplicação gráfica define a sua plataforma (*e.g.*, GNU/Linux, MS-Windows, Sony PS4, MS-XBox, web, etc.) (ADAMS, 2006).

Inicialmente, os jogos eram implementados de forma simples por conta da limitação de hardware das plataformas dos anos 80. As implementações de jogos para *videogames* eram projetadas diretamente para algum hardware proprietário, sem sistema operacional, por muitas vezes sem utilizar comunicação por rede ou armazenamento em memória secundária (ROLLINGS; ADAMS, 2003). Além de diversas plataformas não terem acesso a rede, os serviços para jogos eram inviabilizados pelo custo de manutenção e pela ausência de demanda a qual teriam os requisitos mínimos para jogar (ADAMS, 2006). Na década de 80, o *videogame* Atari foi uma plataforma popular, vendendo 30.000 unidades em seu lançamento contra apenas 2.000 unidades do seu concorrente Intellivision (YARUSSO, 2006).

A crescente de recursos computacionais disponíveis em computadores pessoais e *videogames* após os anos 90, permitiu que desenvolvedores criassem novos estilos de jogos que utilizavam de hardware mais específico (ADAMS, 2006). Dentre esses recursos, iniciou-se o uso da rede de computadores para proporcionar a interação entre jogadores em equipamentos distintos (STATISTA, 2018a). Jogos como EA Habitat<sup>1</sup>, CipSoft Tibia<sup>2</sup> e Jajex Runescape<sup>3</sup> começaram a utilizar, como requisito obrigatório do jogo, a conexão com a Internet para interagir em um mundo compartilhado com outros jogadores. Tais jogos popularizaram um novo gênero, trazendo inovação tecnológica como complemento a sua jogabilidade, propondo novos desafios aos jogadores ao jogar com centenas ou milhares

<sup>1</sup>EA Habitat: <http://www.mobygames.com/game/c64/habitat/credits>

<sup>2</sup>CipSoft Tibia: <http://www.tibia.com/>

<sup>3</sup>Jajex Runescape: <https://www.runescape.com>

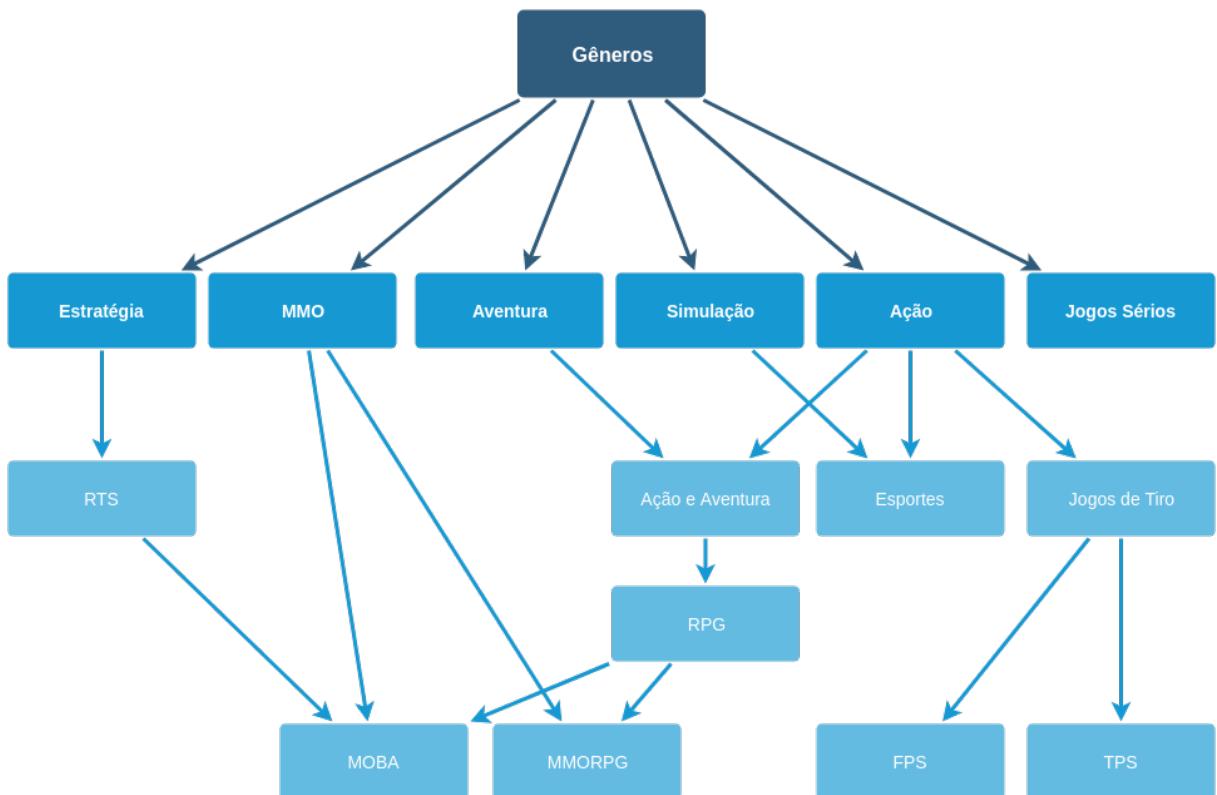
de jogadores simultâneos (GUINNESS, 2013; HUANG; YE; CHENG, 2004), criando o gênero de jogos *Massively Multiplayer Online* (MMO).

Muitos outros jogos do gênero foram criados devido ao aumento da quantidade de público alvo e viabilidade de novas tecnologias, na qual fomentaram a evolução dos jogos MMORPG. Nesse sentido, as redes de computadores realizaram papel de impulsor para várias categorias de jogos, que antes não eram possíveis por conta da limitação de comunicação entre computadores. Sendo assim, torna-se necessário ter uma visão geral das principais categorias de jogos eletrônicos com relação as tecnologias empregadas do ponto de vista de redes de computadores.

### 2.1.1 Árvore de gêneros de jogos eletrônicos

A classificação por gênero é uma ferramenta tradicional para auxiliar a fácil identificação de características de alguma literatura, arte e outras mídias. Dentro de jogos eletrônicos, o gênero permite que jogadores busquem jogos com características conforme o seu interesse (CLARKE; LEE; CLARK, 2015). A árvore pode ser visualizada na Figura 2.1.

Figura 2.1: Árvore de gêneros de jogos eletrônicos simplificada.



Adaptado de: (ADAMS, 2006)

Um gênero de jogo eletrônico é uma categoria específica para agrupar estilos

de jogabilidade parecidos. Porém, um gênero não define de forma explícita o conteúdo expresso em algum jogo eletrônico, mas sim um desafio comum presente no jogo analisado (ADAMS, 2006; HANNA, 2015). Uma breve explicação sobre os principais gêneros (Figura 2.1):

- Estratégia: São focados em uma jogabilidade que exija habilidades de raciocínio e/ou gerenciamento de recurso. Neste gênero, o jogador tem uma boa visualização do mundo, controlando indiretamente as suas tropas disponíveis (ROLLINGS; ADAMS, 2003). É comum encontrar jogos que disponibilizam algum modo de competição entre jogadores usando *Local Area Network* (LAN), *Wide Area Network* (WAN) ou *Peer-to-Peer* (P2P) (ADAMS, 2006).
  - *Real-Time Strategy* (RTS): Utiliza as características de um jogo de estratégia, porém esse subgênero indica que as ações dos jogadores são concorrentes. É comum encontrar modos de jogo competitivo utilizando LAN neste gênero (ADAMS, 2006).
- MMO: Preza pela interação com outros jogadores em um mundo compartilhado (ADAMS, 2006). SecondLife<sup>4</sup> é um jogo focado na interação social, com artifícios de comércio e relacionamentos em um mundo fictício criado pela comunidade (KLEINA, 2018). Em grande parte, esses jogos utilizam tecnologia WAN e Cliente/Servidor (C/S).
  - *Multiplayer Online Battle Arena* (MOBA): Coloca um número fixo de jogadores separados em dois times, no qual o time com maior estratégia de posicionamento e gerenciamento de recursos em equipe ganha a partida. Jogos MOBA perdem algumas características breves do gênero *Role-Playing Game* (RPG), deixando de lado a interpretação e contextualização de um mundo, fixando-se somente em um combate estratégico e momentâneo (distribuído em partidas atômicas) entre as equipes, carregando consigo somente as características de comércio e comunidade dos jogos MMO (ADAMS, 2006). Tal subgênero é popularmente conhecido pelos títulos Blizzard Dota 2<sup>5</sup> e Riot League of Legends<sup>6</sup>. O jogo League of Legends obteve 100 milhões de usuários ativos em 2016 sendo o jogo mais jogado do mundo neste ano (STATISTA, 2018c), tendo

<sup>4</sup>SecondLife: <https://www.secondlife.com/>

<sup>5</sup>Blizzard Dota 2: <http://br.dota2.com/>

<sup>6</sup>Riot League of Legends: <https://br.leagueoflegends.com/pt/>

torneios nacionais e internacionais (SPORTV, 2018). É popular nesse subgênero utilizar tecnologias como LAN, P2P e WAN.

- MMORPG: Herda características dos gêneros ação e aventura, RPG, e MMO diretamente. Nesse gênero é permitido interações em um mundo na qual outros jogadores conjuntamente jogam, na qual a interação entre outros jogadores (herdado dos jogos MMO), com o mundo (herdado dos jogos de ação e aventura) e com objetivos guiados por NPCs (herdados de jogos RPG) se faz como desafio e objetivo do jogo (ADAMS, 2006). Um título popular para esse gênero é o jogo Blizzard Word of WarCraft, a qual tem o título de maior comunidade de jogo em um único serviço do mundo<sup>7</sup>. A grande parte dos jogos MMORPG utilizam tecnologia WAN e C/S.
- Aventura: Caracterizado por desafios envolvendo ações com diversos NPCs ou com o ambiente para solucionar desafios (ADAMS, 2006). A grande parte desses jogos utilizam arquiteturas WAN, P2P ou LAN.
  - Ação e Aventura: Herda características da categoria de Aventura. O jogador é imerso em um mundo para interagir com o ambiente e com NPCs, além de se preocupar com a movimentação no cenário (ADAMS, 2006). Um título popularmente conhecido desse gênero é a série de jogos nomeada Nintendo The Legend of Zelda<sup>8</sup>. É comum nesses jogos encontrar tecnologia LAN ou P2P para modo de jogo cooperativo.
- Simulação: Caracterizados por abordar temas da realidade. São comuns jogos de construção e gerenciamento, animais de estimação, vida social e simulação de veículos (ADAMS, 2006). A grande parte desses jogos não permite a interação entre os demais jogadores. É popular encontrar serviços como *ranking*, loja e janela de notícias utilizando C/S.
  - Esportes: Trata somente da simulação de esportes, nos quais o(s) time(s) podem ser controlados tanto por uma inteligência artificial quanto por jogadores *online* (ADAMS, 2006). O jogo FIFA<sup>9</sup> é um título popular nesse segmento. É comum encontrar tecnologias P2P e LAN.

<sup>7</sup>Blizzard Word of WarCraft: <https://worldofwarcraft.com/pt-br/>

<sup>8</sup>Nintendo The Legend of Zelda: <https://www.zelda.com/>

<sup>9</sup>FIFA: <https://www.easports.com/br/fifa>

- Ação: Preza pela habilidade de coordenação motora e reflexos do jogador, para tomar uma ação a fim de superar seus desafios no cenário alcançando algum objetivo (ADAMS, 2006). É comum encontrar tecnologias LAN, P2P, WAN e C/S.
  - Jogos de Tiro: Usa um número finito de armas para executar ações a distância. O posicionamento, movimentação estratégica e mira são fatores de desafio ao jogador nesse gênero (ADAMS, 2006). É comum encontrar tecnologias LAN, P2P ou WAN.
    - \* *First-Person Shooter* (FPS): Utiliza o método de gravação conhecido como *Point of View* (POV). Nesse método, o modo de exibição do mundo é dado como a visão de um personagem do jogo, na qual o jogador tem visão pelo próprio personagem (HANNA, 2015; ADAMS, 2006). É comum encontrar tecnologias LAN, P2P ou WAN.
    - \* *Third-person Shooter* (TPS): Diferente dos jogos FPS, os jogos TPS utilizam câmeras soltas no cenário no qual o jogador é visível na cena exibida (HANNA, 2015; ADAMS, 2006). É comum encontrar tecnologias LAN, P2P ou WAN.
- Jogos sérios: Tem como objetivo transmitir um conteúdo educacional (HANNA, 2015). O jogo Sherlock Dengue 8 (BUCHINGER, 2014) é um título desenvolvido com o objetivo de conscientizar os problemas e a prevenção da Dengue no Brasil. É comum encontrar tecnologias LAN, P2P, WAN e C/S.

A árvore de gêneros guia tanto os usuários finais para classificar jogos que lhe agradem, quanto desenvolvedores a fim de seguir tendências de mercado pelas características do gênero. Dessa forma, encontra-se um padrão nos jogos, o qual inclusive orienta o desenvolvimento das arquiteturas de tais jogos (HANNA, 2015).

Dentre vários gêneros, alguns utilizam popularmente algumas tecnologias de rede. A Tabela 2.1 indica a correlação de tecnologias de rede comuns nos gêneros, além de trazer a correlação de número de jogadores por gênero de jogo em seus serviços. Essa correlação é importante para identificar as características de jogabilidade referentes a jogabilidade com multijogadores (HANNA, 2015).

Dentre todos os jogos, o gênero MMORPG é o mais impactado pela quantidade de jogadores(KIM; KIM; PARK, 2008), visível na Tabela 2.1. Nesse sentido, as

Tabela 2.1: Tipos de comunicação e quantidade de jogadores impactados por ocorrências conforme o seu gênero.

	LAN	WAN	P2P	C/S	Jogadores Impactados por falhas
ESTRATÉGIA	✓	✓	✓		até 10 (MICROSOFT, 2005)
RTS	✓	✓		✓	até 10 (BLIZZARD, 2010)
MOBA	✓	✓	✓	✓	até 10 (RIOT, 2009)
MMO		✓		✓	mais que 1000 (JAJEX, 2018)
MMORPG		✓		✓	mais que 1000 (JAJEX, 2018)
AVENTURA	✓	✓	✓	✓	até 100 (MOJANG, 2009)
AÇÃO	✓	✓	✓	✓	até 10 (MDHR, 2017)
AÇÃO E AVENTURA	✓	✓	✓	✓	até 10 (MDHR, 2017)
SIMULAÇÃO	✓	✓	✓	✓	até 10 (SCS, 2016)
ESPORTES	✓	✓	✓	✓	até 10 (EA, 2018)
FPS	✓	✓	✓	✓	até 100 (DICE, 2013)
TPS	✓	✓	✓	✓	até 100 (DICE, 2013)
JOGOS SÉRIOS	✓	✓	✓	✓	até 10 (BUCHINGER, 2014)

Fonte: O próprio autor.

arquiteturas do serviço e cliente se tornam um ponto crítico no desenvolvimento a fim de suportar a carga necessária ao desenho do jogo. Por esse motivo, a escolha por abordar o gênero MMORPG se torna interessante do ponto de vista computacional, a fim de analisar o comportamento e consumo de jogos MMORPG, visando obter características destas arquiteturas.

## 2.2 MMORPG

Jogos MMORPG são utilizados como negócio viável e lucrativo, sendo que a experiência de jogabilidade na qual o usuário final será submetido é um fator crítico para o sucesso. O mercado de jogos MMORPG vem crescendo desde 2012 (BILTON, 2011), sendo no ano de 2017 um dos mais lucrativos (STATISTA, 2018b). A projeção deste mercado para 2018 era de mais de 30 bilhões de dólares americanos em circulação sobre esta categoria de jogos (STATISTA, 2017), porém foi ultrapassado no ano de 2017 com 30,7 bilhões de dólares (STATISTA, 2018b).

MMORPG são jogos de interpretação de papéis massivos, originados dos gêneros RPG. A principal característica desse estilo de jogo é a comunicação e representação virtual de um mundo fantasia no qual cada jogador pode interagir com objetos virtuais compartilhados ou tomar ações sobre outros jogadores em tempo real, tendo como principais objetivos a resolução de problemas conforme a sua regra de *design*, o desenvolvimento

do personagem e a interação entre os jogadores(HANNA, 2015).

Um jogo MMORPG é arquitetado em três partes (KIM; KIM; PARK, 2008):

- **Cliente:** Aplicação que realiza as requisições com a interface do serviço, exibindo o estado de jogo de forma imersiva ao jogador. Este tema é melhor abordado na Subseção 2.4.1.
- **Servidor:** O computador, ou conjunto de computadores, que recebe as requisições do cliente a fim de serem processadas pelo Serviço.
- **Serviço:** Implementa as regras de negócio e requisitos do jogo. O serviço disponibiliza uma interface com ações possíveis ao cliente sobre algum protocolo de rede. Este tema é abordado na Subseção 2.4.2.

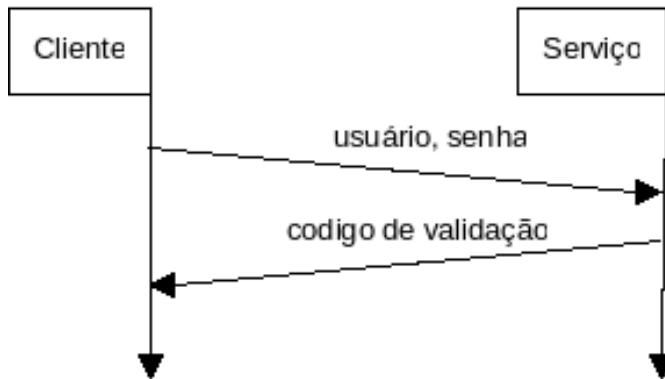
Jogos MMORPG trabalham como serviço. Por este motivo, o modelo de negócios de tais jogos, do ponto de vista de redes de computador, são suscetíveis a perda financeira em casos de negação de serviço (HUANG; YE; CHENG, 2004). A maioria dos jogos MMORPG disponíveis no mercado estão implementados sobre uma arquitetura que executa sobre diversos servidores (CLARKE-WILLSON, 2017), nos quais o desempenho destes servidores influencia tanto na experiência de jogabilidade do usuário final, quanto no custo de manutenção destes serviços (HUANG; YE; CHENG, 2004). Por sua vez, o Cliente é implementado em algum ambiente convencional a jogos, como motores gráficos, bibliotecas gráficas ou sobre alguma outra plataforma, como web. Nesse sentido, torna-se necessário descrever as características de jogabilidade de jogos MMORPG a fim de melhor compreender o funcionamento da arquitetura de um cliente e de um serviço para jogos MMORPG.

## 2.3 Jogabilidade de jogos MMORPG

É comum serviços MMORPG terem regras de negócio parecidas, visto que pertencem ao mesmo gênero. Dessa forma, facilita a compreensão básica de forma genérica de um jogo MMORPG e auxilia a compreender o modelo computacional implementado nestes serviços explicar regras de negócio recorrentes neste gênero. Deste modo, torna-se necessário definir algumas funcionalidades básicas que estão dentro do contexto de jogabilidade de jogos MMORPG para melhor compreensão de sua arquitetura em seções futuras.

O sistema de autenticação é o sistema que geralmente inicia o cliente de algum jogo MMORPG (SALZ, 2016a; RUDDY, 2011). Este sistema é implementado via protocolo *web* ou RPC, a fim de disponibilizar um código para validar todas as futuras ações da seção do usuário. Este pode ser visualizado de forma macro na Figura 2.2.

Figura 2.2: Sistema de autenticação para jogos MMORPG.



Fonte: Adaptado de (THOMPSON, 2008)

O sistema de autenticação visualizado na Figura 2.2 é o principal, porém não o único. O jogo *Realm of the Mad God*<sup>10</sup> é um exemplo na qual não exige autenticação do usuário, entretanto ele não armazena o progresso do jogador caso o jogador não efetue a autenticação.

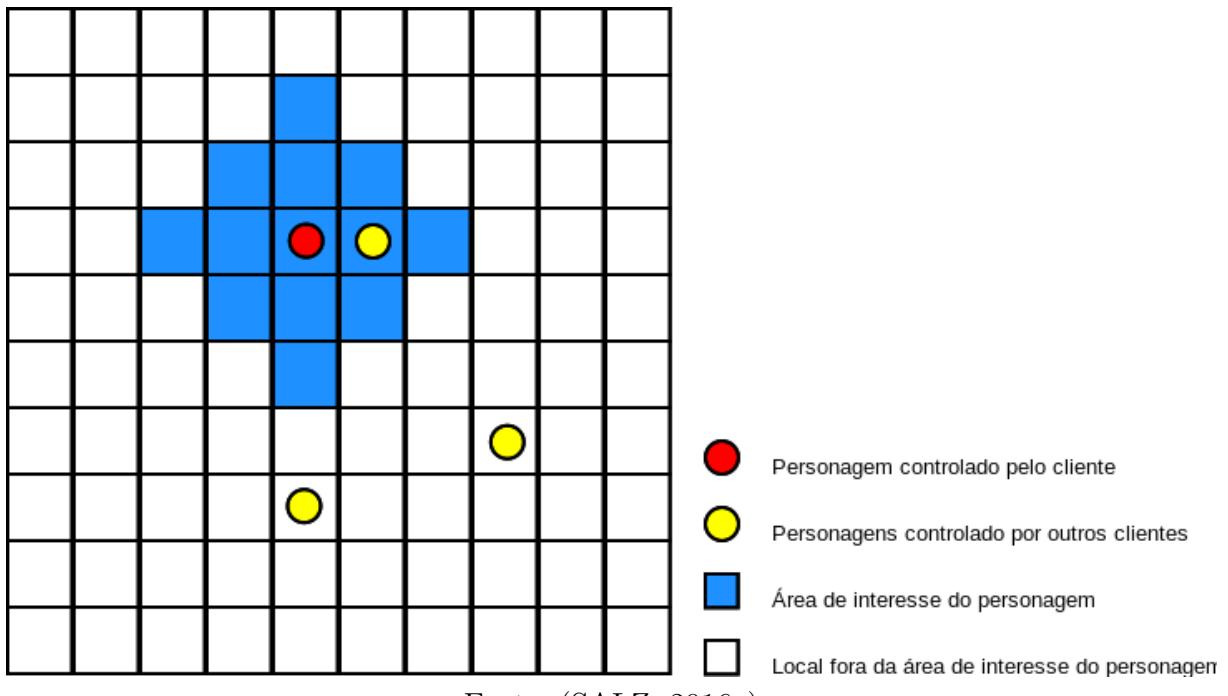
Esse método de autenticação é definido pela RFC7519 (JONES et al., 2015), com a tecnologia *JSON Web Token* (JWT). O código de validação repassado é auditado em qualquer serviço pertencente ao jogo, visto que ele foi assinado pelo sistema de autenticação do serviço (IKEM, 2018).

Após a autenticação, é comum existir um sistema para seleção de personagem, caso o jogo seja desenhado com este objetivo. Efetuada a seleção ou criação de um personagem, este será imerso no mundo compartilhado do jogo com os demais jogadores (RUDDY, 2011). Nos jogos MMORPG é comum a restrição da visão do personagem (Figura 2.3), ora pelas características de jogabilidade do gênero MMORPG ora por motivos de desempenho e otimização. Como o jogador não precisa obter dados de regiões que não estão em sua área de interesse, não há necessidade da transmissão de informações dos objetos que estão fora desse contexto (SALZ, 2016a).

Esse caso pode ser visualizado na Figura 2.3, na qual o personagem selecionado (destacado em vermelho) tem uma área de interesse de baixa distância (SALZ, 2016a),

<sup>10</sup>Realm of the Mad God: <https://www.realmofthemadgod.com/>

Figura 2.3: Área de interesse com base na proximidade de um jogador.



sendo que o jogador não tem informações dos demais objetos e jogadores fora de sua área de interesse a nível de rede. Esta característica impede trapaças (visto que o cliente tem informações que só estão contidas em sua área de interesse) e reduz a frequência de atualização a cada cliente (SALZ, 2016a).

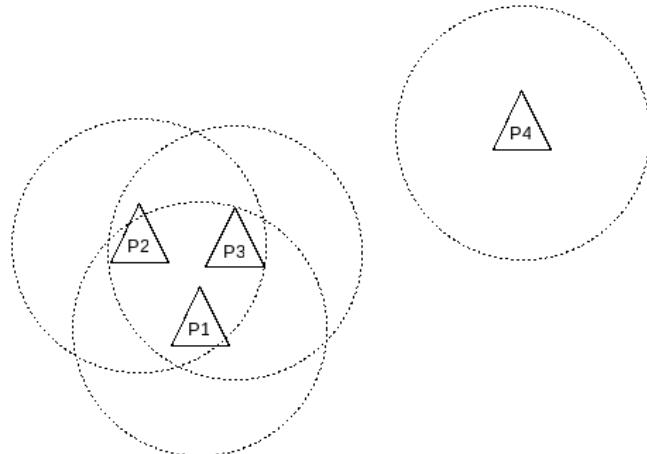
Após o jogador estar com o controle do personagem no ambiente, é comum em tais jogos que ele possa realizar determinadas ações. Nesse sentido, algumas ações comuns dentro do ambiente de um jogo MMORPG (JON, 2010):

- Enviar e receber mensagem no chat;
- Mover-se pelo ambiente;
- Interagir com outros jogadores, NPCs ou objetos fixos do ambiente; e
- Obter itens do ambiente.

O envio e recepção de mensagens do chat é dado com o contexto do posicionamento do personagem (SALZ, 2016a), visível na Figura 2.4. Somente outros personagens dentro de uma distância podem receber alguma mensagem emitida pelo jogador  $P_n$ .

Essa distância (Figura 2.4) pode ser calculada utilizando Distância Euclidiana (DEZA, 2009), na qual a distância entre dois personagens podem ser calculadas pela

Figura 2.4: Chat baseado em contexto de posicionamento, utilizando Distância Euclidiana.



Fonte: Adaptado de (SALZ, 2016a)

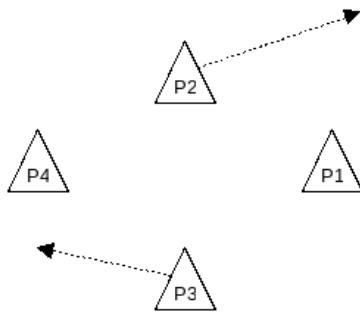
equação  $d(p, q) = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$ . Para diminuir a complexidade das comparações, a fim de decidir quais personagens  $P_n$  devem receber a mensagem, é comum utilizar técnicas de divisão de área utilizando algoritmos como *Quadtree* ou *Octree* (LENGYEL, 2011), subdividindo os quadrantes de uma região do ambiente do jogo a fim de facilitar a consulta de quais personagens estão em determinada área deste ambiente.

Nesse sentido, a Figura 2.4 mostra a interseção entre o raio de quatro personagens. Nesse exemplo, mostra-se visível que as mensagens de  $P_1$  devem ser visíveis a  $P_2$  e  $P_3$ , mas não a  $P_4$ , caso seja utilizado a Distância Euclidiana como regra de distância.

O sistema de movimento pelo ambiente do jogo possibilita que cada jogador movimente seu personagem pelo ambiente a fim de explorá-lo. Dessa maneira, este é um sistema crítico para um jogo MMORPG, visto que o posicionamento de objetos serão utilizados para diversas consultas de proximidade, além de necessitar uma frequência de atualização constante do posicionamento dos jogadores a fim de manter a integridade de funcionamento do serviço de ambiente do jogo (SALZ, 2016a). Um exemplo de movimentação no ambiente pode ser visualizado na Figura 2.5.

Com a movimentação descrita na Figura 2.5, o personagem torna-se livre a explorar o ambiente seguindo as regras de negócio do jogo, permitindo a interação com o ambiente, objetos posicionados no cenário ou outros jogadores. Dessa forma, a interação com estes elementos também são afetados pela área de interesse, na qual o personagem terá um raio limitante para cada tipo de interação no ambiente. Um exemplo de ambiente com personagens ( $P_1$  e  $P_2$ ), objetos ( $O_1$ ) e NPCs (NPC somente) pode ser visualizado na

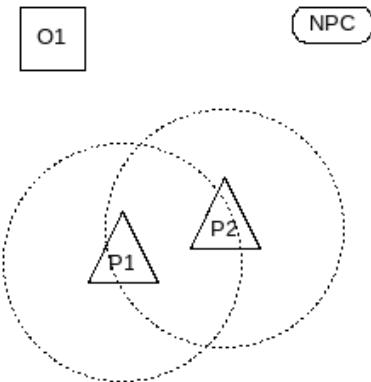
Figura 2.5: Personagens e os seus pontos de destino.



Fonte: O próprio autor.

Figura 2.6 (SALZ, 2016a).

Figura 2.6: Personagens, objetos e NPCs no ambiente.



Fonte: O próprio autor.

Na Figura 2.6 fica visível que a área de interesse pode ser um fator limitante conforme a operação com jogadores, objetos e NPCs (SALZ, 2016a). Dessa forma, o desempenho necessário para a execução dessas tarefas está tanto atrelado ao conjunto de algoritmos utilizado, quanto aos recursos computacionais gastos para tais operações.

Essas operações precisam de desempenho para não causar frustração ao jogador final (HOWARD et al., 2014). Nesse sentido, torna-se necessário conhecer os problemas computacionais conhecidos com relação aos serviços de jogos MMORPG.

## 2.4 Problemas em jogos MMORPG

Uma métrica popular para mensurar o desempenho de um serviço MMORPG é o número de conexões (HUANG; YE; CHENG, 2004) simultâneas suportadas. Em geral, caso o serviço ultrapasse o limite para o qual foi projetado, diversas falhas de conexão, problemas de lentidão ou dessincronização com o cliente podem ocorrer. Neste contexto, as

ocorrências comuns são (HUANG; YE; CHENG, 2004):

- **Longo tempo de resposta aos clientes:** implica em uma qualidade insatisfatória de jogabilidade ao usuário ou até mesmo impossibilitando o uso do serviço.
- **Dessincronização com os clientes:** realiza reversão na aplicação. Reversão é definida pela situação na qual uma requisição é solicitada ao servidor, um pré-processamento aparente é executado e essa requisição é negada, sendo necessário desfazer o pré-processamento aparente realizado ao cliente.
- **Problemas internos ao serviço:** podem estar relacionados a diversos outros erros internos de implementação ou a capacidade de recurso computacional (*e.g.*, sobrecarga no banco de dados, gerenciamento lento do espaço ou inconsistências dentro do jogo perante a regra de negócios).
- **Falha de conexão entre o cliente e o serviço:** causa a negação de serviço ao usuário final.

Existem algumas causas comuns para essas as ocorrências descritas (HUANG; YE; CHENG, 2004):

- **Baixo poder computacional do servidor:** poder computacional baixo para a qualidade de experiência de jogabilidade do usuário final desejada.
- **Complexidade de algoritmos:** o serviço usa algoritmos de alta complexidade ou regras de negócios que demandam por um algoritmo complexo.
- **Limitado pela própria arquitetura:** está limitado diretamente pelo número de conexões, não suportando a carga recebida.
- **Limitado pela rede:** a quantidade de requisições não é suportada pelo meio físico na qual a arquitetura está implantada.

Tais ocorrências estão diretamente correlacionadas a carga a qual tais serviços estão submetidos e podem ser amenizadas utilizando técnicas de provisionamento de recursos e balanceamento de carga (HUANG; YE; CHENG, 2004), mas não suficiente para eliminar tais ocorrências.

A área de desenvolvimento web compartilha várias ocorrências comuns geradas por sobrecarga do serviço (KHAZAEI et al., 2016). Em desenvolvimento web é comum utilizar a abordagem de microsserviços para resolver o problema de sobrecarga, modularizando o funcionamento em módulos menores. Da mesma forma, faz sentido modularizar um serviço MMORPG em microsserviços para suportar cargas maiores e diminuir o custo de manutenção (VILLAMIZAR et al., 2016).

Do ponto de vista da arquitetura de computadores, as operações existentes em um jogo MMORPG seguem um padrão de interação com o mundo, criar, excluir ou manipular objetos deste mundo. Para suprir o desenvolvimento de tais sistemas, se faz necessário compreender os padrões de desenvolvimento de tais arquiteturas na qual suprem as operações básicas de interação com o mundo.

#### 2.4.1 Arquitetura de Clientes MMORPG

A arquitetura de um cliente MMORPG é um aspecto fundamental, mas não único, para o sucesso de um jogo deste gênero. O seu funcionamento é totalmente visível ao usuário final e tem o principal objetivo de exibir o estado do mundo de forma gráfica ao usuário (SALZ, 2016a). Um exemplo de cliente MMORPG é o jogo Sandbox-Interactive Albion<sup>11</sup>, que pode ser visualizado na Figura 2.7.

A Figura 2.7 representa na prática, como será exibido ao usuário final o ambiente do jogo. O modelo teórico apresentado referente a esta figura pode ser visualizado na Figura 2.3.

Do ponto de vista de computação gráfica, um cenário 3D pode ser visualizado como uma árvore. Utilizar árvores para descrever um cenário ajuda tanto no formato de armazenamento em disco para leitura facilitada (*e.g.*, *JavaScript Object Notation* (JSON), *Extensible Markup Language* (XML), etc.), operações de inserção e exclusão, organização do projeto e redução da complexidade utilizando transformações lineares em sistemas gráficos como *OpenGL* e *DirectX* (LENGYEL, 2011). Esse modelo é amplamente utilizado em motores gráficos, na qual pode ser encontrado em motores gráficos populares como *Godot*, *Unity3D* e *Unreal 3*. A Figura 2.8 ilustra um exemplo, na qual exibe a árvore de um cenário na *Integrated Development Environment* (IDE) do motor gráfico *Godot*.

---

<sup>11</sup>Sandbox-Interactive Albion: <https://albiononline.com/en/home>

Figura 2.7: Exemplo de Cliente MMORPG (Sandbox-Interactive Albion).



Fonte: (SALZ, 2016a)

Figura 2.8: *Scene tree view* no motor gráfico Godot.

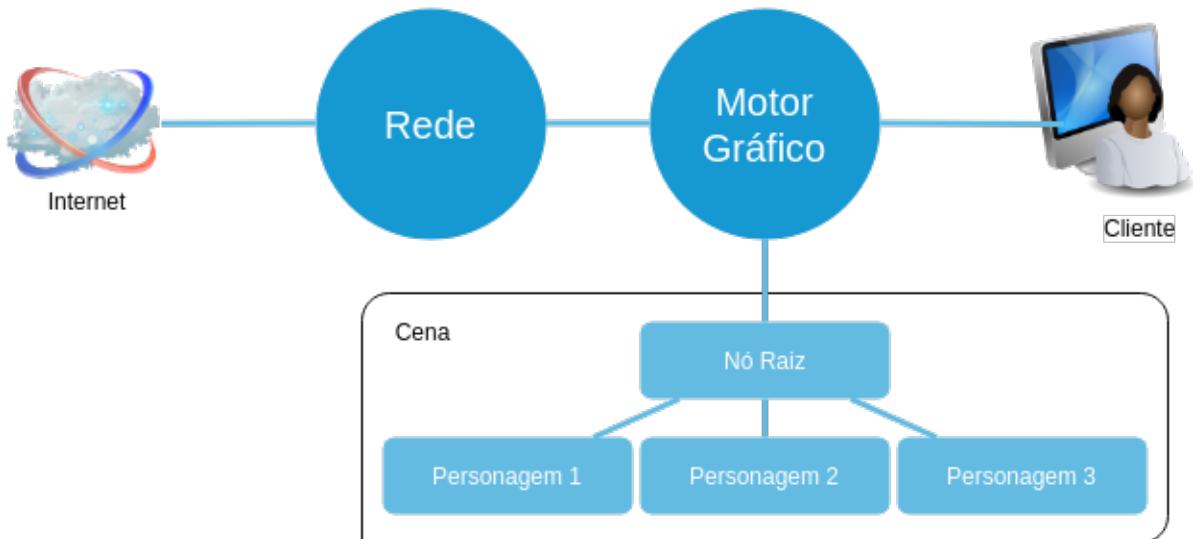
Fonte: O próprio autor.

Dentro dessa árvore, cada nodo tem uma funcionalidade específica. Essas funcionalidades variam a cada motor gráfico, podendo existir nodos focados à parte física, renderização ou controles (LINIETSKY, 2018). Em um jogo MMORPG, o serviço MMORPG será responsável por enviar atualizações dos parâmetros aos nodos frequentemente e o cliente será responsável por realizar chamadas remotas a fim de descrever as ações a qual o jogador aplicou sobre seu personagem (Exit Games, 2017).

Do ponto de vista da rede de computadores, a arquitetura de um cliente de jogo MMORPG deve suportar consultas e chamadas de métodos remotos em um serviço (SALZ, 2016a). Um cliente para um jogo MMORPG pode seguir o estilo de arquitetura *Representational State Transfer* (REST), porém não obrigatoriamente sobre o protocolo *Hypertext Transfer Protocol* (HTTP), mas sim usando algum protocolo RPC sobre o protocolo *Transmission Control Protocol* (TCP) ou *User Datagram Protocol* (UDP) (SALZ, 2016a; CLARKE-WILLSON, 2017). Essa comunicação é realizada pelo módulo *Network*, presente em um cliente MMORPG.

O módulo de *Network* implementado em um cliente de jogo MMORPG é responsável por realizar as requisições RPC conforme as ações solicitadas pelo jogador ao serviço, além de aplicar os parâmetros na *Scene Tree* ou chamar métodos remotos no cliente por ordens do serviço. Além disso, o módulo *Network* possui uma *Thread* dedicada ao gerenciamento de entrada e saída em relação ao serviço (SALZ, 2016a). Os principais módulos podem ser observados na Figura 2.9.

Figura 2.9: Modelo de um cliente genérico.



Adaptado de: (ZELESKO; CHERITON, 1996; FARBER, 2002)

A Figura 2.9 refere-se a uma visão macro de um cliente MMORPG, na qual é possível ver o ator *jogador* o qual pode executar ações sobre seu personagem por meio da *engine*. Por sua vez, existe uma entrada de dados a mais comparado a esquemas de jogos *offline*. Neste caso, o módulo *Network* será igualmente uma entrada de dados, a qual poderá manipular a cena do motor gráfico (FARBER, 2002). Para facilitar o desenvolvimento, a aplicação de cliente é dividida em diversos módulos, entretanto são relevantes ao atual trabalho (SALZ, 2016a):

- **Engine:** É o conjunto que aplicará regras sobre os objetos na *Scene Tree*, receberá entradas do usuário e exibirá a *Scene Tree* de forma imersiva. Unity3D<sup>12</sup> e GodotEngine<sup>13</sup> são exemplos de *engines*.
- **Network:** É o módulo responsável pela comunicação entre o serviço e o cliente, a fim de requisitar chamadas de métodos ou obter informações do servidor para sincronizar os estados de jogo.

Utilizando esses dois módulos é possível sincronizar os estados de jogo e exibi-los ao jogador. Entretanto, faz-se necessário compreender o funcionamento do serviço a fim de escolher um protocolo padrão para essa sincronização.

#### 2.4.2 Arquitetura de Microsserviços

Entende-se por microsserviço, aplicações que executam operações menores de um macrosserviço, da melhor forma possível (CLARKE-WILLSON, 2017; NEWMAN, 2015). O objetivo de uma arquitetura de microsserviços é funcionar separadamente de forma autônoma, contendo baixo acoplamento (NEWMAN, 2015). Seu funcionamento deve ser desenhado para permitir alinhamentos de alta coesão e baixo acoplamento entre os demais microsserviços existentes em um macrosserviço (ACEVEDO; JORGE; PATIÑO, 2017).

Arquiteturas de microsserviços iniciam uma nova linha de desenvolvimento de aplicações preparadas para executar sobre nuvens computacionais, promovendo maior flexibilidade, escalabilidade, gerenciamento e desempenho, sendo a principal escolha de arquitetura de grandes empresas como Amazon, Netflix e LinkedIn (KHAZAEI et al., 2016; VILLAMIZAR et al., 2016). Um microsserviço é definido pelas seguintes características (ACEVEDO; JORGE; PATIÑO, 2017):

- Deve possibilitar a implementação como uma peça individual do macrosserviço.
- Deve funcionar individualmente.
- Cada serviço deve ter uma interface. Essa interface deve ser o suficiente para utilizar o microsserviço.

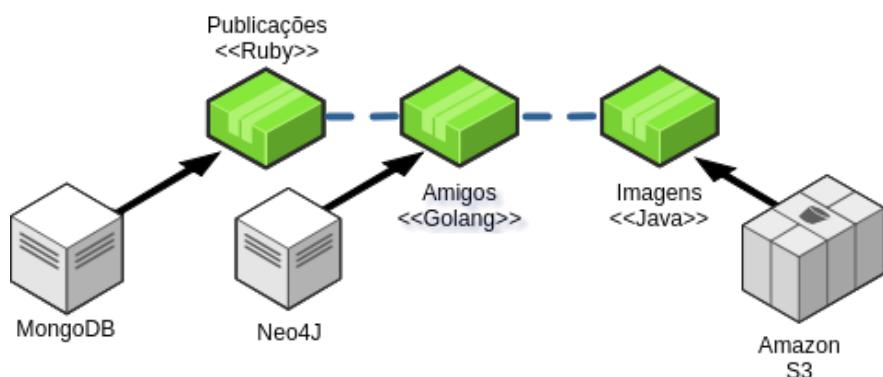
<sup>12</sup>Unity3D: <https://www.unity3d.com>

<sup>13</sup>GodotEngine: <https://www.godotengine.org>

- A interface deve estar disponível na rede para chamada de processamento remoto ou consulta de dados.
- O serviço pode ser utilizado por qualquer linguagem de programação e/ou plataforma.
- O serviço deve executar com as dependências mínimas.
- Ao agregar vários microsserviços, o macrosserviço resultante poderá prover funcionalidades complexas.

O microsserviço deverá ser uma entidade separada. A entidade deve ser implantada sobre um sistema isolado (*e.g.*, Docker<sup>14</sup>, *Virtual Machines* (VMs), *etc.*). Toda a comunicação entre os microsserviços de um macrosserviço será executada sobre a rede, a fim de reforçar a separação entre cada serviço. As chamadas pela rede com o cliente ou entre os microsserviços será executada através de uma *Application Programming Interface* (API), permitindo a liberdade de tecnologia em que cada microsserviço será implementado (NEWMAN, 2015). Isso permite que o sistema suporte tecnologias distintas que melhor resolvam os problemas relacionados ao contexto deste microsserviço. Isso pode ser visualizado na Figura 2.10.

Figura 2.10: Microsserviços podem ter diferentes tecnologias.



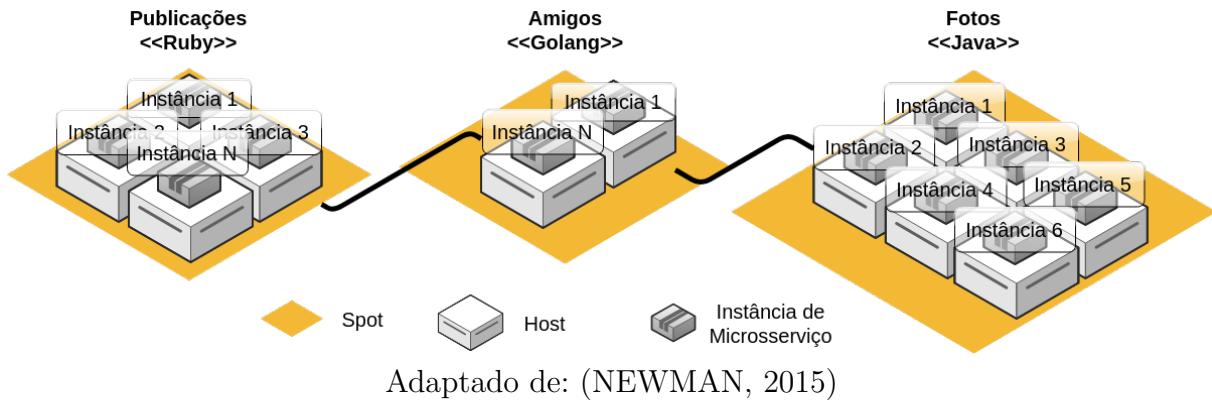
Adaptado de: (NEWMAN, 2015)

Uma arquitetura de microsserviços é escalável, como visível na Figura 2.11. A arquitetura permite o aumento do número de microsserviços sob demanda para suprir a necessidade de escalabilidade. Este modelo computacional obtém maior desempenho, principalmente se executar sobre plataformas de computação elástica, na qual o orques-

<sup>14</sup>Docker: <https://www.docker.com/>

tradutor do macrosserviço pode aumentar o número de instâncias conforme a necessidade de requisições (NADAREISHVILI et al., 2016).

Figura 2.11: Microsserviços são escaláveis.



Microsserviços desenvolvidos para web utilizam arquitetura REST baseado sobre o protocolo HTTP. É uma boa prática utilizar o corpo com conteúdo da requisição e resposta no formato JSON nas chamadas a uma API de microsserviço web (NADAREISHVILI et al., 2016). Entretanto, não é uma prática comum para um serviço MMORPG utilizar o protocolo HTTP pela sua elevada carga administrativa na requisição (HUANG; YE; CHENG, 2004). Por esse motivo, torna-se relevante compreender a composição de uma arquitetura com microsserviços para MMORPG, comparando-a a microsserviços web.

### 2.4.3 Microsserviços para jogos MMORPG

A fim de otimizar o custo operacional das arquiteturas de microsserviços de jogos MMORPG, é incomum a utilização de protocolos *Web* em tais arquiteturas. Por esse motivo, a seção atual mostrará o funcionamento básico do protocolo RPC e a sua utilização para atualização dos parâmetros na *Scene Tree* e o modelo REST (SALZ, 2016a).

Em engenharia de software, é comum a utilização de arquiteturas *Model-View-Controller* (MVC) a fim de organizar o código fonte e prover agilidade de desenvolvimento (CHADWICK; SNYDER; PANDA, 2012; THOMPSON, 2008). A separação de um serviço MMORPG pode ser dada seguindo este padrão de projeto, dividindo-se em três camadas (HUANG; CHEN, 2010):

1. *Model*: Representa qualquer dado presente no jogo (*e.g.*, itens, personagens, NPCs, objetivos, etc.).

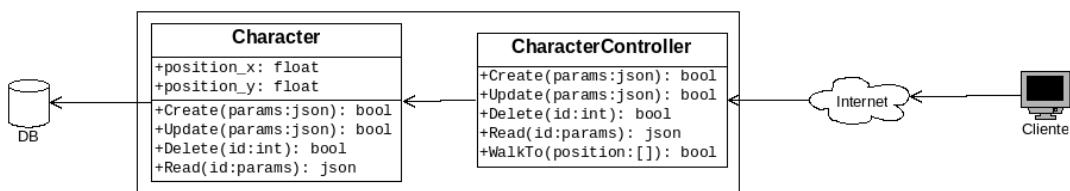
2. *View*: Representa o modo a qual estes dados serão exibidos, do ponto de vista de redes (*e.g.*, O mapa será exibido somente na área de interesse do jogador, no formato JSON).
3. *Controller*: Representa as operações sobre modelos que serão requeridos pelos jogadores (*e.g.*, andar, pegar item, interagir com NPCs, etc.).

Dentro de um *Controller* é implementado operações a qual o cliente pode requirir através de chamadas RPC a fim de manipular ou obter o estado de *Models* da aplicação. Esses métodos padrões seguem o protocolo CRUD, contendo quatro métodos principais para complementar as consultas sobre os *Models* (CHADWICK; SNYDER; PANDA, 2012; THOMPSON, 2008):

1. *Create*: Representa a criação de um novo objeto no banco.
2. *Update*: Representa a atualização de um objeto no banco.
3. *Delete*: Representa a exclusão de um objeto no banco.
4. *Read*: Representa a consulta sobre este objeto no banco.

Para o padrão CRUD, faz-se necessário que os métodos *Read*, *Update* e *Delete* repassem o parâmetro de identificação do objeto a ser consultado (THOMPSON, 2008). Outros argumentos necessários nos métodos *Create* e *Update* são os atributos do objeto, além do seu retorno ser um valor booleano representando se a operação foi bem sucedida (CHADWICK; SNYDER; PANDA, 2012; THOMPSON, 2008). Entretanto, outros métodos mais apropriados ao funcionamento de um *Controller* podem existir. Como exemplo, pode-se visualizar uma interface CRUD na Figura 2.12.

Figura 2.12: Cliente pode realizar requisições CRUD ao serviço.



Fonte: Adaptado de (SALZ, 2016a).

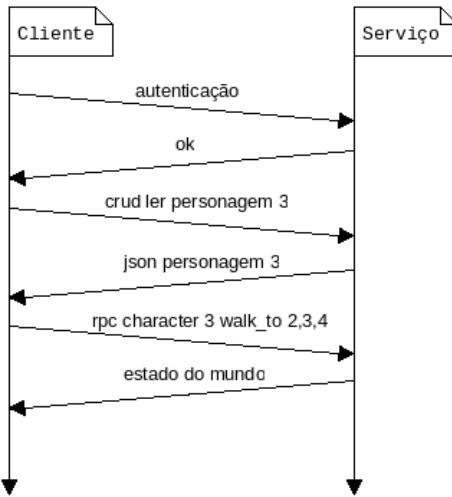
Essas operações são executadas utilizando protocolo RPC (SALZ, 2016a). As requisições de métodos remotos são realizadas entre dois processos distintos, a fim de

gerar uma computação distribuída (XEROX, 1976). Entretanto, a base do protocolo não é legível como em servidores web que utilizam JSON para transmissão de estrutura de dados, mas sim uma codificação binária nomeado *External Data Representation* (XDR) (Internet Society, 2006).

Uma técnica comum em jogos é a compressão de pacotes utilizando mapeamento *hash* de bytes (THOMPSON, 2008). Tanto o cliente quanto o serviço precisam ter a mesma estrutura de dados. Dessa forma, é possível trocar o nome das funções solicitadas em RPC por poucos bytes para transitar na rede. Já para operações CRUD, pode-se utilizar tanto requisições sobre o protocolo HTTP ou sobre um protocolo otimizado sobre TCP dependendo da necessidade de desempenho (THOMPSON, 2008).

Dado um serviço que não é implementado sobre uma arquitetura de microserviços, a utilização de dois protocolos irá complicar o gerenciamento de threads para responder de duas formas diferentes, entretando o seu esquema de estados ficará simplificado. Esta simplificação pode ser visualizada na figura 2.13.

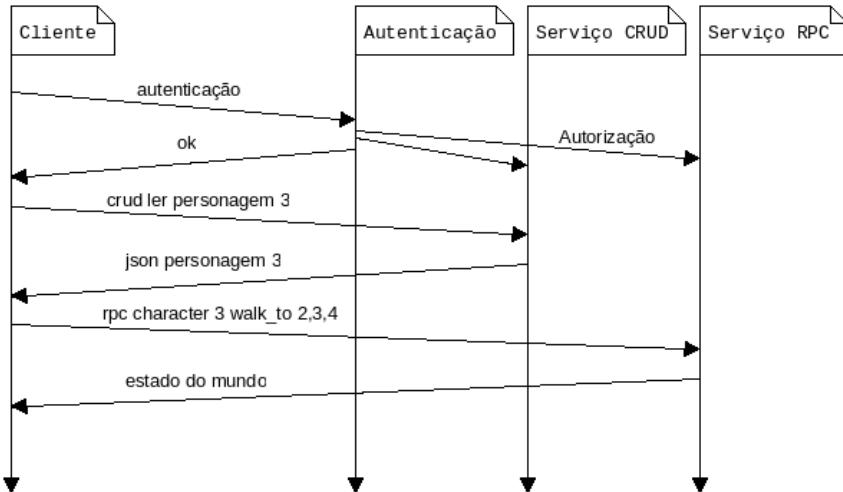
Figura 2.13: Diagrama de requisições entre serviço e cliente com operações CRUD e RPC em uma arquitetura monolítica.



Fonte: Adaptado de (THOMPSON, 2008)

Entretanto, diferente da Figura 2.13, a fim de simplificar a complexidade da implementação e gerenciamento de concorrência no serviço, pode-se implementar utilizando o paradigma de microserviços. Como relatado na Seção 2.4.2, uma arquitetura de microserviços permite múltiplas tecnologias, pois a comunicação entre todos os elementos de um microserviço será pela rede. Por esse motivo, é possível utilizar um serviço web para realizar operações CRUD e um serviço dedicado para realizar operações RPC. Essa arquitetura pode ser melhor visualizada na Figura 2.14.

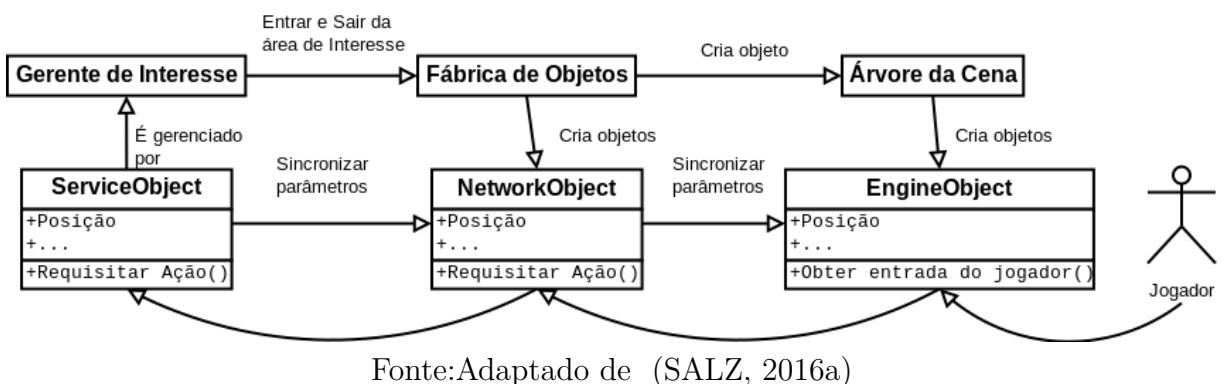
Figura 2.14: Diagrama de requisições entre serviço e cliente com operações CRUD e RPC em uma arquitetura de microserviços.



Fonte: O próprio autor.

Como resultado da integração entre Cliente, Serviço e Motor Gráfico, o resultado final obtido é descrito pelo diagrama presente na Figura 2.15. Tal integração está presente no jogo Sandbox-Interactive Albion<sup>15</sup> (SALZ, 2016a). Percebe-se, neste contexto, que o jogo deverá funcionar sem o motor gráfico, do ponto de vista de redes e estrutura de dados (SALZ, 2016a).

Figura 2.15: Diagrama de integração entre Cliente e Serviço, considerando a *engine Unity3D*.



A Figura 2.15 ilustra a separação da camada de renderização de objetos da árvore da cena, a camada de integração com o cliente e serviço (descrito anteriormente como módulo *Network*) e o serviço. Nesse sentido, a alteração entre clientes e serviços facilita um sistema de teste de carga e busca de erros automatizado, facilitando a manutenção e desenvolvimento incremental do serviço (SALZ, 2016a).

Utilizando estes padrões de projeto, algumas arquiteturas tornam-se popula-

<sup>15</sup>Sandbox-Interactive Albion: <https://albiononline.com/en/home>

res no desenvolvimento de jogos MMORPG. Para este fim, torna-se de interesse a este trabalho realizar um levantamento de algumas arquiteturas comuns em jogos massivos.

## 2.5 Arquiteturas MMORPG identificadas

Dentre as arquiteturas identificadas, as qualificadas dentro do paradigma de arquiteturas de microsserviços são as arquiteturas Rudy (Subseção 2.5.1), Salz (Subseção 2.5.2) e Willson (Subseção 2.5.3).

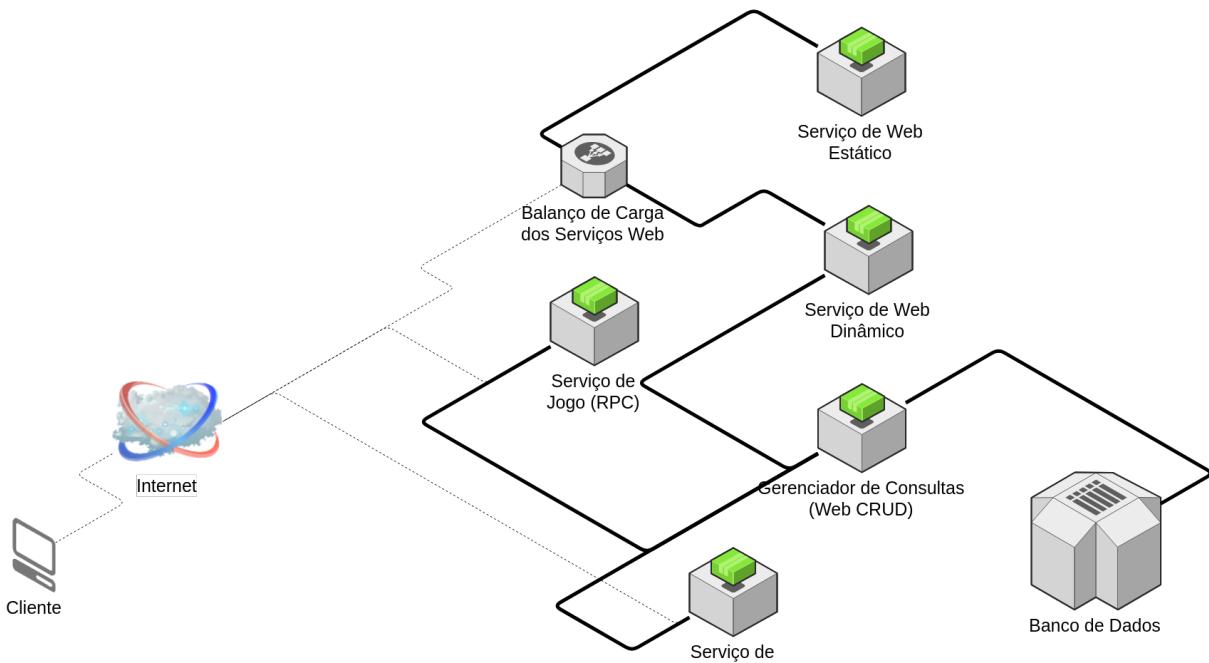
Em geral, as arquiteturas de forma genérica contém microsserviços *web* para *E Commerce*, operações CRUD através da web e distribuição de atualizações. Os microsserviços de gerenciamento de jogo, por sua vez, respondem através do protocolo TCP, podendo ser sobre RPC ou protocolos proprietários. A gerência de jogo é a principal mudança entre as arquiteturas, na qual contém abordagens de paralelismo diferentes:

- A arquitetura Rudy (Subseção 2.5.1) utiliza uma abordagem com um número menor de microsserviços, focado em manter serviços para consulta de dados de forma eficiente entre os serviços *web* e o serviços de gerenciamento de jogo.
- A arquitetura Salz (Subseção 2.5.2) aborda um modelo de paralelismo mais complexo comparado a arquitetura Rudy, utilizando diversos microsserviços para funções específicas das funcionalidades do jogo. Dessa forma, o ambiente do jogo torna-se escalável ao número de jogadores, porém a latência tende a aumentar.
- A arquitetura Willson (Subseção 2.5.3) utiliza um modelo intermediário, evitando a divisão em múltiplos serviços para o gerente de jogo e utilizando um modelo de paralelismo próximo a arquitetura Salz.

### 2.5.1 Arquitetura elaborada por Rudy

A arquitetura Rudy (RUDDY, 2011) tem como objetivo criar múltiplos mundos isolados, na qual cada microsserviço será responsável por um ambiente a qual não compartilha dados com os demais ambientes. Esta é uma característica importante para esta arquitetura, visto que o processamento de ações pelo serviço de jogo não precisa lidar com múltiplos processos (RUDDY, 2011). Esta arquitetura pode ser visualizada na Figura 2.16.

Figura 2.16: Arquitetura Rudy completa.



Adaptado de: (RUDDY, 2011).

No total, seis microsserviços distintos constam na arquitetura Rudy (Figura 2.16) para o seu funcionamento, não sendo necessário o microsserviço de pagamento (utilizado somente para regra de negócios). Os microsserviços que compõem a arquitetura são definidos pelas suas seguintes responsabilidades (RUDDY, 2011):

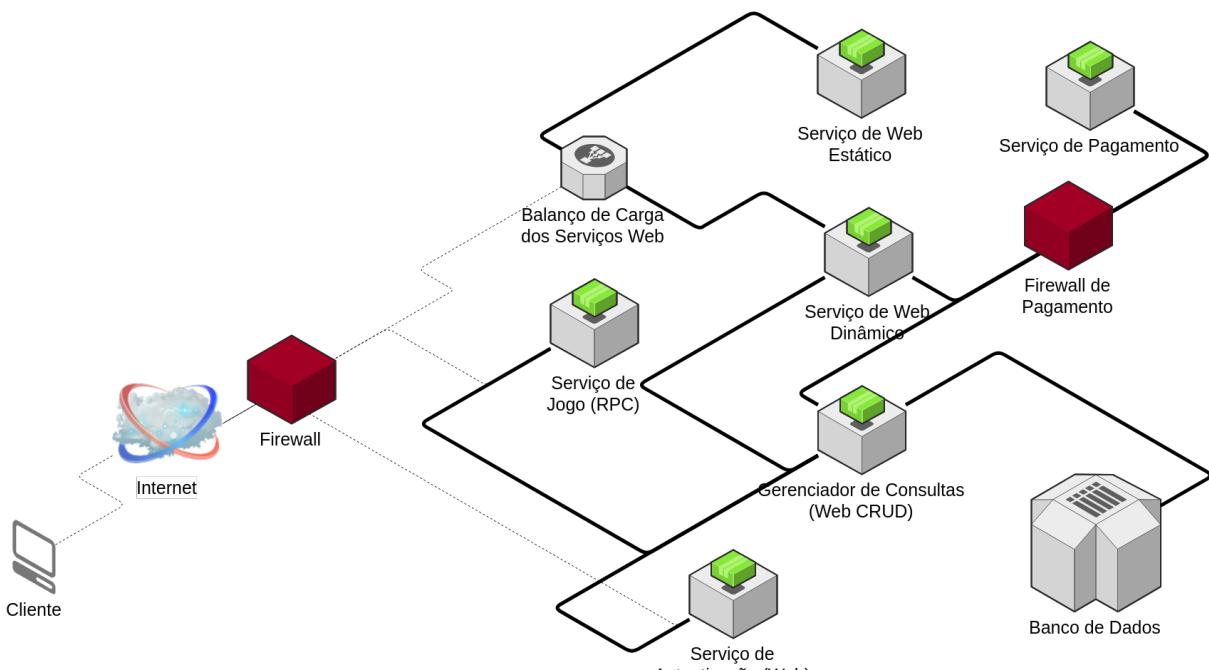
- Serviço de web estático:** Armazena documentos estáticos para o serviço web (*e.g.*, imagens, executáveis do jogo, páginas web fixas, etc). Responde sobre o protocolo HTTP.
- Serviço de web dinâmico:** Sistema web para cadastro de contas, guias, informações sobre atualizações, compras e demais demanda de páginas dinâmicas. Responde sobre o protocolo HTTP.
- Balanço de carga web:** Realiza a distribuição de carga sobre o *Serviço web estático* e o *Serviço web dinâmico*. Responde sobre o protocolo HTTP.
- Serviço de Jogo:** Gerencia um mundo inteiro, em um único serviço. Esta abordagem segregaria os jogadores em diversos canais, contendo um número máximo de conexões por canal. Cada canal opera sobre uma instância deste microsserviço. Este serviço opera sobre o protocolo RPC.

5. **Serviço de Autenticação:** Gerencia a autenticação das conexões ao *Serviço de jogo*. Este serviço opera sobre o protocolo RPC.

6. **Gerenciador de Consultas:** Realiza consultas em memória e disco, utilizando vários bancos de dados diferentes, simulando o uso de banco de dados distribuídos, algo complexo de ser implementado utilizando banco de dados SQL (*e.g.*, PostgreSQL<sup>16</sup>, MySQL<sup>17</sup>, etc). Este serviço opera sobre o protocolo HTTP.

No contexto do atual trabalho, o serviço de pagamento será ignorado, visto que ele não serve para o funcionamento básico do serviço. Dentre todos os microsserviços, o usuário só tem acesso ao serviço de balanço de carga pelo protocolo HTTP e o serviço de jogo sobre o protocolo RPC, tendo os demais serviços protegidos por um *firewall* (RUDDY, 2011). A proteção do *firewall* é aplicada no serviço de pagamento e no ponto de acesso ao serviço, podendo ser visualizada na Figura 2.17.

Figura 2.17: Arquitetura Rudy completa com *firewall*.



Adaptado de: (RUDDY, 2011).

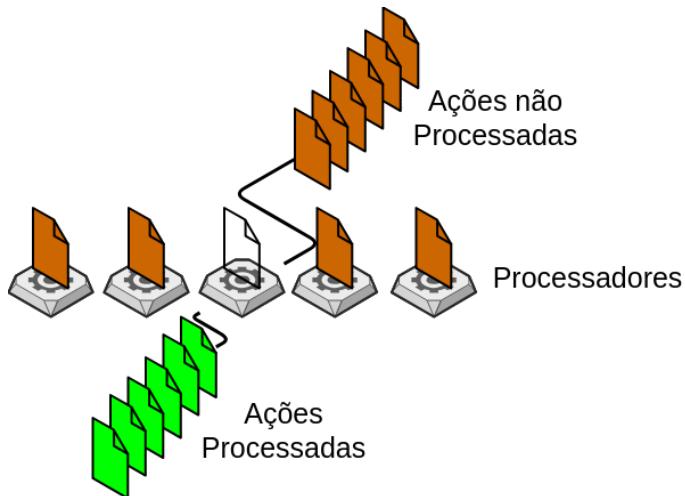
O funcionamento interno do serviço de jogo trabalha em rodadas, visando não penalizar usuários com baixa transferência de dados entre o cliente e o serviço. Cada cliente produz requisições para serem consumidas pelo ciclo de processamento do gerente de

<sup>16</sup>PostgreSQL: <https://www.postgresql.org>

<sup>17</sup>MySQL: <https://www.mysql.com/>

jogo. Entretanto, o serviço irá consumir de forma igualitária uma requisição de um jogador diferente, como uma fila (SALZ, 2016a; RUDDY, 2011). O modelo de processamento do gerente de ambiente pode ser visualizado na Figura 2.18.

Figura 2.18: Modelo de processos *Thread Pool*.



Adaptado de: (RUDDY, 2011; RINGLER, 2014).

O modelo de processamento do gerente de mundo, visualizado na Figura 2.18 trabalha com um padrão *Thread Pool* (RINGLER, 2014; RUDDY, 2011), executando a chamada de método remoto de cada jogador em uma fila, o qual prioriza executar as chamadas sem repetir a mesma conexão. Dessa forma, cada jogador pode executar somente um método concorrente, sem competir com os demais. Todas as requisições são enfileiradas no *buffer* de rede do serviço. Caso o cliente entre em sua vez de processamento, e nenhuma chamada remota esteja na fila, ele é pulado.

Um serviço que demanda atenção na arquitetura Rudy é o Gerenciador de Consultas, um serviço web que implementa uma camada sobre diversos bancos de dados a fim de prover variedade entre vários bancos de forma facilitada por requisições web, utilizando operações CRUD. Implementar esta camada garante uma padronização de acesso ao banco de dados, porém adiciona um possível gargalo a arquitetura (RUDDY, 2011). Os pontos positivos de utilizar esta camada de consultas na arquitetura são:

1. Não permite acesso direto ao banco de dados do serviço web e do serviço de jogo.
2. Permite maior manejo a migrações em tabelas e troca de tecnologias.
3. Define uma sintaxe estrita para consulta, via CRUD.
4. Permite acesso do banco a diversos serviços, sem gerenciar o banco.

5. Permite contar número de requisições e tempo das requisições.

Contudo, adicionar uma camada sobre os bancos de dados para gerenciamento tem pontos negativos (RUDDY, 2011).

1. Aumenta a complexidade de implementação, teste, administração e ponto de falha.
2. Adiciona limites como número de conexões, número de requisições, etc.

Entretanto, esta arquitetura não permite escalar um único ambiente para um número de jogadores simultâneos maior ao designado pelo hardware que hospeda o serviço. Por este motivo, as arquiteturas Salz e Willson tomam abordagens para subdividir o ambiente do jogo em mais serviços, podendo assim escalar um único ambiente para mais jogadores simultâneos.

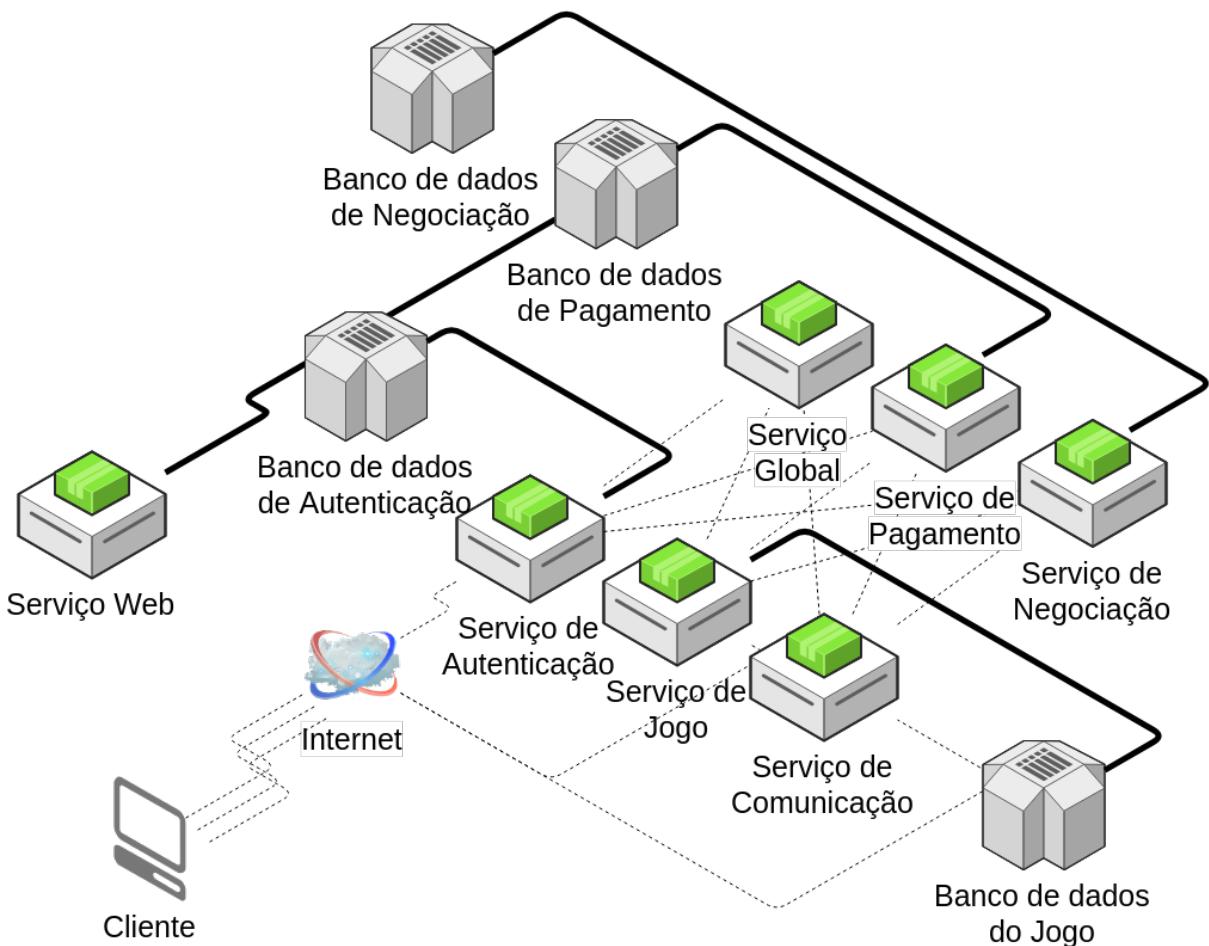
### 2.5.2 Arquitetura elaborada por Salz

A arquitetura elaborada por Salz (SALZ, 2016a), a qual pode ser visualizada na Figura 2.19 contém sete microsserviços especializados para seu funcionamento. Para o funcionamento adequado, o cliente necessita manter três conexões abertas com o servidor, sendo a conexão de jogo a com maior demanda de banda e a qual necessita o menor tempo de latência (SALZ, 2016a).

A Figura 2.19 exibe a arquitetura Salz de forma completa, na qual encontram-se quatro bancos de dados distintos, sendo eles o banco de *Pagamento*, *Negociação*, *Autenticação* e *Jogo*, sendo os bancos de Autenticação e Jogo com tecnologias *Not Only SQL* (NoSQL). Os demais bancos utilizam tecnologia *Structured Query Language* (SQL). Referente aos microsserviços que compõem a arquitetura, tem-se os seguintes elementos (SALZ, 2016b):

1. **Serviço de Comunicação:** Gerencia a troca de mensagens entre os jogadores. Opera sobre o protocolo RPC.
2. **Serviço de Autenticação:** Recepiona e gerencia as conexões dos clientes entre os demais microsserviços. Opera sobre o protocolo HTTP.
3. **Serviço de Jogo:** Gerencia o estado do mundo, referente a um *chunk* do ambiente. Opera sobre o protocolo RPC.

Figura 2.19: Arquitetura Salz.



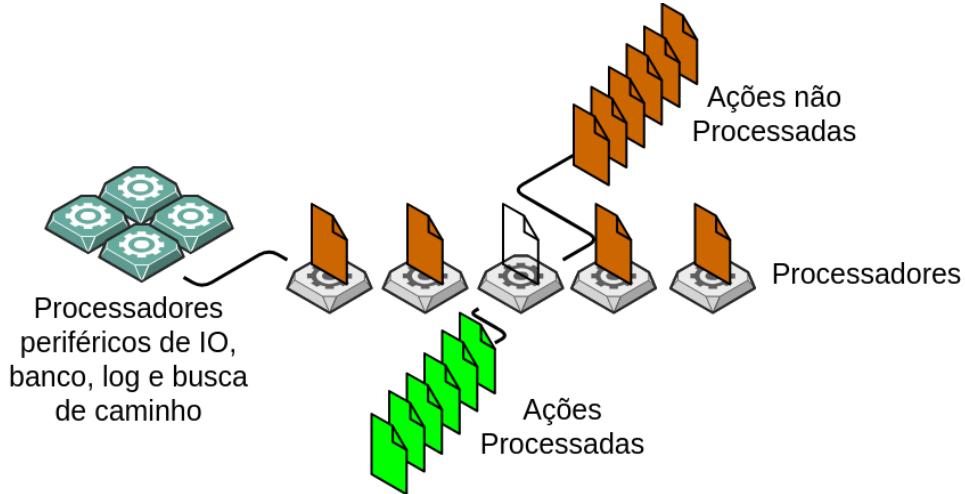
Adaptado de: (SALZ, 2016a).

4. **Serviço Global:** Gerencia operações globais (*e.g.*, interações entre grupos, procedimentos recorrentes globais, *etc.*). Opera sobre HTTP.
5. **Serviço de Pagamento:** Efetua operações bancárias com serviços externos de pagamento e gerencia o estado de pagamento das contas. Opera sobre o protocolo HTTP.
6. **Serviço de Negociação:** Opera como um serviço de leilão para itens do jogo. Opera sobre o protocolo HTTP.

Os microsserviços que compõem a arquitetura Salz utilizam em grande parte serviços *web*, utilizando somente o protocolo RPC para o serviço de jogo, autenticação e comunicação. No serviço de jogo, tanto o cliente quanto o serviço podem invocar métodos remotos através do protocolo RPC, tendo como padrão métodos com retorno nulo (SALZ, 2016b; Exit Games, 2018). Esta abordagem garante que o usuário não fique esperando pelo retorno do serviço para continuar a lógica do jogo (FARBER, 2002). Para este

funcionamento, o modelo de processamento paralelo deste serviço possui mais regras, a qual não são abordadas pela arquitetura Rudy (Figura 2.16). O modelo de paralelismo do serviço de jogo pode ser visualizado na Figura 2.20.

Figura 2.20: Modelo de paralelismo do serviço de jogo na arquitetura Salz.



Adaptado de: (SALZ, 2016b; CLARKE-WILLSON, 2013).

O microsserviço de jogo executa a lógica do jogo, visível na Figura 2.20, para uma única área em um único processo. Esta decisão é dada por conta da interação entre objetos ser complicada de executar em paralelo, visto o gerenciamento do custo de gerenciamento de semáforos necessários, caso execute estas ações em paralelo (SALZ, 2016b).

Outra facilidade de implementar um modelo com um único processo para as interações entre objetos é facilitar o não manejo de objetos a qual não estão no campo de visão de todos os jogadores do serviço, realizando estas operações somente quando necessário (SALZ, 2016a; SALZ, 2016b).

As entradas e saídas de mensagens(*e.g.*, rede, Banco de dados, *etc.*) e busca de caminho é executado por processos de trabalho separado do principal, utilizando a técnica de *Thread Pool* (SALZ, 2016b; SALZ, 2016a; RINGLER, 2014). Todos os demais microsserviços operam sobre múltiplos processos, a partir do processo de conexão TCP ao serviço (SALZ, 2016b). Diferente da arquitetura Rudy(Seção 2.5.1) a qual prioriza a resolução de requisições, evitando a executar métodos consecutivos da mesma conexão, na arquitetura Salz (Figura 2.19) as chamadas de processo remoto são executadas em ordem *First In First out* (FIFO) (SALZ, 2016b). Desta forma, quanto melhor a conexão com o serviço, mais requisições um cliente poderá executar. Nesse sentido, uma conexão que

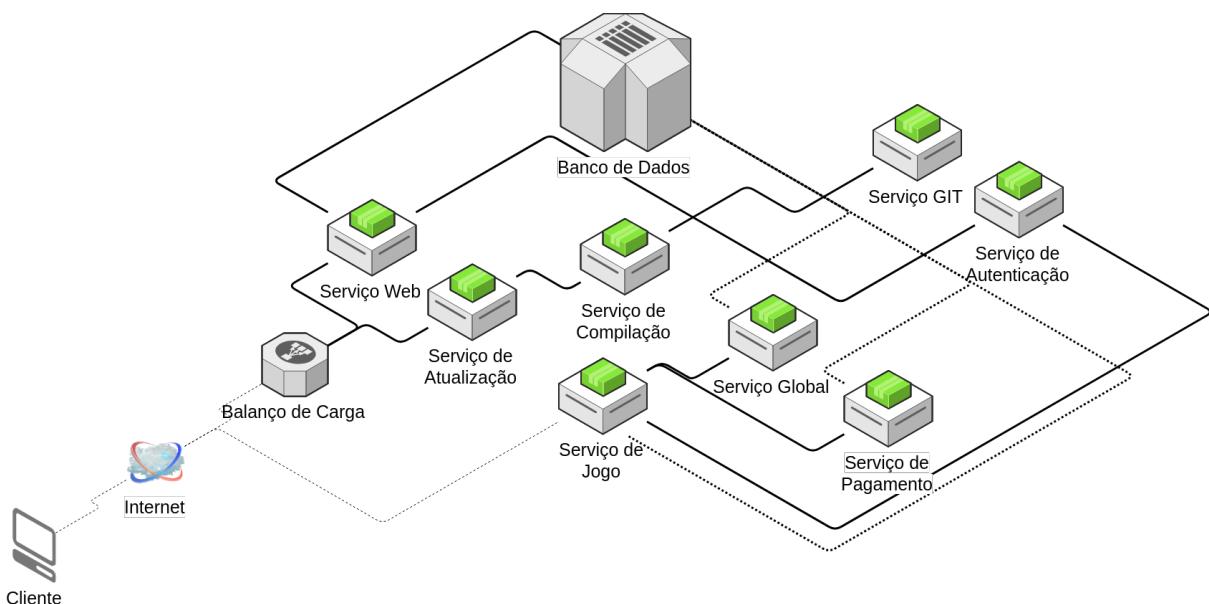
transfere mais requisições terá mais vantagem sobre uma conexão a qual não transfere tantas requisições, executando mais operações por segundo.

Na arquitetura Salz, são necessárias três conexões TCP com o servidor, de forma contínua (SALZ, 2016a). O custo desta operação é alto, e por isso nem sempre é viável utilizar esta abordagem, principalmente para jogos nos quais a demanda de banda seja menor. Para evitar esta decisão, da mesma forma é notória a abordagem utilizada pela arquitetura Willson.

### 2.5.3 Arquitetura elaborada por Willson

A arquitetura elaborada por Clarke-Willson leva como principal característica a preocupação da disponibilização de atualizações aos clientes (CLARKE-WILLSON, 2013). Essas atualizações são versionadas a todos os microsserviços utilizando sistemas de versionamento de código fonte (CLARKE-WILLSON, 2017; CLARKE-WILLSON, 2013). Por este motivo, a sua arquitetura inclusive compreende microsserviços de armazenamento de arquivos sobre protocolo HTTP para atualização dos clientes (CLARKE-WILLSON, 2017). Uma outra característica do serviço de jogo é a utilização de uma única conexão TCP por cliente. Essa arquitetura pode ser visualizada na Figura 2.21.

Figura 2.21: Arquitetura Willson.



Fonte: (CLARKE-WILLSON, 2017).

A arquitetura Willson, exibido na Figura 2.21, mostra um gradual entre a arquitetura Rudy (Figura 2.16) e a arquitetura Salz (Figura 2.19), propondo uma ar-

quitetura híbrida, na qual divide funcionalidades em outros microsserviços, porém ainda mantém diversas funcionalidade junto ao serviço de jogo evitando consumo de rede (SALZ, 2016a; CLARKE-WILLSON, 2013). Essa abordagem garante menor latência de resposta, porém terá maior consumo de recursos da mesma máquina hospedeira do serviço (CLARKE-WILLSON, 2013). Os microsserviços que compõem a arquitetura Willson são (CLARKE-WILLSON, 2013; CLARKE-WILLSON, 2017):

1. **Serviço web:** Exibe informações do jogo, oferece operações CRUD para cadastro de usuários e o sistema de pagamento. Opera sobre o protocolo HTTP.
2. **Serviço de Atualização:** Integrado junto ao processo de desenvolvimento, fornecendo versionamento do cliente por um sistema web. Este microsserviço utiliza serviços internos ao desenvolvimento da aplicação. Opera sobre o protocolo HTTP.
3. **Serviço de Autenticação:** Gerencia a autenticação dos jogadores através de um serviço web. Também gerencia ações sociais (*e.g.* sistema de amigos, grupos, nações, comércio, *etc*). Opera sobre o protocolo HTTP.
4. **Serviço de Balanceamento de carga:** Gerencia a carga entre os serviços web da arquitetura. Opera sobre o protocolo HTTP.
5. **Serviço de Jogo:** Processa a lógica de jogo. Cada instância deste microsserviço gerencia um *chunk* do ambiente do jogo. Opera sobre o protocolo RPC.
6. **Serviços Globais:** Processa rotinas globais do jogo, a qual não dependem do posicionamento do jogador. Opera sobre o serviço RPC.

O serviço de jogo da arquitetura Willson é comparada ao serviço similar na arquitetura Salz, definido no Subcapítulo 2.5.2, entretanto utiliza a técnica de *thread pool* com valor de processos da fila de processadores fixo conforme o *hardware* hospedeiro do serviço (CLARKE-WILLSON, 2017). Para modelos de paralelismo, pode-se utilizar os seguintes números de processos paralelos (CLARKE-WILLSON, 2013):

1. O dobro de número de núcleos: exige maior carga do serviço por troca de contexto entre os processos.
2. O número exato de núcleos mais n: O serviço perderá tempo de processamento, trocando de contexto para outro processo, porém de forma mais sutil ao dobro do número de núcleos de processamento.

3. O número de núcleos: exigindo um número menor de carga de contexto, dividindo somente com o sistema operacional.
4. O número de núcleos menos um: liberando um núcleo para o sistema operacional.

O número padrão de processos paralelos para processamento de requisições é o número de núcleos menos um, visando evitar a troca de contexto com o sistema operacional (CLARKE-WILLSON, 2013). Essa troca de contexto trás variação na latência da resposta do serviço, a qual não tem controle pelo próprio serviço. Nesse sentido, a melhor escolha é o número de núcleos menos um, escolhido após um teste de *stress*, definido no Subcapítulo 2.5.3.

Entretanto, não foi encontrado análises públicas sobre as arquiteturas Rudy, Salz e Willson, com relação ao uso de recursos dessas arquiteturas. Nesse sentido, o atual trabalho define o seu problema sobre o consumo de recursos em buscar uma análise sobre o comportamento das arquiteturas referenciadas.

## 2.6 Definição do Problema

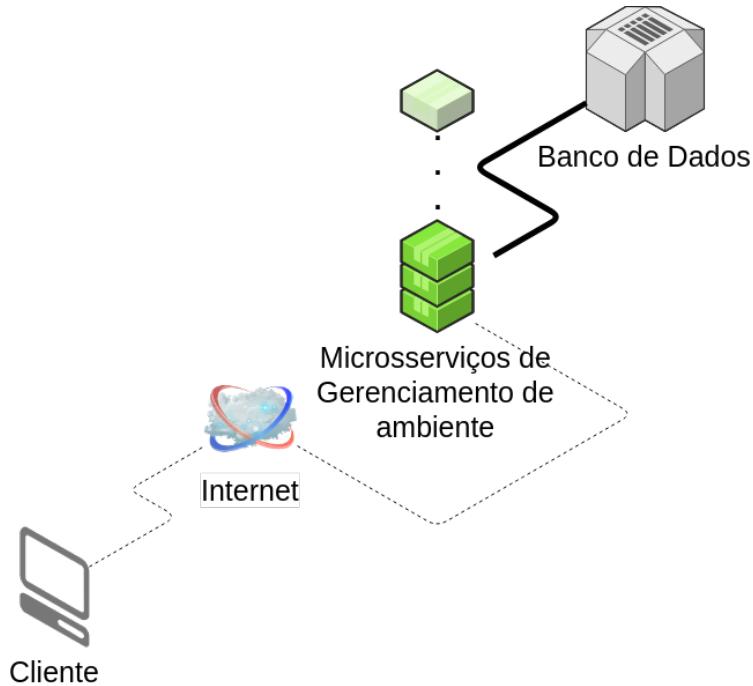
A escolha de uma arquitetura para serviços MMORPG que não suportem uma elevada carga de trabalho podem ser um eventual problema na escalabilidade do negócio de empresas que forneçam tais serviços. Em alguns serviços, a divisão da carga é dada pela área de interesse, dividindo o ambiente em pedaços a fim de diminuir a carga no serviço. Porém, locais populares no ambiente do jogo ainda são vulneráveis a disponibilidade de serviço (HUANG; YE; CHENG, 2004).

Nesse sentido, arquiteturas de microsserviços surgiram com o objetivo de aumentar a disponibilidade e viabilizar produtos de demanda massiva. Tais arquiteturas dividem o seu funcionamento em módulos menores a fim de proporcionar maior demanda utilizando uma abordagem distribuída. Assim, o custo de atender uma alta demanda de conexões pode consumir uma quantia de recursos computacionais ou uma qualidade insatisfatória de serviço, inviabilizando a sua utilização no mercado.

Um exemplo de implementação de arquitetura de um jogo MMORPG pode ser visualizado na Figura 2.22, no qual cada cliente deve conectar em um microsserviço de gerenciamento de mundo para receber as atualizações da região e submeter seus co-

mandos. Nessa arquitetura, é necessário levar em conta o funcionamento, método de compartilhamento dos dados e abordagem para processamento do ambiente do jogo a fim de descrever a sua escalabilidade e qualidade de serviço.

Figura 2.22: Arquitetura de um jogo MMORPG genérico.



Fonte: O próprio autor

A arquitetura genérica descrita na Figura 2.22 também precisa corresponder as demandas necessárias do *GameDesign* do próprio jogo. Neste caso, só um tipo de microsserviço é visível nesta rede, porém poderiam existir microsserviços responsáveis pela parte social, comercial e estatísticas, aumentando o grau de complexidade da arquitetura. Em relação aos recursos utilizados por uma arquitetura, este trabalho foca na análise das arquiteturas de microsserviços descritas na literatura a fim de analisar o seu comportamento e desempenho. Porém, não foi encontrado nenhuma análise das arquiteturas de microsserviços propostas por Rudy, Salz e Willson, tornando a escolha de um modelo de arquitetura uma tarefa complexa a qual demande testes de tentativa e erro ou análises particulares para viabilizar o funcionamento destes sistemas. Dessa forma, este trabalho tem como principal objetivo guiar, utilizando análises, futuras escolhas de modelos a qual se adequem melhor a realidade de futuros projetos de jogos MMORPG baseados em microsserviços.

## 2.7 Considerações parciais

Este capítulo conceituou jogos eletrônicos, gênero de jogo e especificou as características de um jogo MMORPG. Após apresentar sobre o gênero de jogo abordado, detalha-se a sua jogabilidade, problemas relevantes a este gênero do ponto de vista de rede de computadores e por fim sobre técnicas e abordagens populares acerca do desenvolvimento destes serviços, em específico sobre o paradigma de arquitetura de microsserviços.

O desenvolvimento de arquiteturas de microsserviços para jogos MMORPG é uma tarefa multidisciplinar a qual pequenas variações de protocolo, algoritmos ou microsserviços dentre os modelos impacta diretamente no consumo de recurso destas arquiteturas. Nesse sentido, o atual capítulo trouxe inicialmente um levantamento teórico sobre as principais características deste gênero de jogo, características das principais camadas de cliente, servidor e serviço e por sua vez mostrou abordagens viáveis de desenvolvimento para ambas as camadas. Por fim trouxe uma introdução ao paradigma de arquiteturas de microsserviços, abordando em seguida a sua aplicação para jogos MMORPG.

Após a introdução técnica necessária para entendimento das arquiteturas, foi realizado um levantamento de arquiteturas de microsserviços da literatura, descrevendo sua funcionalidade, topologia e por fim protocolos utilizados. Entretanto, não foi encontrado trabalhos correlacionados as arquiteturas encontradas, dessa forma encontrando um dos pontos de apoio científico deste trabalho.

A literatura aborda previsibilidade de carga, análise de disponibilidade e uso de recurso de tais arquiteturas (a fim de guiar escolhas na etapa de *Game Design* do produto e viabilizar o comércio do mesmo), relatados na Seção 3. Torna-se de interesse para projetistas de desenvolvimento de jogos MMORPG analisar o impacto e uso de recursos computacionais ao implantar uma arquitetura de microsserviços MMORPG visando melhorar a disponibilidade de tais jogos. Nesse sentido, se faz necessário realizar um levantamento bibliográfico a cerca dos temas de jogos MMORPG e arquiteturas de microsserviços com a finalidade de comprovar a utilidade da proposta do atual trabalho.

### 3 Trabalhos Relacionados

Para nortear o desenvolvimento da análise de microsserviços utilizados em jogos MMORPG proposto no atual trabalho, essa seção apresenta outros trabalhos que têm o escopo ou objetivo similar, no qual monitoraram e analisaram serviços de jogos MMORPG ou arquiteturas de microsserviços. Ao apresentar estes trabalhos, busca-se apresentar o contexto e objetivo, e então aprofundar em características dos trabalhos, métricas utilizadas e ferramentas que auxiliaram nas análises.

Para encontrar os trabalhos relacionados, foi efetuada uma busca pelos termos MMORPG ou *microservices*. Entretanto, os dois termos dificilmente se correlacionam nos meios de busca disponíveis para a elaboração deste TCC. Nesse sentido, os trabalhos relacionados abordam arquiteturas de jogos MMORPG ou arquiteturas de microsserviços.

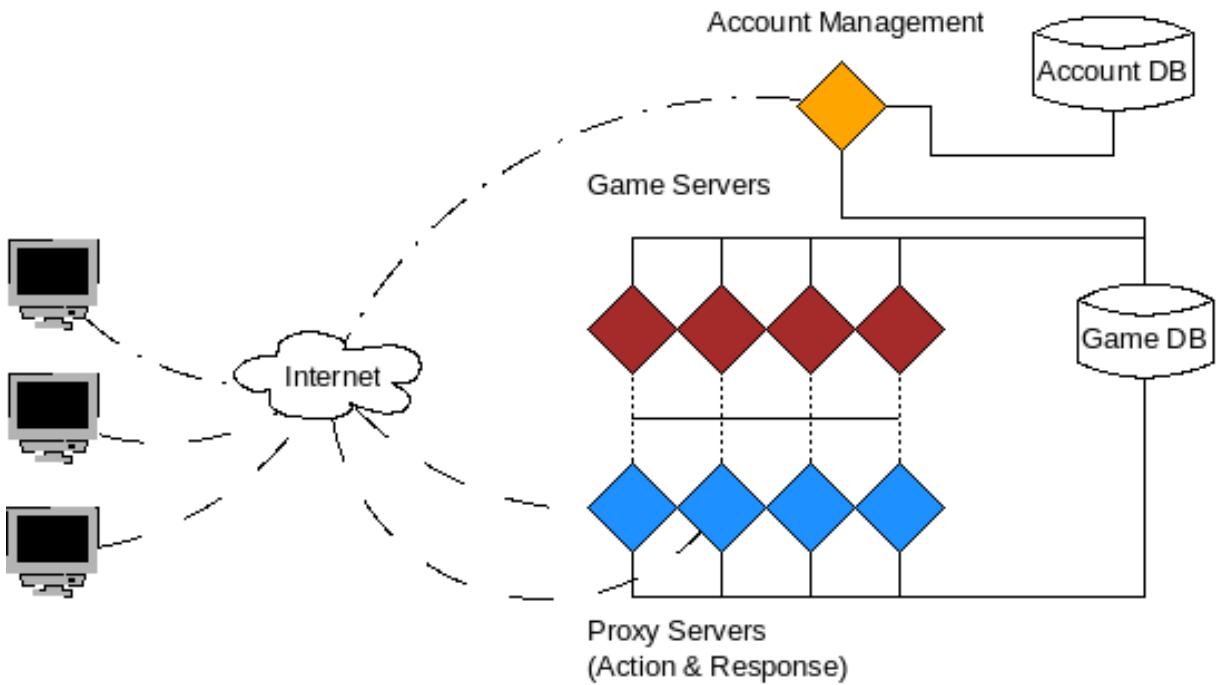
Como critério de análise, foi observado em questão qual a classificação em que o trabalho encontra-se, entre previsão de carga ou análise de arquitetura, e quais métricas são utilizadas na análise.

#### 3.1 Huang et al. (2004)

O trabalho de (HUANG; YE; CHENG, 2004) investiga a relação entre os recursos utilizados e o número de conexões presentes em um serviço MMORPG distribuído. Neste trabalho é relatado que a infraestrutura utiliza três serviços: Um *Game Server* sobre protocolo TCP, um *Proxy Server* também sobre protocolo TCP, e um servidor web para autenticação que executa sobre uma interface HTTP. O foco de análise é o *Proxy Server*, um serviço especificado em receber requisições e repassar atualizações da área de interesse destes jogadores, e o *Game Server*, um serviço especificado para consumir as requisições realizadas pelo jogador.

A infraestrutura do servidor de jogo contém um *Proxy Server Farm* utilizando o algoritmo *Round Robin* com pesos para balanceamento de carga entre cada cliente. Cada *Proxy Server* é responsável por comunicar com os demais microsserviços privados ao servidor, baseado com a área de interesse de sua conexão. O protocolo de comunicação

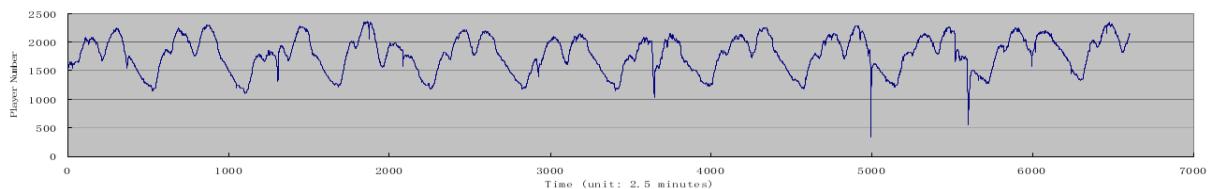
Figura 3.1: Arquitetura distribuída utilizando proxy.



Adaptado de: (HUANG; YE; CHENG, 2004).

utilizado entre o Cliente e *Proxy Server* é baseado em RPC (FARBER, 2002; BORELLA, 2000), porém não é relatado sobre o protocolo de comunicação utilizado entre o *Proxy Server* e o *Game Server*. A sua arquitetura pode ser observada na Figura 3.1, na qual obteve seus dados durante 100 dias para realizar as análises. A Figura 3.2 demonstra uma amostra do número de conexões pelo tempo no serviço obtido.

Figura 3.2: Número de conexões no serviço pelo tempo decorrido.

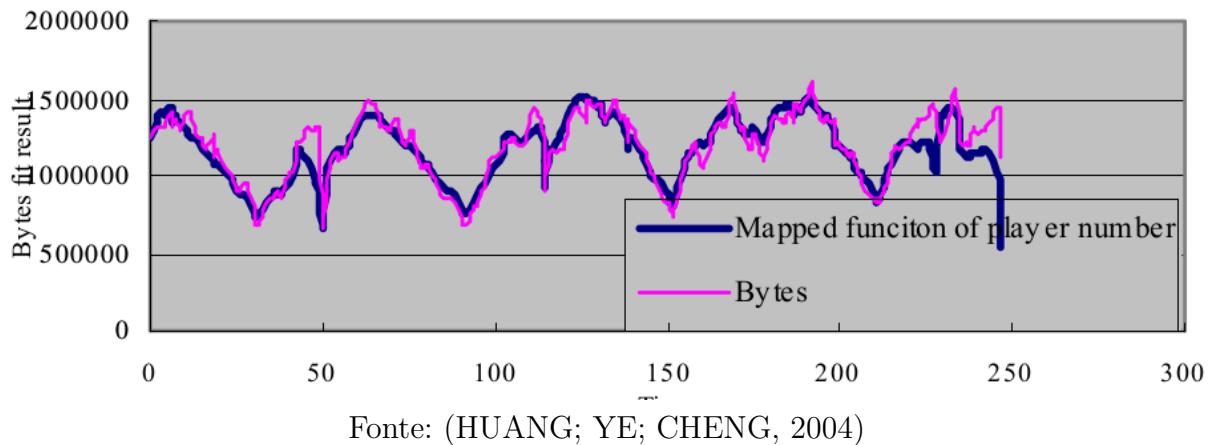


Fonte: (HUANG; YE; CHENG, 2004).

Como análise, o autor correlacionou o número de conexões com número de pacotes e banda, consumidos, utilizando uma função estatística linear. Esta função pode ser utilizada com regressão linear para prever consumo de recursos futuros e por fim realocar mais recursos ao serviço, contribuindo com escalabilidade vertical autônoma. Um exemplo de aplicação dessa regressão linear pode ser visualizada na Figura 3.3, no qual o autor compara o consumo de banda real comparado a regressão linear.

Entretanto, a escalabilidade horizontal não pode ser prevista, visto que não é

Figura 3.3: Regressão linear comparado ao consumo de banda real do servidor.



Fonte: (HUANG; YE; CHENG, 2004)

analisado o posicionamento de cada personagem a fim de dividir os ambientes em pedaços menores com outros serviços. Como trabalhos futuros é relatado a análise do posicionamento de personagens para escalabilidade horizontal, a análise de outras arquiteturas e a análise de outros gêneros de jogos, além do impacto de utilizar balanço de carga e provisionamento de recursos de forma dinâmica.

## 3.2 Villamizar et al. (2016)

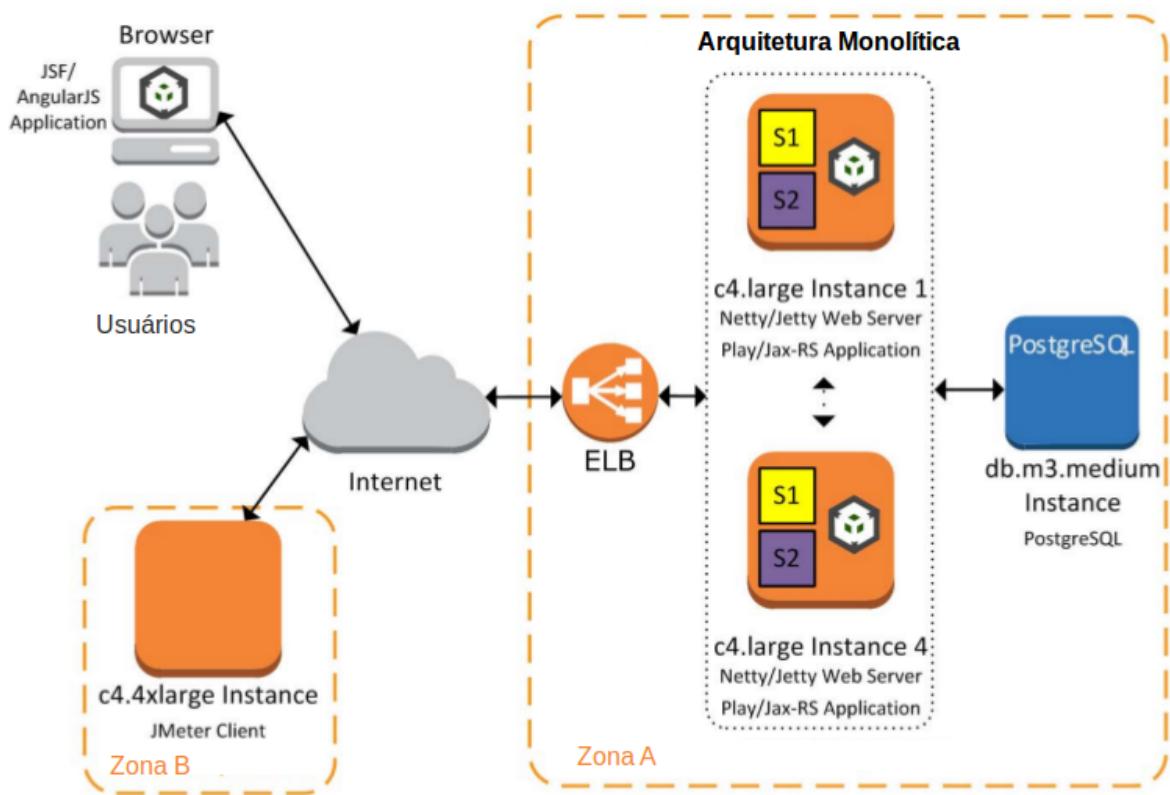
O trabalho de (VILLAMIZAR et al., 2016) investiga o custo de arquiteturas de microsserviços, arquiteturas *Platform as a Service* (PaaS) orientadas a eventos e aplicações monolíticas para aplicações web. A sua principal motivação é a comparação de custos para a tradução de sistemas legados para arquiteturas distribuídas. Para isso, o autor preparou três instâncias de testes com suas configurações desenhadas a fim de ter o maior número de requisições por minuto com o mesmo custo financeiro.

**Instância I.** Utilizando quatro instâncias AWS *c4.large*, uma instância AWS *c4.xlarge* e uma instância AWS *db.m3.medium*. A Figura 3.4 exibe a implantação de uma aplicação web monolítica. Essa arquitetura foi implementada utilizando *Jax-RS* e *Play Framework*.

**Instância II.** Utilizando três instâncias AWS *c4.xlarge*, uma instância AWS *t2.small* e uma instância AWS *db.m3.medium*. A Figura 3.5 exibe a implantação de uma aplicação de microsserviços web. Essa arquitetura foi implementada utilizando *Play Framework*, framework para desenvolvimento web para a linguagem Java e Scala<sup>1</sup>.

<sup>1</sup>Play Framework: <https://www.playframework.com/>

Figura 3.4: Arquitetura monolítica web implementada na AWS.



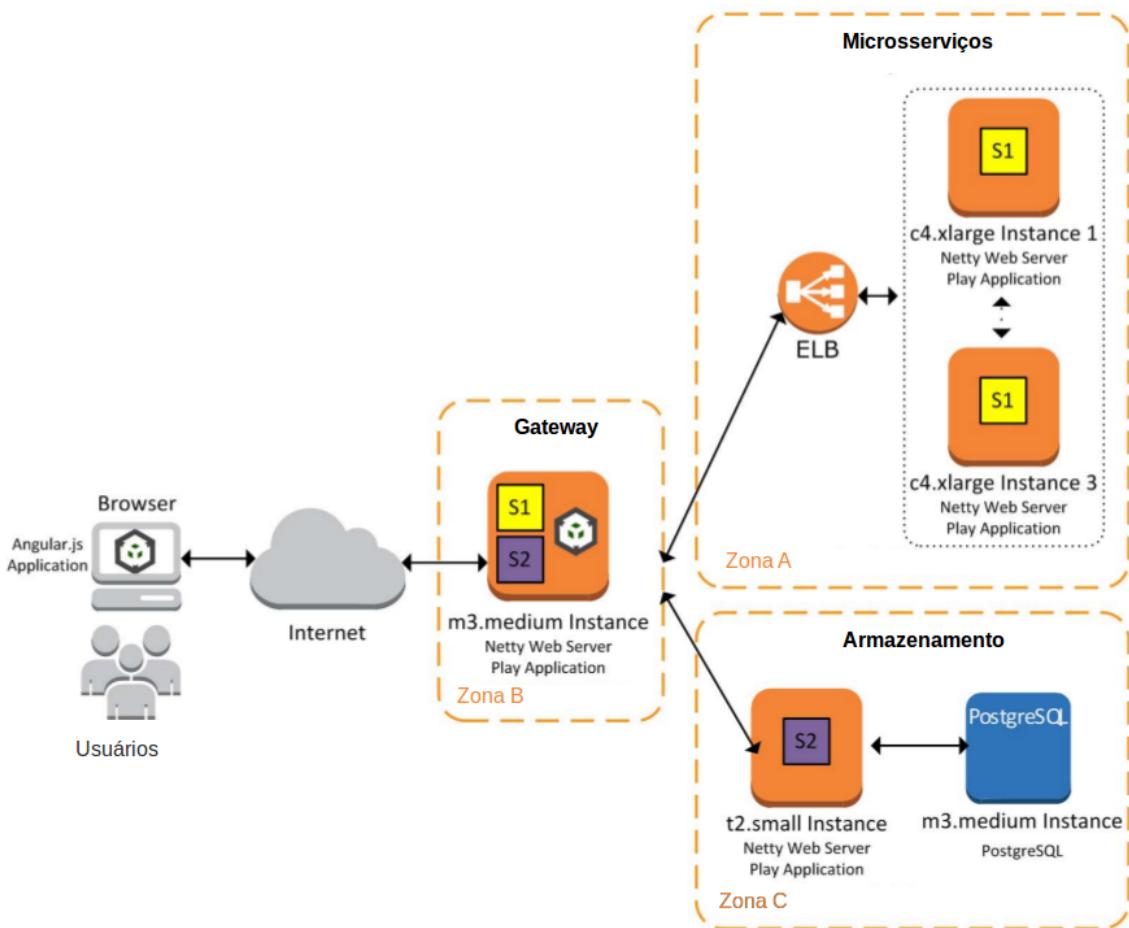
Fonte: (VILLAMIZAR et al., 2016)

**Instância III.** Utilizando duas instâncias AWS *lambda* *S1*, duas instâncias AWS *lambda* *S2*, uma instância AWS *S3 Bucket* e uma instância AWS *db.m3.medium*. A Figura 3.6 exibe a implantação de uma aplicação de microsserviços web utilizando a tecnologia AWS *lambda*. Essa arquitetura foi implementada utilizando o *framework Node.js*<sup>2</sup>, nas quais as funções de *gateway* são implementadas em quatro funções independentes do tipo *microservice-http-endpoint*.

Foi concluído que a arquitetura de microsserviços, nas condições desta aplicação, podem reduzir até 13.42% em gastos com a infraestrutura. Essa redução pode ser observada na Figura 3.7. O autor alerta sobre tolerância a falhas, transações distribuídas, distribuição de dados e versionamento de serviço.

<sup>2</sup>Node.js: <https://nodejs.org/en/>

Figura 3.5: Arquitetura de microsserviços web implementada na AWS.



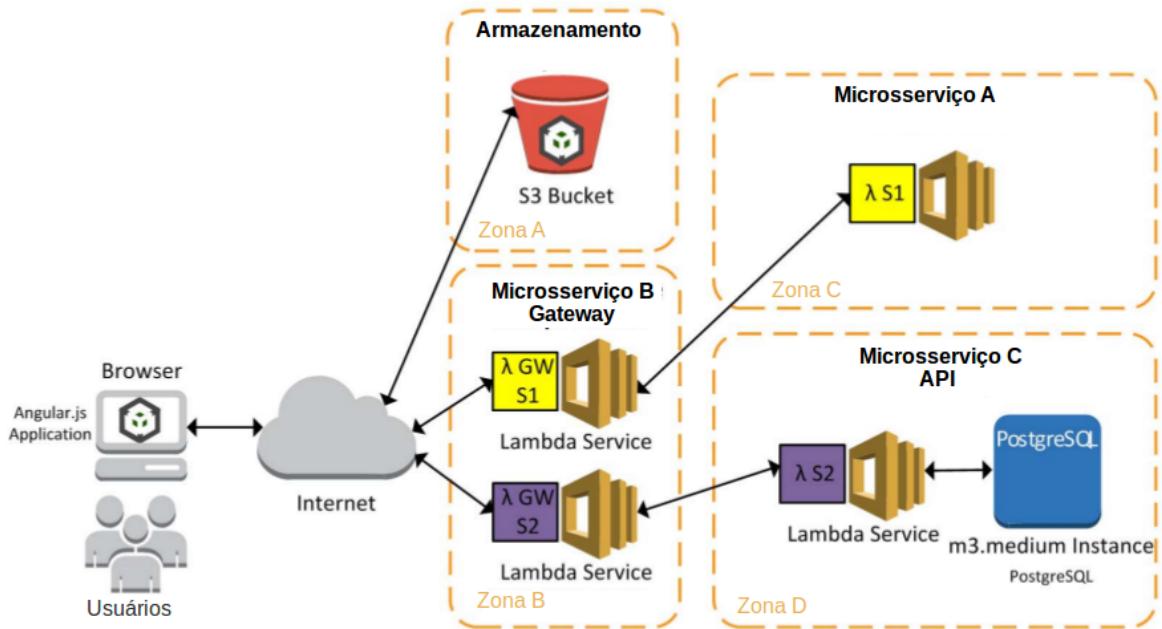
Fonte: (VILLAMIZAR et al., 2016)

### 3.3 Suznjevic e Matijasevic (2012)

O trabalho de (SUZNJEVIC; MATIJASEVIC, 2012) tem seu objetivo a fim de prever a carga a qual um serviço MMORPG pode receber utilizando a complexidade das operações nos contextos de *Player vs Player* (PvP) e *Player vs NPCs* (PvNPCs) a qual um personagem pode realizar em um ambiente. Este trabalho usa com base o modelo descrito na Seção 3.1, um modelo distribuído em serviços na qual efetuam o processamento de uma região do ambiente virtual.

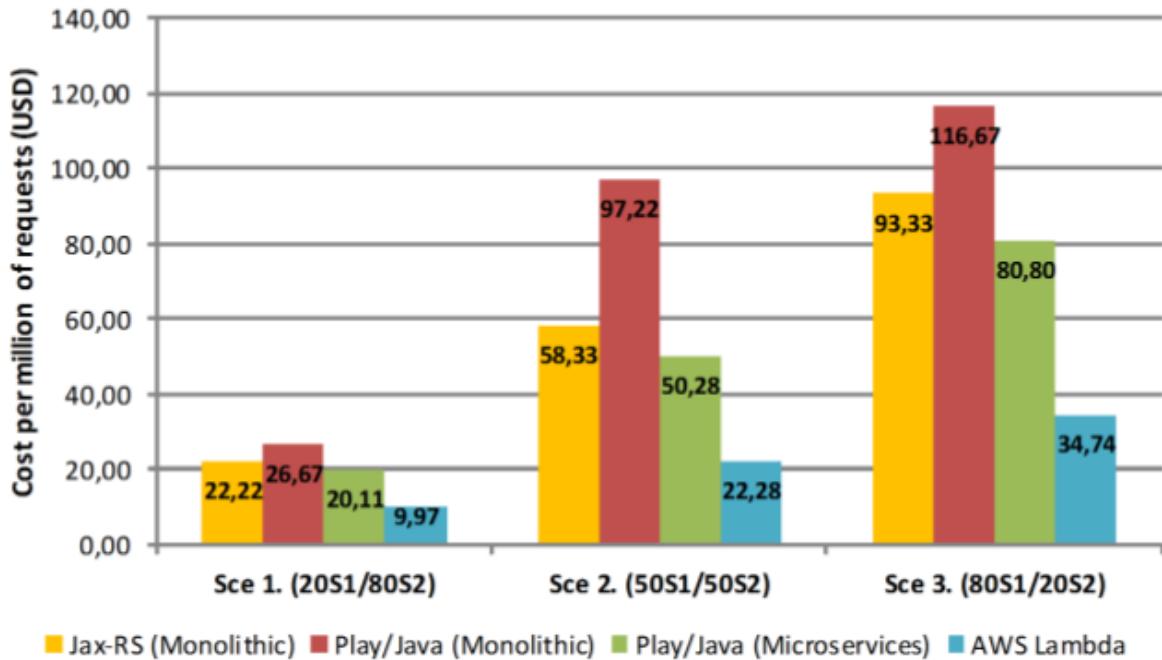
A análise realizada leva em conta a complexidade das ações no ambiente, a qual pode ser descrita na Tabela 3.1. Essa tabela exibe as ações que podem ser executadas para interagir com o ambiente, seja essa interação com PvP (jogador com outro jogador) ou PvNPCs (jogador com um personagem ou objeto gerenciado pelo serviço). Os contextos analisados nessa tabela são:

Figura 3.6: Arquitetura de microsserviços web implementada na AWS utilizando a tecnologia *lambda*.



Fonte: (VILLAMIZAR et al., 2016)

Figura 3.7: Custo por um milhão de requisições em dólares utilizando diferentes arquiteturas sobre a AWS.



Fonte: (VILLAMIZAR et al., 2016)

- *Questing:* Contexto de missão, na qual um grupo de jogadores ou um grupo de NPCs podem ser afetados nas ações.
- *Trading:* Contexto de negociação, na qual a complexidade leva em conta somente o

Tabela 3.1: Complexidade da interação com o ambiente, por contexto da interação.

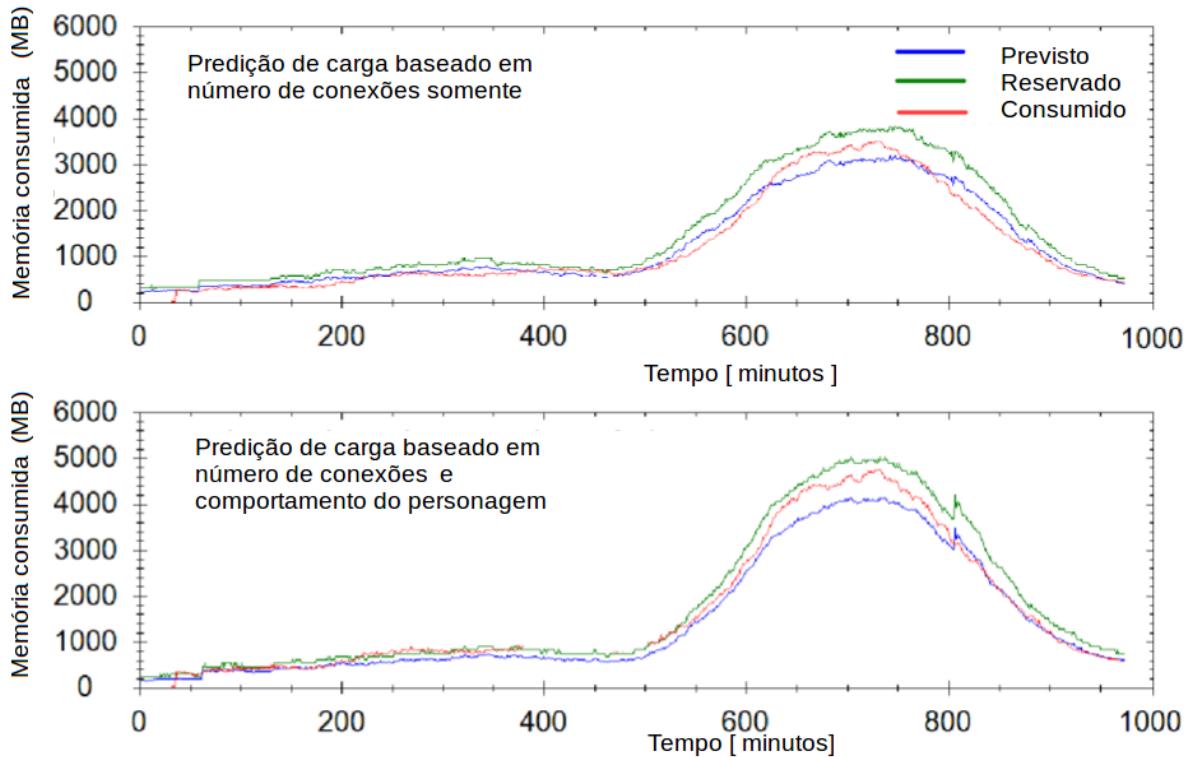
Contexto da ação	PvP	PvNPCs	Número de NPCs	Network [kbits/s]
Questing	$O(n)$	$O(n \log(n))$	$N \leq 6$	11.4
Trading	$O(n)$	$O(n)$	$N \leq 20$	8.1
Dungeons	$O(n^2)$	$O(n^2)$	$N \leq 20$	18.3
PvP combat	$O(n^3)$	$O(n)$	$N = 0$	24.1
Raiding	$O(n^2 \log(n))$	$O(n^3)$	$N \leq 40$	32.0

Fonte: (SUZNJEVIC; MATIJASEVIC, 2012)

número de itens negociados.

- *Dungeons*: Contexto de exploração, na qual o ambiente pode ser modificado conforme as ações dos personagens em um ambiente isolado para este grupo.
- *PvP combat*: Contexto de batalha entre jogadores, na qual as ações entre os jogadores influenciam diretamente o estado do personagem oponente.
- *Raiding*: Representa um contexto específico de exploração, na qual múltiplos jogadores unem forças a fim de combater outro grupo de jogadores ou NPCs.

Figura 3.8: Regressão levando em conta a complexidade das ações e contexto dos personagens.



Fonte: (SUZNJEVIC; MATIJASEVIC, 2012)

A abordagem utilizou as complexidades das ações, o número de conexões e o contexto de cada personagem no ambiente para predizer a banda utilizada. Pode-se visualizar uma regressão na Figura 3.8.

O autor conclui que o contexto de interação com o ambiente de cada personagem tem relevância com o consumo de CPU e banda, a qual pode ser calculada com sua complexidade a fim de desenvolver uma ferramenta para predição de carga sobre serviços MMORPG. Entretanto, essa predição não é feita em tempo real, não contribuindo para a automação da escalabilidade vertical e horizontal da arquitetura de microsserviços.

### 3.4 Análise dos trabalhos relacionados

Para os trabalhos relacionados, são analisados o tipo de pesquisa realizado sobre Microsserviços e serviços MMORPG. Também são levantados quais os recursos computacionais relacionados com estas análises.

Nos trabalhos relacionados existem duas abordagens utilizadas pelos autores, na qual estão relacionados a Previsão de Carga ou Comparação entre Arquiteturas de microsserviços (VILLAMIZAR et al., 2016; SUZNJEVIC; MATIJASEVIC, 2012):

- **Comparação de arquiteturas:** O autor utiliza alguma métrica a fim de decidir qual a melhor alternativa de arquitetura para determinada aplicação.
- **Previsão de carga:** Projetar a carga futura sobre algum recurso a fim de escalar automaticamente a sua arquitetura na nuvem, a fim de reduzir gastos de recursos ociosos e diminuir o impacto de ocorrências aos usuários finais.

Dessa forma, pode-se dividir os trabalhos relacionados com a sua categoria. Esta relação pode ser visualizada na Tabela 3.2.

Tabela 3.2: Trabalhos relacionados por categoria.

Autor	Categoria
(SUZNJEVIC; MATIJASEVIC, 2012)	Previsão de carga
(VILLAMIZAR et al., 2016)	Comparação de arquiteturas
(HUANG; YE; CHENG, 2004)	Previsão de carga

Fonte: O próprio autor.

Para o trabalho referente a comparação de arquiteturas (VILLAMIZAR et al., 2016), o recurso utilizado foi o custo por um determinado número de requisições

web. Já para os trabalhos referentes a previsão de carga (SUZNJEVIC; MATIJASEVIC, 2012; HUANG; YE; CHENG, 2004) foi levantado o consumo da banda, CPU e memória, entretanto não levantou-se o número de conexões e latência aos usuários, na qual podem ser observados na Tabela 3.3. Nenhum trabalho relatou limite de conexões ou latência do sistema.

Tabela 3.3: Trabalhos relacionados por recurso analisado.

Autor	CPU	Memória	Banda	Custo	Latência	Limite de conexões	Complexidade de Algoritmos
(HUANG; YE; CHENG, 2004)	✗	✗	✓	✗	✗	✗	✗
(VILLAMIZAR et al., 2016)	✗	✗	✗	✓	✗	✗	✗
(SUZNJEVIC; MATIJASEVIC, 2012)	✓	✓	✓	✗	✗	✗	✓

Fonte: O próprio autor.

Outro ponto a ser analisado são as arquiteturas utilizadas. Os trabalhos referentes a previsão de carga não utilizaram uma arquitetura de microsserviços, mesmo sendo um sistema distribuído. Nesses serviços, os sistemas eram dependentes entre si e precisavam um dos outros para o seu funcionamento. A relação de arquiteturas abordadas pode ser visualizado na Tabela 3.4, na qual mostra quais trabalhos abordaram os temas sobre arquiteturas de microsserviços e jogos MMORPG. Além disso, não foi descrito um método de replicação automática dos serviços a fim de prover alta disponibilidade de forma automatizada, sendo abordado como um trabalho futuro.

Tabela 3.4: Arquiteturas analisadas.

Autor	Arquitetura de Microsserviços	Arquitetura Distribuída	MMORPG
(HUANG; YE; CHENG, 2004)	✗	✓	✓
(VILLAMIZAR et al., 2016)	✓	✓	✗
(SUZNJEVIC; MATIJASEVIC, 2012)	✗	✓	✓

Fonte: O próprio autor.

Este Trabalho de Conclusão de Curso pretende analisar uma arquitetura de microsserviços especificada a jogos MMORPG, visando métricas de desempenho e custo de operação. Os recursos analisados serão CPU, Memória, Banda, Custo, Latência, Limite de Conexões e Complexidade dos Algoritmos empregados.

## 3.5 Considerações parciais

Após a busca de trabalhos relacionados, encontrou-se três principais trabalhos a qual analisam ora arquiteturas de microsserviços ora arquiteturas de jogos MMORPG. Nesse

sentido, tais trabalhos relacionados não realizam uma intercessão entre ambos os temas, a qual o atual trabalho propõe a análise. Também foi identificado as métricas a qual estes trabalhos analisaram. Nesse sentido, o atual trabalho irá propor a análise sobre as mesmas métricas, entretanto completando com métricas que são de interesse para analisar pontos de falha em tais arquiteturas.

Por fim, foi apresentado trabalhos relacionados a qual guiaram os moldes do plano de testes e a proposta deste trabalho, porém com utilizando ambos os temas de arquiteturas de microsserviços e jogos MMORPG, realizando uma análise com mais características sobre o serviço do jogo.

## 4 Proposta para análise de arquiteturas MMORPG

As arquiteturas de serviços MMORPG são desenvolvidas visando suprir as necessidades do projeto do jogo desenvolvido, de forma a viabilizar a utilização deste serviço. Nesse sentido, jogos com mecânicas de projeto similares possuem implementações parecidas para os clientes. Entretanto, a arquitetura escolhida impacta no custo de operação e qualidade do serviço aos jogadores. Por este motivo, diferentes arquiteturas com o mesmo *design* não são comparáveis entre si, visto que dependem das regras de negócio do jogo.

Ao desenvolver um serviço MMORPG é necessário decidir uma arquitetura que possibilite reduzir custos, consumo de recursos e minimize ocorrências para os jogadores a fim de viabilizar a sua implantação como produto. Porém, a impossibilidade de comparação direta entre as arquiteturas de serviço MMORPG instiga a análise das características básicas destas arquiteturas que possam influenciar o *game design*, tais como consumo de recursos, tempo de resposta, latência e número máximo de clientes simultâneos conectados nos microsserviços. Sendo assim, uma análise do consumo de recursos computacionais das arquiteturas levantadas previamente na literatura tem valor científico no auxílio da escolha de implementações de arquiteturas de microsserviços, em específico para serviços MMORPG.

Neste capítulo é descrita a proposta para análise de consumo de recursos computacionais em arquiteturas MMORPG. Inicialmente, é descrita a proposta em alto nível (Seção 4.1), trazendo os objetivos desta análise, quais recursos e métricas serão analisadas (TCC-II). Os Critérios de Análise (Seção 4.2) exibem como os dados obtidos devem ser interpretados, baseando-se nos objetivos da análise das arquiteturas. O Plano de Testes (Seção 4.3) exibe como será realizada a coleta dos dados, descrevendo o ambiente, cenário, critérios e os testes que serão realizados.

## 4.1 Proposta

Tendo analisado os trabalhos relacionados (Seção 3) e as arquiteturas específicas para jogos MMORPG, o presente trabalho tem como objetivo analisar as arquiteturas MMORPG visando complementar a análise de arquitetura e consumo de recursos computacionais não analisados nos trabalhos relacionados. Em específico, serão obtidos os valores referentes aos uso dos seguintes recursos computacionais nas arquiteturas Rudy (Subseção 2.5.1), Salz (Subseção 2.5.2) e Willson (Subseção 2.5.3):

1. **CPU:** o uso de CPU, com sua representação sendo em relação a porcentagem de processamento nos núcleos utilizados;
2. **Memória:** Quantidade de memória utilizada pelos processos do serviço/arquitetura. A sua representação será como dado absoluto;
3. **Rede:** Banda de rede utilizada nas operações de entrada e saída para cada microserviço, utilizando valores absolutos. Juntamente será obtido o valor de latência do cliente ao microsserviço, verificando se o congestionamento da rede afeta a latência do microsserviço.

Além dos recursos computacionais, esta análise levará em conta valores referentes a outras métricas. As métricas, cujos os valores serão obtidos são:

1. **Número máximo de jogadores simultâneos:** Descobrir o limite de conexões para as arquiteturas propostas a análise. Será representado como valor absoluto.
2. **Tempo de resposta das requisições:** Descobrir o tempo de resposta por categoria de requisição, conforme o número de jogadores no serviço. Será representado como tempo decorrido, em milissegundos.

Todos estes valores serão obtidos a partir de simulações, e por este motivo faz-se necessário descrever o comportamento dos jogadores simulados (Seção 4.3.2). Espera-se, em situações adversas, caracterizar os comportamentos das arquiteturas bem como gargalos e os custos de recursos computacionais para manutenção das arquiteturas de microsserviços. Para este fim, faz-se necessário a descrição dos critérios que serão utilizados durante a análise dos valores obtidos nos experimentos.

## 4.2 Critérios de análise

A fim de padronizar a análise dos dados obtidos, estes serão estabelecidos usando como base o comportamento dos valores obtidos em referenciais. Neste sentido, a análise dos dados obtidos será guiada pelo esperado dos valores obtidos em um serviço padrão:

1. **Consumo CPU:** Espera-se estressar com um elevado número de requisições.
2. **Consumo Memória:** Espera-se estressar com requisições nas quais exija armazenamento em memória.
3. **Vazão Rede - Entrada:** Espera-se estressar com requisições nas quais tenham uma carga de dados elevada.
4. **Vazão Rede - Saída:** Espera-se estressar com respostas nas quais tenham uma carga de dados elevada.
5. **Número de Conexões Simultâneas:** Servirá como guia de comparação com os demais valores e desempenho da arquitetura;
6. **Tempo de resposta das requisições:** Servirá como guia de desempenho da arquitetura; e
7. **Latência entre cliente e serviço:** Servirá como guia de comparação com os demais valores.

Em um caso de uso ideal, todos os recursos não são estressáveis, com um número de conexões simultâneas elevado. Porém, espera-se para este trabalho um possível conjunto de ocorrências, na qual podem ou não ocorrer. Este conjunto servirá de guia / exemplo de problemas relevantes retirado dos valores obtidos. Logo, a simulação e cenários foram elaborados para forçar tais ocorrências.

A partir dos valores obtidos, e seguindo o esperado de uma arquitetura totalmente relacionada ao número de conexões, espera-se encontrar um conjunto de eventuais problemas nas arquiteturas. Um conjunto exemplo destes problemas estão listados na Tabela 4.1.

Tabela 4.1: Possíveis conjuntos para a análise.

Recursos							Descrição
CPU	Memória	Rede Entrada	Rede Saída	Conexões Simultâneas	Tempo de Resposta	Latência	
↑				↖			Rotina de processamento de requisições está ocupando muita CPU
	↑			↓			O microsserviço está armazenando informações as quais poderiam estar alocadas em outros microsserviços
		↑		↓			Uma entrada de dados elevada pode indicar o uso de um protocolo inapropriado para o serviço
			↑	↓			Caso a saída esteja muito elevada pode indicar uma configuração inapropriada de elementos que são transitados na rede ou uso inadequado de protocolos
				↓	↑		Pode estar relacionado ao desempenho de processamento, modelo de paralelismo ou congestionamento de rede
				↓	↑	↑	Está relacionado com congestionamento da rede
↓		↑	↑				Possível gargalo na rede ou protocolo ineficiente
↑	↑	↓	↓				Possível gargalo nos algoritmos utilizados no serviço
				↓			Bloqueio de novas conexões pelo sistema operacional ou modelo de paralelismo
↑	↑	↑	↑	↑	↓	↑	Límite de processamento da arquitetura
↓	↓	↓	↓	↑	↓	↓	Teste ideal

Fonte: O próprio autor.

Não foram encontrados trabalhos para guiar a caracterização dos dados, sendo assim a caracterização foi definida genericamente. Dessa forma, a linha de base para a caracterização será encontrada ao decorrer da análise, utilizando dos valores de testes com poucas conexões (nenhum cliente e um cliente), esperando que o serviço escale linearmente conforme o inicio do processo. A linha de base definida será uma das contribuições para trabalhos futuros.

A Tabela 4.1 relaciona os recursos conforme dois padrões:

- Valores acima da média ( $\uparrow$ ).
- Valores baixo da média ( $\downarrow$ ).

Espera-se encontrar problemas mais detalhados, além de problemas padronizados de forma genérica na Tabela 4.1. Pode ser possível identificar eventuais problemas conforme o tipo de requisição e projeto da arquitetura. Estes problemas servirão como guias na análise final das arquiteturas.

Segundo estes critérios de análise, torna-se necessário definir um plano de testes a fim de obter os dados conforme os cenários e casos de uso definidos no atual trabalho. Tais testes servem para ocasionar situações nos serviços a fim de obter dados para posterior análise.

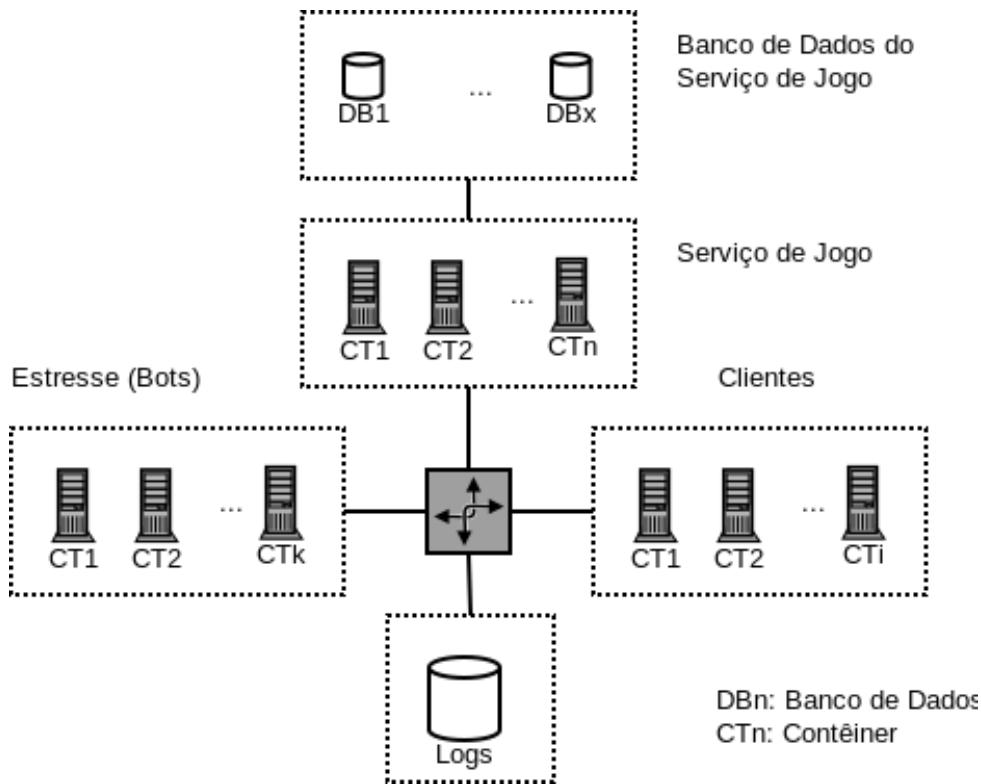
## 4.3 Plano de testes

O plano de testes define os cenários que serão aplicados sobre as arquiteturas de microserviços para jogos MMORPG selecionadas. Esta seção serve para descrever formas de estressar as arquiteturas, a fim de obter valores para análise. Entretanto, antes de relatar os cenários de teste, é importante descrever o ambiente no qual serão realizados os experimentos. A Figura 4.1 descreve a infraestrutura utilizada para execução das camadas de aplicação utilizadas nos testes.

Como visível na Figura 4.1, o ambiente de testes planejado está organizado em cinco regiões. Essas regiões foram isoladas com o objetivo de diminuir o impacto de desempenho e consumo de recursos por outras ferramentas durante a coleta de dados. Por este motivo, as regiões da infraestrutura planejada são:

1. **Serviço de Jogo:** A camada de serviço da infraestrutura dos testes concentrará a arquitetura dos microsserviços referente às arquiteturas de microsserviços analisadas.
2. **Banco de dados do serviço de jogo:** A camada de banco de dados do serviço de jogo conterá os serviços de dados e web a fim de manter um padrão de banco de dados para ambos os serviços utilizados e auxiliar na inicialização dos testes.

Figura 4.1: Ambiente de testes definido para a coleta de dados.



Fonte: O próprio autor.

3. **Estresse:** A camada de estresse será responsável por realizar requisições ao serviço a fim de estressá-lo, simulando padrões de requisição de um padrão de um jogador.
4. **Cliente:** A camada de cliente será composta pelos mesmos elementos da camada de estresse, porém em um ambiente controlado para que a alta demanda dos clientes neste ambiente não interfira nas métricas obtidas.
5. **Dados:** A camada de dados será composta por um banco de dados de *log* a fim de armazenar os dados obtidos da camada Cliente e Serviço, posteriormente utilizado exclusivamente na coleta de dados.

Tais regiões da infraestrutura utilizada no ambiente de testes devem manter um padrão ao qual seu propósito é garantir a inexistência de interferência entre os testes, focando em obter métricas válidas para posterior análise. Dessa forma, espera-se dividir as aplicações conforme a sua rede, facilitando o seu monitoramento. Entretanto, por se tratar de um sistema baseado em serviço, espera-se utilizar uma considerável parte do mesmo sistema de cliente para as três arquiteturas, excluindo-se os casos no quais a arquitetura necessite de alterações.

Para os casos de uso, serão utilizadas as arquiteturas de microsserviços específicos a jogos MMORPG obtidos da literatura. São essas elas:

1. Arquitetura Rudy (Subseção 2.5.1), na qual baseia-se na segregação de jogadores por canais;
2. Arquitetura Salz (Subseção 2.5.2), na qual baseia-se em gerar muitos serviços escaláveis; e
3. Arquitetura Willson (Subseção 2.5.3), na qual baseia-se em extrair microsserviços de regras de negócio mais custosas.

Tais arquiteturas vão impactar o serviço de jogo, banco de dados e as requisições as quais os clientes deverão realizar. Espera-se obter os valores referente a diferença de consumo de recursos computacionais dentro de cenários controlados utilizando o ambiente de testes.

Com o objetivo de obter dados, torna-se claro a necessidade de estresse das arquiteturas em múltiplos casos diversos, garantindo assim a confiabilidade dos dados obtidos. Dessa forma, foi desenvolvido um cenário que comporte ambas as arquiteturas de microsserviços propostas na análise. Este cenário possibilita a execução do experimento junto a simulação de clientes.

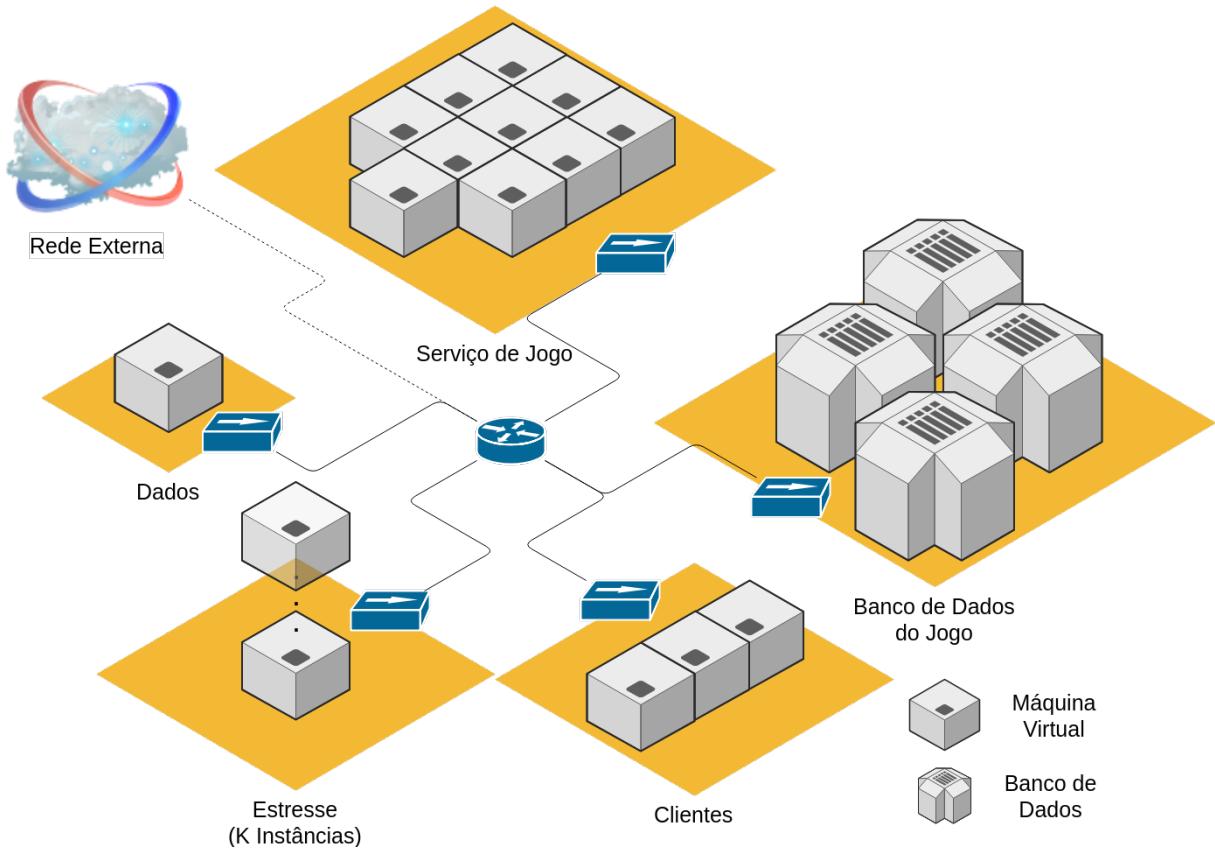
#### 4.3.1 Cenário

O cenário reflete diretamente sobre o ambiente proposto para esta análise. Este será executado sobre cinco camadas, nas quais cada camada estará isolada em sub-redes em uma nuvem de computadores.

O cenário será composto por cinco sub-redes, as quais cada uma será responsável pela operação de uma região do ambiente planejado (Figura 4.2). Esta divisão física em redes facilitará a orquestração dos serviços na rede, seguindo boas práticas de implantação de microsserviços. Além disso, pode-se garantir que a interface do serviço público para o cliente segue o padrão de um serviço MMORPG real.

O cenário, exibido na Figura 4.2 descreve cinco sub-redes. Estas redes são descritas da seguinte forma:

Figura 4.2: Rede de execução dos testes.



Fonte: O próprio autor.

1. **Dados:** Armazena e processa os dados obtidos da arquitetura. É usado pelas redes *Cliente* e *Serviço de Jogo*.
2. **Banco de dados do serviço de jogo:** Armazena dados do jogo. É usado pela rede *Serviço de Jogo*.
3. **Serviço de Jogo:** Executa sistemas *web* e RPC, em específico da camada de serviço de jogo. Gera métricas para os serviços na rede *Dados* e manipula os bancos de dados na rede *Banco de dados do serviço de jogo*.
4. **Estresse:** Executa múltiplos clientes, a fim de estressar o serviço de Jogo.
5. **Clientes:** Executa um número controlado de clientes para obter métricas de tempo de resposta e latência entre as redes *Cliente* e *Serviço de Jogo*.

Conforme a descrição de cada rede, existe uma interdependência dentre as redes. Tal interdependência pode ser visualizada na Tabela 4.2.

A Tabela 4.2 refere-se a dependência das redes, conforme os serviços necessários

Tabela 4.2: Tabela de interdependência das sub-redes.

<i>Linha depende de Coluna</i>	Dados	Banco de dados do serviço de jogo	Serviço de Jogo	Estresse	Clientes
Dados	–	Não	Não	Não	Não
Banco de dados do serviço de jogo	Não	–	Não	Não	Não
Serviço de Jogo	Sim	Sim	–	Não	Não
Estresse	Não	Não	Sim	–	Não
Clientes	Sim	Não	Sim	Não	–

Fonte: O próprio autor.

para o seu funcionamento. Portanto, espera-se bloquear o tráfego de pacotes entre redes que são independentes.

A implantação do serviço de jogo utilizará métodos de implantação de micro-serviços, com ferramentas para gerenciamento de nós em uma rede de contêineres. Por este motivo, todas as máquinas virtuais da rede *Serviço de Jogo* terão os mesmos recursos, facilitando o comportamento do gerenciador ao escalar os serviços.

A implantação dos bancos de dados do serviço de jogo devem ser implantadas diretamente no sistema operacional da máquina virtual. Seguindo assim uma boa prática de não armazenar dados dentro de contêineres. Por sua vez, estas máquinas virtuais serão focadas em armazenamento.

A implantação dos clientes sobre a rede *Clientes* executará um número limitado de contêineres fixo sobre as máquinas virtuais, para que não estresse as instâncias desta rede. Nesse sentido, os recursos computacionais requeridos por estas instâncias são menores em relação aos demais.

A estrutura de implantação da rede *Estresse* será dada por um gerenciador de nós em uma rede de contêineres. Esta rede terá mais instâncias sob demanda conforme o caso de teste.

O limite de recursos do cenário é definido na Tabela 4.3, sendo definida a

faixa de endereços de cada rede a qual foi projetada. Estes valores podem ser alterados conforme a necessidade dos testes, tendo seus recursos reduzidos ou incrementados.

Tabela 4.3: Limite de recursos por instância de cada rede.

Nome da Rede	Rede	Instâncias	Armazenamento / Ins.	N. Núcleos	Memória
Dados	10.0.*.* / 255.255.0.0	1	250GB	4	4GB
Banco de dados do serviço de jogo	10.51.*.* / 255.255.0.0	4	100GB	4	2GB
Serviço de Jogo	10.52.*.* / 255.255.0.0	10	25GB	4	4GB
Estresse	10.100.*.* / 255.255.0.0	N	25GB	4	4GB
Clientes	10.101.*.* / 255.255.0.0	3	10GB	2	1GB

Fonte: O próprio autor.

A partir da Tabela 4.3, espera-se definir um limite de recursos máximos utilizados pelo Serviço de Jogo. A única rede que pode variar conforme o teste será a rede Estresse, visto que a demanda para estressar uma rede pode mudar conforme as características do teste.

A partir deste cenário é definido qual o comportamento dos clientes na execução dos testes. Nesse sentido, uma definição das características mínimas de regra de negócio, padrão de comportamentos e interface esperada para o serviço e cliente é necessária.

### 4.3.2 Simulação de Clientes

Utilizando a simulação de clientes objetiva-se estressar as arquiteturas utilizando um ataque com *bots*. Neste cenário, para padronizar a coleta de dados, todos os *bots* terão a mesma rotina evitando assim um comportamento aleatório, a qual pode descharacterizar os dados obtidos para análise.

Porém, como requisito para estipular as requisições, faz-se obrigatório realizar um levantamento de requisitos no qual tanto o serviço quanto o cliente devem implementar. Nesse sentido, a Tabela 4.4 relaciona a funcionalidade com o impacto de implementação de tal funcionalidade.

A Tabela 4.4 relata uma lista de funcionalidades mínimas que serão executadas na simulação. A partir destas funcionalidades, pode-se definir quais requisições estarão disponíveis na API para o cliente requisitar ao serviço. A lista de comandos públicos é exibida na Tabela 4.5.

A Tabela 4.5 descreve todos os comandos que estarão disponíveis na rede. Além destes, serão implementados outros comandos para API privada do serviço. Porém, os

Tabela 4.4: Requisitos das funcionalidades e respectivo impacto de implementação.

Requisito	Descrição	Implementação
Identificação	Gera uma numeração única (token) com base em uma tupla de dados.	Será implementado utilizando algoritmo de <i>hash</i> , de forma a garantir que este token seja único e diferente a cada implementação.
Autenticação	Recebe o token e garante que não existe nenhuma conexão utilizando o mesmo token.	Será implementado usando um serviço de chave valor, como o Redis.
Seleção de Personagem	Uma conexão deve requerer o controle de um personagem.	Será implementado utilizando uma árvore de cena interna ao serviço, onde o tipo do nó será Personagem e o seu nome será o nome do personagem.
Envio de Mensagem	Será possível enviar mensagens e receber. Elas serão baseadas na região. Será mantido uma distância fixa para todos os casos de uso.	Deve existir uma estrutura de busca interna ao serviço para consultar personagens de uma região em relação a um usuário.
Movimentação	Será possível movimentar o personagem para as células adjacentes. Isso indica que o posicionamento do personagem será baseado em uma matriz.	Esta ação deve comunicar a atualização para todos os demais jogadores da região de interesse.
Ataque	Ao atacar, o jogador causará um dano aleatório baseado em seu nível em todos os inimigos das células adjacentes.	Esta ação irá manipular a árvore de objetos da cena do serviço e deverá notificar todos os jogadores desta área de interesse.
Consumo de Itens	Ao consumir um item, os atributos do personagem serão alterados, influenciando na regra de negócio utilizada nas arquiteturas de teste.	Implicará na utilização de um banco de dados em memória e manipulação de dados não visíveis pela árvore de cena.

Fonte: O próprio autor.

Tabela 4.5: Requisitos mínimos funcionais para a implementação da simulação descrita.

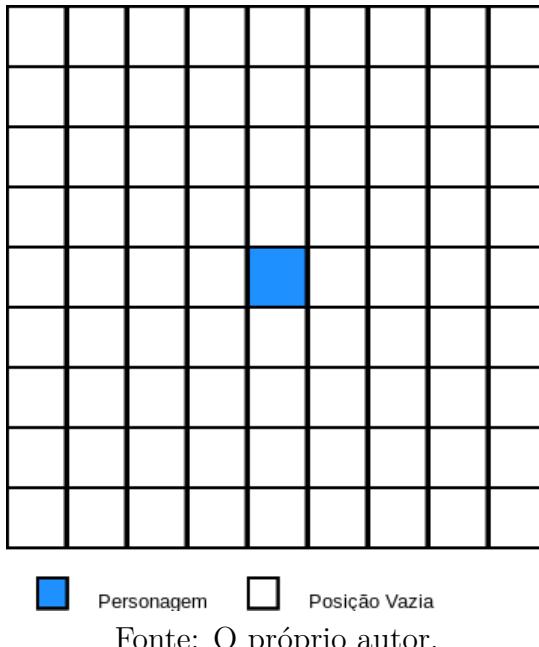
Nome	Argumentos	Retorno	Protocolo	Direção
<i>Auth</i>	<i>Username, Password</i>	JSON	Web	Cliente para Serviço
<i>CreateAccount</i>	<i>Username, Password</i>	JSON	Web	Cliente para Serviço
<i>UpdateAccount</i>	<i>Username, Password</i>	JSON	Web	Cliente para Serviço
<i>CreateCharacter</i>	<i>Token, Character Name</i>	JSON	Web	Cliente para Serviço
<i>DeleteCharacter</i>	<i>Token, Character ID</i>	JSON	Web	Cliente para Serviço
<i>SelectCharacter</i>	<i>Token, Character ID</i>	JSON	RPC	Cliente para Serviço
<i>WalkTo</i>	<i>Token, PosX, PosY</i>		RPC	Cliente para Serviço
<i>ConsumeItem</i>	<i>Token, Item</i>		RPC	Cliente para Serviço
<i>AttackHere</i>	<i>Token</i>		RPC	Cliente para Serviço
<i>SendMessage</i>	<i>Token, Message</i>		RPC	Cliente para Serviço
<i>UpdateMapEstate</i>	<i>NPC, Action, MoreData</i>		RPC	Serviço para Cliente
<i>UpdateCharacterEstate</i>	<i>NPC, Action, MoreData</i>		RPC	Serviço para Cliente
<i>ReceiveMessage</i>	<i>NPC, Message</i>		RPC	Serviço para Cliente
<i>ReBind</i>	<i>IP, Port</i>		RPC	Serviço para Cliente

demais comandos da API privada não serão utilizados pelo cliente, sendo necessariamente um requisito para o funcionamento do serviço com determinada arquitetura.

O ambiente da simulação será baseado em matrizes. Cada mapa pode ser visualizado como uma matriz de 100x100 unidades. Dessa forma, temos um ambiente com valor exato, o qual facilita cálculos internos do serviço e cliente, assim promovendo um ambiente vasto porém sem utilizar recursos de forma exagerada para os testes. Os personagens podem locomover-se para as células adjacentes a sua localização. O raio de interesse é de quatro células. Um exemplo de estado do ambiente do jogo pode ser visualizado na Figura 4.3.

A partir da área de interesse do jogo, como o da Figura 4.3, o *bot* poderá decidir

Figura 4.3: Área de interesse da simulação com raio de quatro células.

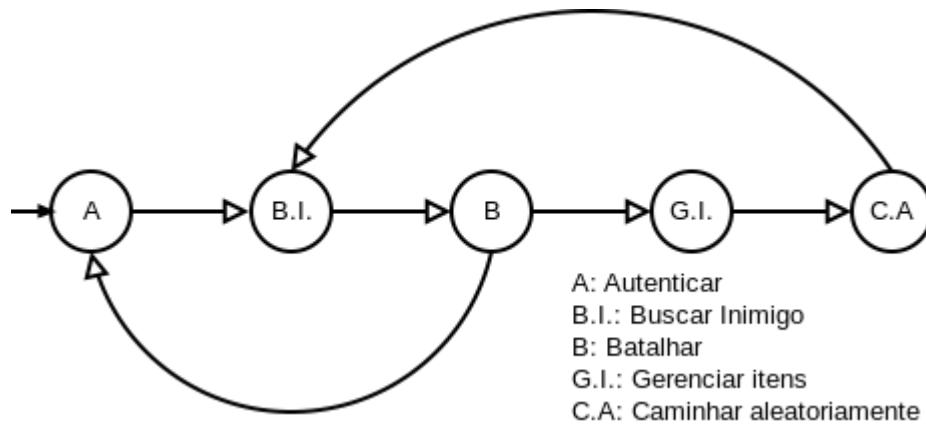


Fonte: O próprio autor.

suas ações baseado em um autômato. Caso ele alcance os extremos do mapa, ele será movimentado para outro mapa. Nos casos de arquiteturas com múltiplos gerenciadores de mundo, será utilizado o comando *ReBind* para realizar a conexão com o microsserviço correto após o translado do personagem.

Todas as ações do *bot* no mapa são baseadas em um autômato. Sendo assim, espera-se obter um padrão de movimentação a fim de evitar ciclos que um jogador comum dificilmente realizará (*e.g.*, Andar para frente e para trás, ficar equipando itens em ciclo, ficar consumindo itens até acabar, *etc.*). A movimentação do personagem seguirá o autômato descrito na Figura 4.4.

Figura 4.4: Autômato de movimentação dos personagens simulados.



Fonte: O próprio autor.

A Figura 4.4 descreve o comportamento de um *bot* no ambiente do jogo simu-

lado. Ele seguirá um padrão de procurar batalhas, batalhar, gerenciar itens do personagem e buscar novas batalhas. Este conjunto de ações simula o comportamento de busca de itens dentro de um jogo MMORPG. As características específicas de cada estado são definidas da seguinte forma:

- Autenticar: Realizará autenticação com o serviço *web* ou *rpc* apropriado a arquitetura. Neste passo o *bot* receberá as informações do seu personagem.
- Buscar Inimigo: Caminhará de forma aleatória a fim de buscar um inimigo, caso não exista em sua área de interesse. Caso encontre em sua área de interesse, irá aproximar deste inimigo.
- Batalhar: Irá atacar um inimigo, aplicando dano a este NPC. O personagem poderá receber dano. Estes valores serão fixos baseados em seu equipamento. O personagem pode perder todos os seus pontos de vida, sendo desconectado do serviço.
- Gerenciar Itens: O *bot* irá consumir itens para recuperar pontos de vida e usar equipamentos melhores aos já utilizados.
- Caminhar Aleatoriamente: O personagem caminhará aleatoriamente por um número de passos máximo. Após isso, voltará a buscar uma batalha.

Esta sequência de ações visa forçar aos *bots* a exploração do cenário. A cada mudança de estado, o jogador irá anunciar no chat a sua troca de ação. Isso contribuirá com o monitoramento do comportamento dos personagens tanto quanto usará a funcionalidade de chat. Para simular a ação de resposta de mensagens de um jogador, cada *bot* que receber uma mensagem terá uma porcentagem fixa de 25% de probabilidade de responder a sua ação no atual momento. Um *bot* não poderá responder a uma mensagem na qual ele mesmo emitiu na região.

Ao realizar um *ReBind* ou transitar de um mapa para o outro, o estado atual do autômato volta para o estado **Autenticar**. Caso já esteja autenticado, continuará a sua busca por inimigos na nova região.

O ambiente final da simulação possui três mapas no eixo horizontal e no eixo vertical, visto que esta é a combinação mínima para que o serviço tenha um mapa central com bordas para efetuar transições de novos personagens.

Após definido as características dos clientes simulados, pode-se definir os testes que serão executados sobre as arquiteturas de microsserviços específicos para jogos MMORPG. Tais testes servem de guia para obter as métricas para a análise das arquiteturas de microsserviços especificadas.

### 4.3.3 Testes

Os testes que serão executados no atual trabalho esperam analisar a quantia de recursos mínimos para executar os microsserviços e o crescimento de consumo de recursos conforme o crescimento de clientes simultâneos. Nesse sentido, foram definidos os seguintes testes:

1. Executar as arquiteturas com zero jogadores simultâneos, por 5 minutos;
2. Executar as arquiteturas com um jogador apenas; e
3. Executar as arquiteturas com zero jogadores simultâneos, aumentando a cada minuto o número jogadores (*e.g.*, 10) ao serviço, até o serviço obter algum erro interno de microsserviço e terminar a sua execução por erro interno.

Os dados são capturados ao decorrer do tempo, podendo assim relacionar os recursos consumidos com o número de conexões simultâneas no serviço e com os demais recursos. Esta estrutura auxiliará a visualização e interpretação dos dados para posterior análise.

Todos os testes serão executados utilizando a arquitetura Rudy, Salz e Willson. Eles serão executados sequencialmente, aplicando sobre o cenário cada arquitetura e seus devidos clientes sobre o cenário, a fim de obter as métricas para posterior análise. A partir desta sequência de testes, são obtidos valores suficientes para a análise das arquiteturas de microsserviços específicas a jogos MMORPG. Estes serão utilizados para análise conforme os critérios definidos na Seção 4.2, buscando possíveis gargalos e problemas de escalabilidade do sistema.

## 4.4 Considerações parciais

Este capítulo definiu os objetivos da análise de microsserviços específicos a jogos MMORPG. Para alcançar tais objetivos, se fez necessário definir quais métricas serão obtidas e como

estes dados serão analisados. Por fim, definiu-se qual o ambiente, cenário e testes que serão executados para obter tais dados.

Dessa forma, foi definido que será obtido os valores de uso de CPU, memória, vazão de rede (tanto para entrada quanto saída), número de conexões simultâneas, tempo de resposta das requisições e latência entre cliente e serviço. Estes valores serão analisados conforme possíveis problemas conhecidos, definidos na seção de critérios (Seção 4.2).

A fim de obter tais valores para análise, o plano de testes definiu um ambiente baseado em camadas conforme a demanda das arquiteturas submetidas a análise. Foram projetadas cinco sub-redes no cenário de testes baseado nas regiões definidas no ambiente de testes. A partir destas sub-redes, foi estipulado valores de recursos computacionais limites e números de instâncias máximas. Entretanto, não foi possível definir o limite de instâncias para a sub-rede de estresse, visto que não foi encontrado estudos anteriores para ter uma linha base de consumo de recursos. Dessa forma, o atual trabalho contribuirá conjuntamente com futuros trabalhos guiando o consumo de recursos para as arquiteturas Rudy, Salz e Willson. Por fim, foram definidas as regras de negócio básicas para os clientes simulados, baseados em um autômato.

Os testes que serão executados sobre as redes estão baseados na finalidade de obtenção de valores de recursos mínimos para execução dos serviços e crescimento de valores de recursos conforme um número de clientes simultâneos.

## 5 Desenvolvimento das arquiteturas propostas

Após o levantamento e fichamento dos requisitos para execução dos experimentos sobre as arquiteturas MMORPG Rudy, Salz e Willson, é notório a necessidade de exemplificar o funcionamento de tais arquiteturas. Nesse sentido, o capítulo atual levanta as devidas considerações sobre o desenvolvimento prático das arquiteturas específicas a jogos MMORPG utilizadas neste trabalho.

Durante todos os experimentos, o ambiente é composto pelo mesmo conjunto de computadores, utilizando as mesmas topologias. Este tópico é abordado no capítulo atual com maior profundidade na Seção 5.1.

Além da padronização, componentes básicos não foram alterados para as arquiteturas, tais como o conjunto de computadores que hospedaram os bancos de dados e a arquitetura que armazenou as métricas para posterior análise. O sistema de coleta de informações de recursos é abordado na Seção 5.2.

A explicação técnica dos projetos estão organizadas da seguinte forma para cada serviço desenvolvido das arquiteturas:

1. Linguagem de Programação e tecnologias: Enumera as linguagens de programação e quais módulos ou bibliotecas foram utilizados para o seu desenvolvimento.
2. Protocolos Utilizados: Enumera protocolos utilizados pelos serviços.
3. Modelo de processamento: Enumera o seu modelo de processamento, podendo ser passivo as requisições do ecossistema ou ativamente processar dados da arquitetura.

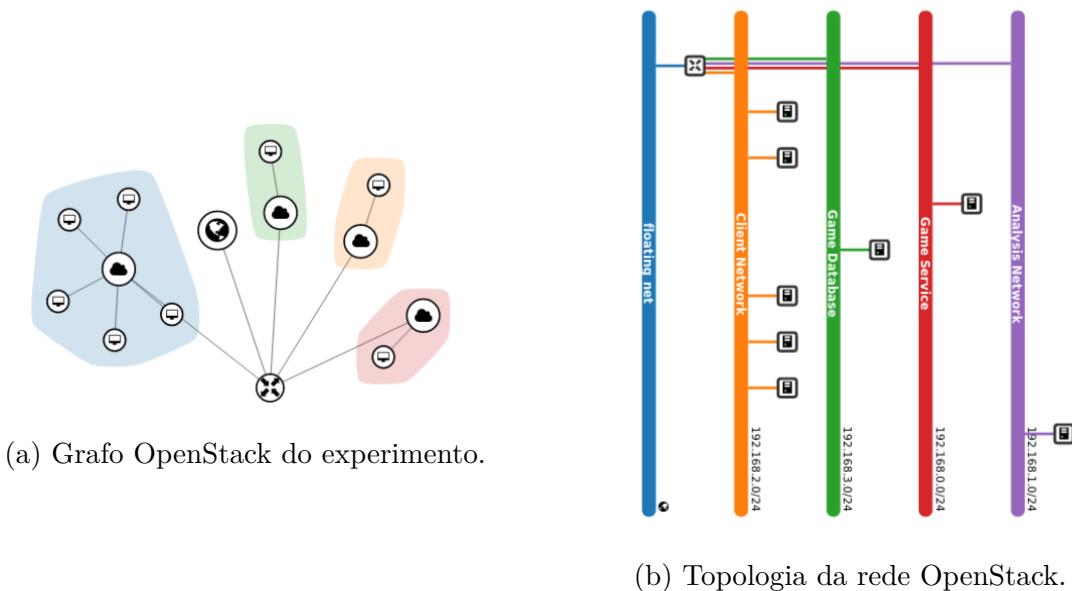
Ao final é levantado o funcionamento do cliente das arquiteturas Rudy (Subseção 2.5.1), Salz (Subseção 2.5.2) e Willson (Subseção 2.5.3). Nesta etapa é exibido o fluxo de mensagens a partir das ações realizadas pelo cliente. Nesse sentido este capítulo abrange todos os aspectos técnicos da aplicação, levando em consideração o seu fluxo de funcionamento e integração entre os microsserviços no ecossistema da arquitetura.

## 5.1 Topologia da Rede

A topologia da rede seguiu o padrão proposto para a coleta de dados. O principal objetivo desta topologia é segregar as instâncias na camada de rede, evitando interferências do orquestrador de microsserviços que é utilizado na etapa de implantação. A segregação em sub-redes é importante visto que, a fim de otimizar o transporte de dados entre os serviços, tais orquestradores trocam mensagens na rede utilizando memória ou disco, visto que uma conexão pode ser realizada para o próprio hospedeiro. Segregando em sub-redes, o orquestrador é forçado a realizar a conexão com as instâncias externas, forçando a utilização da pilha completa do protocolo TCP e UDP, a qual são eventuais pontos de gargalo em tais arquiteturas.

Para gerenciar as sub-redes foi utilizado o gestor da Nuvem Tchê, orquestrada com o OpenStack<sup>1</sup>. A topologia utilizada nos testes com as três arquiteturas podem ser visualizada na Figura 5.1.

Figura 5.1: Topologia da rede no gestor de redes do OpenStack.



Fonte: O próprio autor.

A Figura 5.1 mostra a disposição das instâncias em sub-redes exibindo-as em formato de árvore e em barras. A visualização com formato árvore ajuda a compreender a conexão entre cada máquina do ambiente de testes. Já a visualização em formato barra, exibe características técnicas como a divisão por *Internet Protocol* (IP) e nome da rede. Ambos os gráficos são complementares para formar as características básicas da

<sup>1</sup>OpenStack: <https://www.openstack.org/>

interconexão das sub-redes.

A Figura 5.1 também exibe 8 instâncias utilizadas nos testes. Cada instância pode ter configurações padrões de *Operating System* (OS), VCPU, memória. Além disso, o OpenStack permite a criação rápida de réplicas de uma instância. Dessa forma facilitando a configuração da rede final. As características específicas das instâncias estão listadas na Tabela 5.1.

Tabela 5.1: Instâncias das redes de teste

Nome	OS	VCPU	Memória	Rede ou IP	Réplicas
<i>client_*</i>	Ubuntu Server 16.04	1	1 GB	192.168.2.*	5
<i>game_service</i>	Ubuntu Server 16.04	4	8 GB	192.168.0.8	1
<i>game_database</i>	Ubuntu Server 16.04	1	1 GB	192.168.3.11	1
<i>data_analisisys</i>	Ubuntu Server 16.04	4	4 GB	192.168.1.2	1

Fonte: O próprio autor.

A Tabela 5.1 enumera as máquinas virtuais utilizadas no experimento, descrevendo as suas principais características. É utilizado as seguintes instâncias no experimento:

- *client\_\**: são responsáveis por executar os clientes a qual realizam o ataque de carga ao serviço.
- *game\_service*: é responsável unicamente por executar os microsserviços das arquiteturas Rudy, Salz e/ou Willson.
- *game\_database*: é responsável por executar o banco de dados Postgres e Redis, ambos utilizados em todas as arquiteturas.
- *data\_analisisys*: é responsável por executar o banco de métricas e o sistema de monitoramento.

Em especial, a categoria de instância *client\_\**, a qual executa os clientes, foi escalada em 5 instâncias. Este número foi definido com base em testes com a arquitetura Rudy, escalando máquinas virtuais dado a instância do serviço pré definido na Tabela 5.1. O menor valor de instâncias *client\_\** escaladas para alcançar o estresse da instância *game\_service* foi 5. Nesse sentido, será utilizado para todos os testes o valor de 5 instâncias para o ataque de carga de serviço.

A gestão das instâncias, dentro de cada sub-rede, foi automatizada com algum sistema de orquestração de microserviços disponível para o Docker, em nível de aplicação. Nesse sentido, a configuração inicial de cada instância foi feita de forma automatizada, com um *script* na linguagem *Bash* o qual configura o ambiente com a plataforma *Docker*<sup>2</sup>. A Tabela 5.2 enumera as versões do Docker instaladas na máquina e o orquestrador escolhido para operação.

Tabela 5.2: Orquestradores utilizados no teste.

Nome	Docker Engine	Docker Compose	Orquestrador
<i>client_*</i>	18.09	3.0	Docker Swarm
<i>game_service</i>	18.09	3.0	Docker Swarm
<i>game_database</i>	18.09	3.0	Docker Compose
<i>data_analisis</i>	18.09	3.0	Docker Compose

Fonte: O próprio autor.

Em específico neste projeto, foram utilizados os orquestradores Docker Swarm e Docker Compose (Tabela 5.2). Estes dois orquestradores facilitam, respectivamente, a implantação de arquiteturas complexas sem relacionamento com o disco rígido e o controle de volumes para armazenamento temporário de dados. Dessa forma, os serviços implantados nas instâncias *game\_service* e *client\_\** são gerenciados pelo Docker Swarm, visto que ambos não necessitam de acesso a disco e podem ser escalados para mais de uma instância. Por sua vez, os serviços restantes precisam de acesso a disco. Dessa forma, estes foram implantados utilizando o Docker Compose.

Em especial, sistema de coleta de métricas é executado na instância *data\_analisis* e foi implantado utilizando Docker Compose. Esta instância é uma peça fundamental para o sucesso do atual trabalho, haja visto que toda a captura de métricas é armazenada e processada nesta instância. Assim, faz-se necessário detalhar o funcionamento deste sistema computacional.

## 5.2 Sistema de coleta de informações de recursos

O sistema de coleta de informações tem como principal objetivo receber requisições do estado da aplicação ou instância. Em específico para esta aplicação, o sistema de métricas relacionará um montante (GBs, MBs, Tempo de Requisição) em relação ao tempo, formando gráficos os quais podem ser comparados entre si dentro de um período.

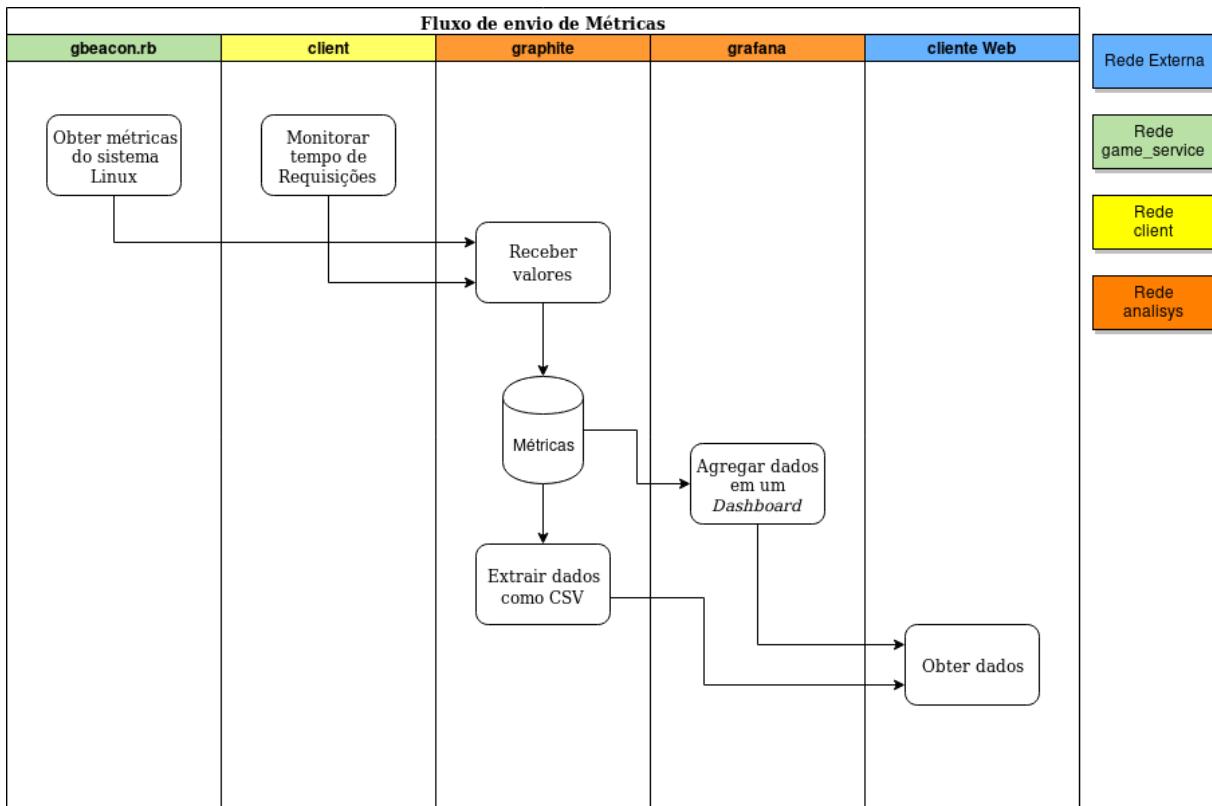
<sup>2</sup>Docker: <https://www.docker.com/>

Para esta solução, a pilha de monitoramento escolhida é o banco de dados para métricas Graphite<sup>3</sup> e o gestor web para visualização Grafana<sup>4</sup>. Estas soluções foram escolhidas por afinidade, facilidade de uso e por serem ferramentas de uso em ambientes reais para este objetivo.

Utilizando o Grafana, é possível verificar e comparar períodos diretamente por seu sistema web, durante a execução dos testes. Isto permite maior flexibilidade e segurança da obtenção de dados, na qual são monitorados de forma facilitada pelo operador dos testes em seu *Dashboard*.

O Grafana exibirá os dados armazenados no banco de dados Graphite, na qual desempenha papel de banco de dados específico para armazenamento de métricas. Nesse sentido, este é capaz de otimizar a transferência de informações a qual são enviadas tanto do serviço de jogo e do cliente. O fluxo de transferência é exibido na Figura 5.2.

Figura 5.2: Fluxo dos valores obtidos no cliente e serviço.



Fonte: O próprio autor

Como exibido na Figura 5.2, existem dois tipos de monitores no qual populam os dados automaticamente no sistema de monitoramento. São eles o monitor de recurso e o monitor de tempo de requisição.

<sup>3</sup>Graphite DB: <https://graphiteapp.org/>

<sup>4</sup>Grafana: <https://grafana.com/>

O monitor de recurso é utilizado no *game\_service* na qual envia métricas do consumo de recursos pela instância. Este monitor foi implementado para este TCC na linguagem Ruby e é distribuído como uma aplicação completa para monitoramento de sistemas baseados em Debian. O mesmo possui a automatização de implantação do servidor Graphite e Grafana utilizando Docker Compose em seu repositório <sup>5</sup>.

Durante os testes é executado o monitor de recurso na arquitetura do *game\_service*, a qual enviará as métricas obtidas do sistema operacional para o servidor Graphite indicado. Este monitor obtém os recursos programados da máquina hospedeira, a qual estará hospedando o serviço docker da arquitetura de microsserviços em teste. Dessa forma temos o monitoramento de recursos em tempo de execução, levando em consideração o custo de gerenciamento do Docker. Entretanto, o seu consumo para gestão é desrespeitável para tal experimento.

Por sua vez, o monitor de tempo de resposta é implementado diretamente no cliente, a qual captura o tempo das requisições do serviço do jogo. Ele foi isolado em um pacote na linguagem Golang, na qual pode ser reutilizado em futuros projetos <sup>6</sup>.

Por sua vez, os dados são analisados tanto nos gráficos padrões de monitoramento do grafana, quanto pós processados por outras ferramentas como o Gnuplot <sup>7</sup>, na qual são obtidos através da API de *render* <sup>8</sup> do Graphite. Dessa forma também existe a flexibilidade de obter métricas para estudos estatísticos externos a pilha padrão do Graphite/Grafana.

## 5.3 Arquitetura Rudy

A arquitetura Rudy (Subseção 2.5.1), a primeira arquitetura desenvolvida, teve o seu funcionamento reduzido aos microsserviços básicos para o funcionamento do Gerente de Mundo. Nesse sentido, os microsserviços implementados para esta arquitetura foram:

1. Serviço de Jogo: Ou *rudygh*.
2. Gerenciador de Consultas: Ou *rudydb*.

---

<sup>5</sup>gbeacon.rb: [github.com/schweigert/grafana-beacon/](https://github.com/schweigert/grafana-beacon/) e rubygems.org/gems/gbeacon

<sup>6</sup>metric package: <https://github.com/schweigert/mga/tree/master/libraries/metric>

<sup>7</sup>Gnuplot: <http://www.gnuplot.info/>

<sup>8</sup>Graphite render API: [https://graphite.readthedocs.io/en/latest/render\\_api.html](https://graphite.readthedocs.io/en/latest/render_api.html)

3. Serviço de Autenticação: Ou *rudyauth*.
4. Serviço Web Dinâmico: Ou *rudyweb*.

Além destes microsserviços, a arquitetura utiliza os serviços PostgreSQL e Redis, ambas de código aberto. Tais serviços são utilizados respectivamente como banco de dados permanente e banco de dados em memória cache para autenticação. Este método de autenticação foi replicado para as demais arquiteturas.

Em relação aos protocolos utilizados na arquitetura Rudy, esta arquitetura utiliza serviços com o protocolos RPC e HTTP. A relação detalhada é exibida na Tabela 5.3.

Tabela 5.3: Protocolos dos microsserviços da arquitetura Rudy.

Microsserviço	Linguagem de Programação	Porta	Protocolo
rudygh	Golang 1.11 / RPC Nativo	3000	TCP
rudydb	Golang 1.11 / Gin Framework	3000	HTTP
rudyauth	Golang 1.11 / RPC Nativo	3000	TCP
rudyweb	Golang 1.11 / Gin Framework	3000	HTTP

Fonte: O próprio autor.

Além dos protocolos utilizados, a Tabela 5.3 relaciona a linguagem / tecnologia utilizados para escrever o serviço. Tanto nos microsserviços da arquitetura Rudy, quanto aos demais microsserviços implementados, foram escritos na linguagem Go, evitando a repetição de códigos compartilhados entre eles. Ambos os serviços compartilham dos mesmos modelos de padrão MVC tanto para as aplicações RPC e HTTP.

Para o desenvolvimento dos microsserviços da arquitetura Rudy, foram utilizados os seguintes pacotes externos da linguagem:

- Gin: Framework focado em desenvolvimentos de API para web escrito em Golang.
- Redis: Pacote de conexão sobre TCP ou UDP em um serviço Redis.
- Protobuf: Pacote que implementa serialização de dados estruturados de modo mínimo. É utilizado para minimizar custo de chamadas RPC em Go.
- Gorm: Golang *Object-Relational Mapping* (ORM) que permite a conexão com o banco de dados Postgres sem utilizar SQL.

- Graphite: Pacote que permite a conexão com o servidor de logs Graphite para enviar métricas.
- Gorequest: Pacote para estruturar requisições Web utilizado para comunicação interna entre serviços Web utilizando assinatura JWT.
- Testify: Suíte de testes para manter o funcionamento íntegro durante a implementação da aplicação. Em específico, a aplicação possuí 90% de convergência de código, a qual garante a integridade de futuras alterações.

A pilha de desenvolvimento utilizando Golang 1.11 deve-se pela simplicidade de escrita de código e eficiência (linguagem compilada e tipada). Além de sua eficiência computacional, existe uma grande comunidade que apoia o desenvolvimento de microserviços utilizando Golang, a qual contribui continuamente para desenvolver bibliotecas para suas soluções. Nesse sentido, todas as bibliotecas utilizadas são distribuídas no modelo *Open Source*, pelas licenças *MIT*, *BSD* e *GPLv3*. Estas mesmas bibliotecas são utilizadas nas demais arquiteturas.

## 5.4 Dados obtidos da arquitetura Rudy

Para obter os dados da arquitetura Rudy, foram efetuadas 3 testes diferentes. Em específico, a diferença entre cada teste é o crescimento de conexões por minuto. Os testes foram automatizados da seguinte maneira:

- Utilizando uma conta nova por minuto.
- Utilizando duas contas novas por minuto.
- Utilizando cinco contas novas por minuto.

Para cada ítem citado, a cada minuto, é escalonado um novo cliente robô que realizará as seguintes ações:

- Criar uma conta.
- Autenticar a conta.

- Criar um personagem.
- Selecionar personagem.
- Movimentar-se pelo jogo, enviar mensagens e receber mensagens.

Dentro deste cenário, espera-se estressar a vazão da rede ou CPU, visto que são poucos dados para armazenamento em memória do serviço de jogo. Vale ressaltar que a latência da rede não foi prejudicada durante o teste, tornando-se constante, como já esperado.

#### 5.4.1 Consumo de CPU pelo serviço Rudy

Um dos objetivos da coleta de CPU da arquitetura Rudy é analisar pontos de gargalo. Em especial a arquitetura Rudy tem uma abordagem serial das requisições, a qual serve para evitar a concorrência das estruturas de dados entre múltiplos jogadores.

Esta abordagem é comum para jogos onde a frequência de atualizações é menor, visto que seu principal fator limitante é a fila de requisições. Dessa forma, espera-se obter um teto limite de CPU durante a análise, a qual afetará diretamente o consumo da banda e tempo de resposta dos clientes.

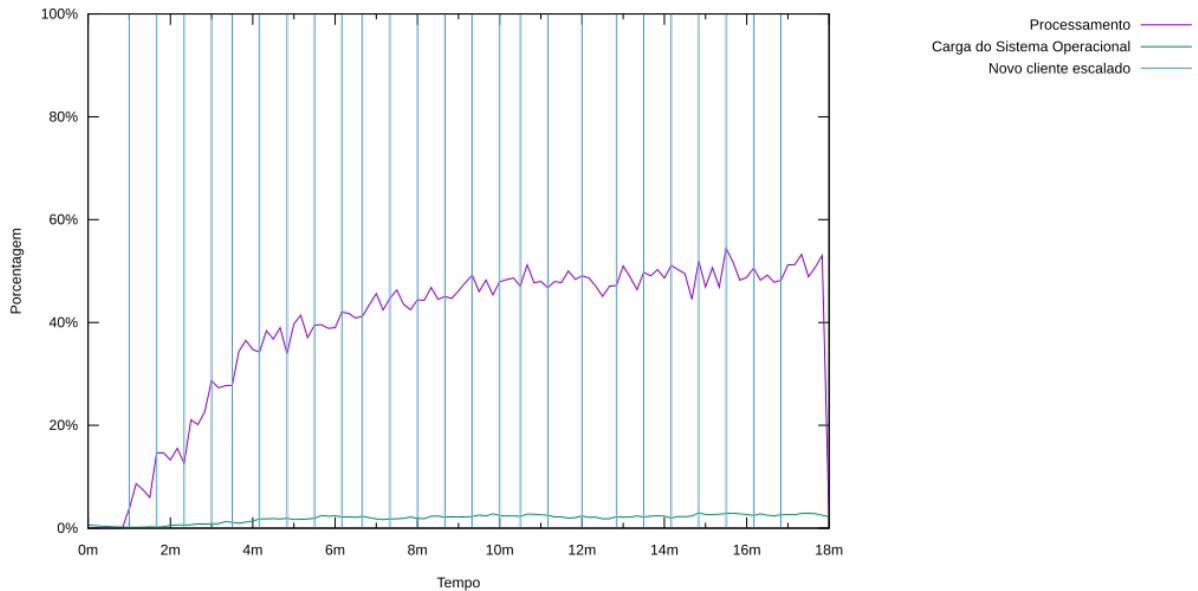
As variáveis relacionadas a este teste são o número de conexões. Na Figura 5.3 exibe a carga da CPU.

#### 5.4.2 Consumo de Memória pelo serviço Rudy

O principal objetivo da coleta de memória da arquitetura Rudy é analisar possíveis vulnerabilidades a qual podem ser sanadas com a escalabilidade do sistema. Em geral, jogos MMORPG permitem a divisão de suas regiões em *chunks*, a qual são distribuídos entre os serviços. Esta divisão de carga também é aplicada na arquitetura Rudy. Dessa forma, espera-se que a memória seja impactada pelo número de conexões simultâneas em um único pedaço do mundo, consumindo memória para processamento do protocolo RPC e armazenamento dos dados do cliente e do mundo, a qual consomem recursos conforme a regra de negócio.

Outro ponto de consumo da memória é referente ao consumo da pilha de execução das chamadas web. Em geral, esta aplicação só consumirá memória especificamente

Figura 5.3: Consumo de CPU no servidor utilizando a arquitetura Rudy



Fonte: O próprio autor

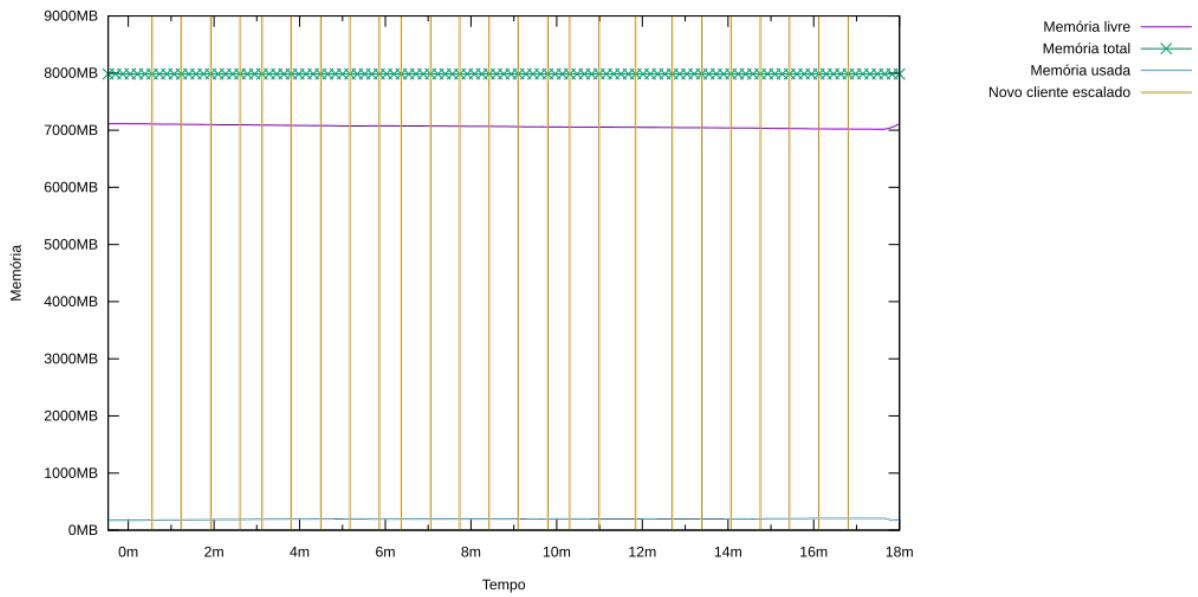
durante a requisição, no qual armazenará o estado no banco de dados, não precisando executar outras operações para manutenção da memória local. Nesse sentido, este teste tende a ter um baixo consumo de memória, visto que as regras de negócio são simples e a memória tende a ser linearmente consumida conforme o número de conexões simultâneas.

O teste executado na arquitetura obteve os dados que condizem com a expectativa dos testes, obtendo um baixo consumo de recursos por parte das aplicações da arquitetura Rudy. A Figura 5.4 compara a memória total com a memória utilizada pela arquitetura.

A comparação exibida na Figura 5.4 mostra somente a escala entre a memória total do serviço com relação a memória consumida. Porém, esta visualização fica comprometida, visto que não fica perceptível a baixa variação da memória consumida com relação ao número de conexões no serviço. A Figura 5.5 trata a escala e aplica somente os dados da memória utilizada pela arquitetura.

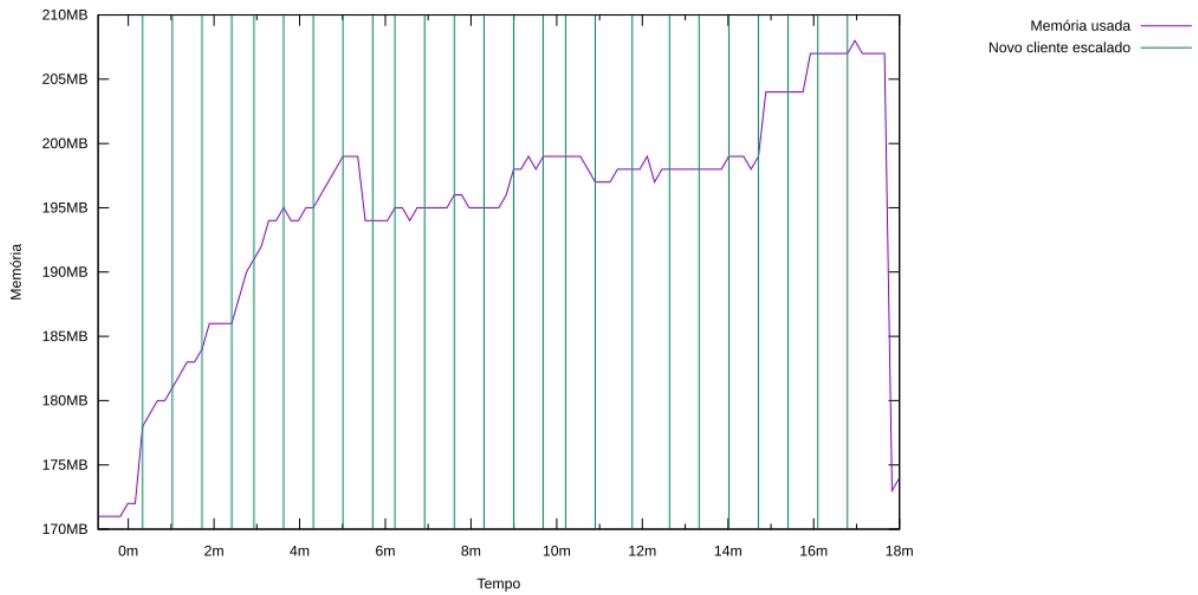
Após o tratamento, torna-se notório o crescimento da memória conforme o número de conexões, utilizando uma alíquota de valor pequeno, se comparado ao recurso disponível. Nesse sentido, não foi possível estressar a memória do serviço como um recurso crítico para os testes, porém isto não invalida estes dados para futura comparação com as demais arquiteturas.

Figura 5.4: Registro de memória no servidor utilizando a arquitetura Rudy



Fonte: O próprio autor

Figura 5.5: Memória consumida no servidor utilizando a arquitetura Rudy



Fonte: O próprio autor

### 5.4.3 Consumo de Banda pelo serviço Rudy

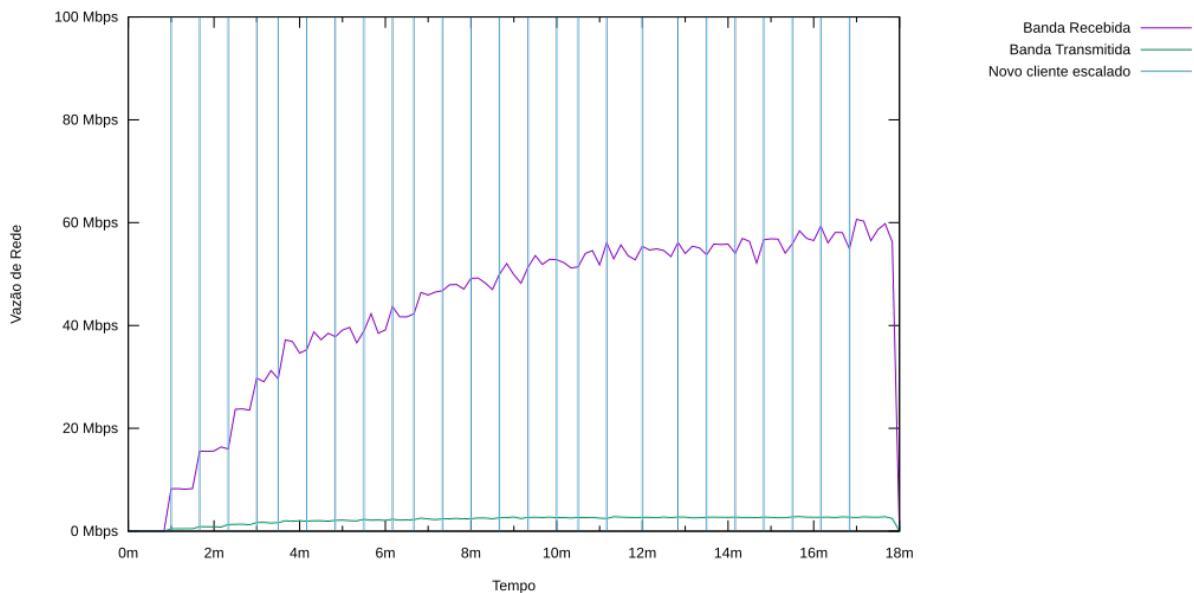
O principal objetivo do monitoramento do consumo da banda pelo serviço Rudy é verificar se a rede está enfileirando as requisições nos serviços RPC na arquitetura. Esta validação é feita caso a rede tenha um limite notório na entrada e saída da rede e o aumento do tempo de requisição, sem comprometer a CPU do serviço.

Caso as requisições sejam enfileiradas internamente pelo microsserviço RPC

(Gerente de Jogo), como o modelo de processamento de requisições da arquitetura Rudy propõe, espera-se que o tempo de resposta que aumente. O aumento do tempo de resposta é um reflexo da concorrência para escrever a requisição RPC na fila de processamento do serviço, a qual dependerá do escalonador de threads escolher qual processo escreverá a sua requisição na fila.

Com os dados da banda da arquitetura Rudy obtidos do teste, pode-se observar que o crescimento do consumo da rede não é linear. Os dados podem ser visualizados na Figura 5.6.

Figura 5.6: Consumo de Banda no servidor utilizando a arquitetura Rudy



Fonte: O próprio autor

Como observado na Figura 5.6, existe um limite de crescimento da banda. Com as configurações do ambiente, este valor tende a 60Mbps. Vale ressaltar que este gargalo não é proveniente da rede (a qual ficou limitada em 100Mbps), visto que a métrica de latência ficou constante durante os testes. Dessa forma, esta tendência pode ser pela concorrência a fila de processamento, a qual espera-se confirmar com o tempo de resposta do cliente.

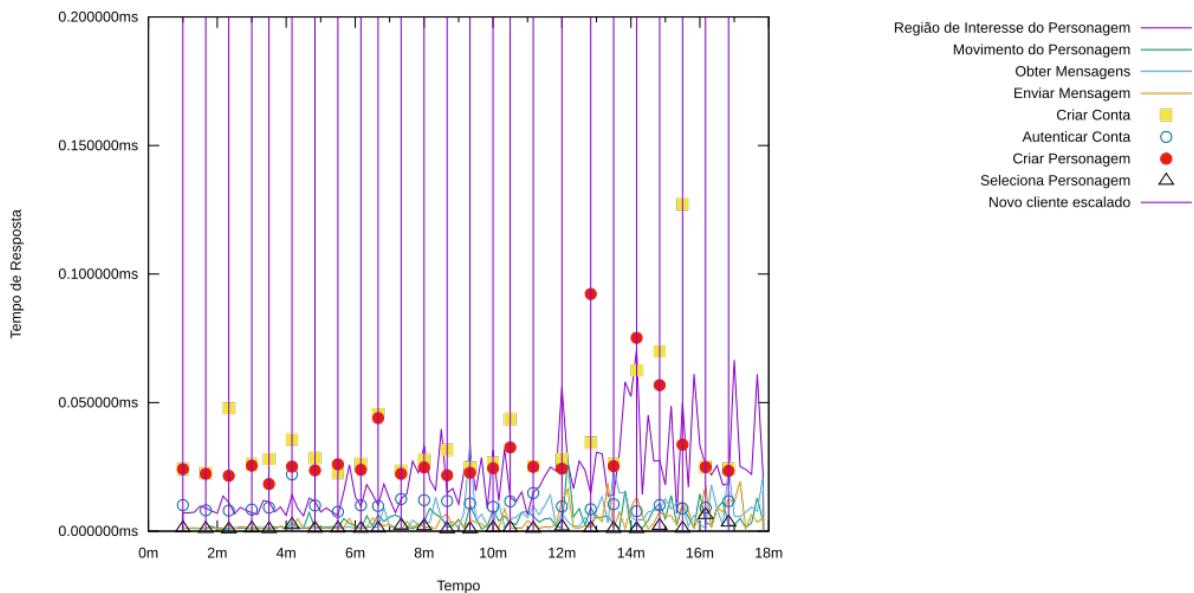
#### 5.4.4 Tempo de Resposta pelo cliente Rudy

Para a arquitetura Rudy, o tempo de resposta do cliente irá confirmar se o enfileiramento das requisições RPC ocorreram como previsto. Nesse sentido, espera-se que o tempo de requisição aumente de forma superlinear conforme o número de clientes aumente.

Além disso, será possível visualizar o impácto de operações realizadas no banco com o tempo de resposta do serviço, visto que ao efetuar uma operação no banco a mensagem será transmitida pelos microsserviços web dinâmico e pelo gestor do banco de dados, além do banco de dados por fim. Este impácto chamadas sobre o protocolo HTTP com múltiplas camadas deve ser notório conforme a demanda de usuários, atrapalhando o gestor de rede com o consumo de CPU e rede para atende-los.

Para comparação entre as requisições RPC e web, a Figura 5.7 exibe ambas em uma única Figura. Espera-se mostrar o crescimento superlinear das requisições RPC, a qual chegam a ultrapassar o tempo de resposta das requisições HTTP com múltiplas camadas.

Figura 5.7: Tempo de Resposta do cliente servidor utilizando a arquitetura Rudy

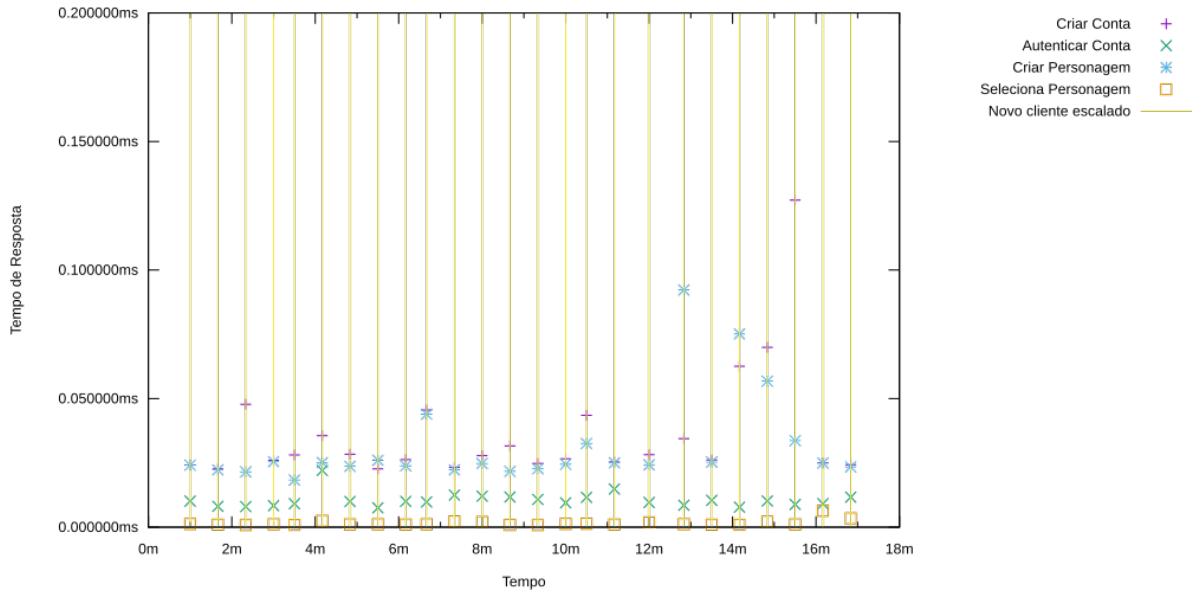


Fonte: O próprio autor

A Figura 5.7 mostra, dentro de um intervalo de tempo, o tempo de resposta categorizado conforme a ação realizada pelo cliente no serviço. Para facilitar a visualização, este gráfico foi dividido em duas categorias, segregando as requisições sobre HTTP e sobre RPC. A Figura 5.8 exibe o tempo de resposta das requisições web em relação ao tempo. Mostra-se na figura a dispersão que ocorre nos tempos de requisições sobre o protocolo HTTP conforme o recurso do serviço é consumido por novos clientes.

A Figura 5.8 exibe uma dispersão conforme o número de conexões do serviço aumenta. Entretanto, espera-se que este tempo de requisição sobre o protocolo HTTP aumente somente como um reflexo do consumo de CPU e banda pelos clientes. Por este

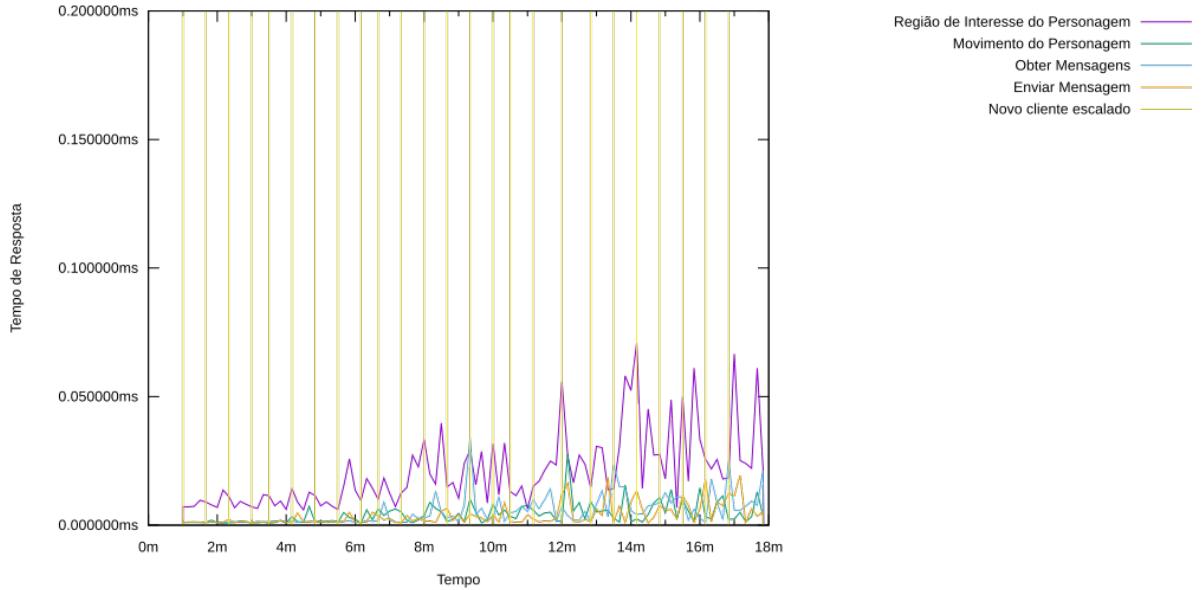
Figura 5.8: Tempo de Resposta de requisições Web do cliente servidor utilizando a arquitetura Rudy



Fonte: O próprio autor

motivo, é notório a necessidade de exibir a parte o tempo de requisição das chamadas RPC. Os dados do protocolo RPC podem ser visualizados na Figura 5.9.

Figura 5.9: Tempo de Resposta de requisições RPC do cliente servidor utilizando a arquitetura Rudy



Fonte: O próprio autor

A Figura 5.9 exibe o desparelhamento provocado pelo sistema de filas implementado nesta arquitetura nos serviços RPC. Percebe-se também que esta disparidade aumenta diretamente com as requisições HTTP ao escalar um novo cliente. Estas infor-

mações indicam que de fato ocorre o enfileiramento das requisições, além disso a carga das requisições HTTP agravam o enfileiramento, aumentando显著emente o tempo de resposta durante os próximos segundos após a chamada HTTP.

A partir dos dados obtidos das requisições HTTP, exibidos na Figura 5.8, percebe-se que os valores exercem baixa influência conforme a carga de usuários utilizando o serviço. Nesse sentido, é possível agrupar todos os dados a fim de obter a sua média, mediana, valor máximo e mínimo das requisições sobre o protocolo HTTP. Estes dados estão dispostos na Tabela 5.4.

Tabela 5.4: Média, mediana, máximo e mínimo obtidos pelas requisições HTTP.

Requisição HTTP	Máximo	Mínimo	Média	Mediana
Criar conta	0,221395227 s	0,020872847 s	0,039408472375 s	0,0292280260 s
Autenticar Conta	0,207815276 s	0,007954407 s	0,019106303125 s	0,0106579220 s
Criar Personagem	0,090555727 s	0,019188871 s	0,027868142025 s	0,0250552605 s
Selecionar Personagem	0,010201612 s	0,000752974 s	0,002188604100 s	0,0021886041 s

Fonte: O próprio autor.

A Tabela 5.4 comprova a estabilidade das requisições HTTP, na qual a média, mediana e o valor mínimo estão próximos para todos os tipos de requisição. Nota-se pela Requisição HTTP Selecionar Personagem, na qual realiza a busca dos dados no banco em paralelo a requisição, a maior estabilidade e eficiência dentre as demais citadas. Nesse sentido, torna-se visível que a instabilidade, por mais que baixa, esteja relacionada com a busca ou inserção de dados nos bancos da arquitetura.

As requisições RPC também podem apresentar características similares. Entretanto, as requisições RPC concorrem com as requisições HTTP entre os recursos do serviço. Nesse sentido, torna-se interessante obter a média, mediana, máximo e mínimo dos tempos de requisição levando em consideração o número de clientes escalados. A Tabela ?? exibe a média, mediana, máximo e mínimo, segregando conforme o número de conexões do serviço.

## 6 Considerações & Próximos passos

Este TCC tem como objetivo levantar questões relacionadas a complexidade de coordenação de arquiteturas de microsserviços para jogos MMORPG. Estas arquiteturas são complexas devido a quantidade de módulos e serviços necessários que visam suprir a demanda do mercado para esse gênero de jogo. Sendo assim, para melhor descrever as características de tais arquiteturas, este trabalho propõe a realização de uma análise sobre três arquiteturas distintas descritas na literatura.

Dentre as três arquiteturas selecionadas, é explícito o objetivo a qual cada arquitetura foi planejada. O principal objetivo da arquitetura Rudy é normalizar conexões de diferentes qualidades com o seu modelo de paralelização de requisições. Já para a arquitetura Salz o seu objetivo é escalabilidade para jogadores simultâneos em um único ambiente. A arquitetura Willson tem como objetivo reduzir o tempo de resposta aos clientes. Porém, este objetivo não é descrito de forma clara e objetiva pelos autores, os quais desenvolveram tais arquiteturas dentro de cenários de demanda específicos do mercado. Dessa forma, espera-se encontrar padrões de valores que venham de encontro com tais objetivos.

Com relação aos testes, os valores de recurso para o cenário não podem ser estipulados. Dessa forma, foi definido um teto a fim de limitar o consumo das arquiteturas. Porém, espera-se não utilizar o limite destes recursos, tornando o cenário de testes mais simples. Caso o consumo de recursos seja menor ao limite, todas as arquiteturas serão testadas utilizando o mesmo cenário.

Com relação aos testes, faz necessário verificar na totalidade se será viável desenvolver, implantar e testar todas as arquiteturas no período de tempo estipulado. Dessa forma, podem haver alterações para o TCC II. Espera-se para o TCC II projetar, desenvolver, implantar e analisar os dados, conforme o cronograma estipulado.

O atual trabalho, iniciado no primeiro semestre de 2018, atingiu seus objetivos. Dentre as atividades estipuladas para a primeira fase do TCC, todas foram concluídas, porém não dentro do prazo estipulado de um semestre.

As principais considerações acerca da pesquisa referenciada são relevantes as

características de projeto de um serviço distribuído em microsserviços para atender jogos MMORPG. Elas podem ser enumeradas da seguinte forma:

- Preocupação de consumo de recursos e escalabilidade para muitos jogadores, evitando a segregação de jogadores em seu ambiente;
- Preocupação dos autores das arquiteturas acerca do processamento em relação ao escalonador de processos do sistema operacional, cache de dados e/ou requisições e modelo de processamento paralelo das requisições, visando alto desempenho das arquiteturas; e
- Preocupação dos desenvolvedores das arquiteturas com o funcionamento independente do cliente, fomentando o desenvolvimento multiplataforma do gênero.

Estas preocupações ampliam a utilidade do atual trabalho. Entretanto, o conteúdo encontrado para a pesquisa referenciada dificultou o processo de desenvolvimento da referenciação teórica. Em específico, as dificuldades encontradas durante a elaboração da primeira parte do atual trabalho foram:

- Dificuldade de encontrar material científico ou acadêmico correlacionando arquiteturas de microsserviços e arquiteturas para jogos MMORPG.
- Os autores das arquiteturas especificam com exatidão determinados pontos relevantes ao projeto, mas não foi identificado um aprofundamento para determinados microsserviços da arquitetura. Dessa forma foi necessário realizar buscas na literatura a fim de viabilizar tais projetos, seguindo as especificações das arquiteturas.

Além das dificuldades com relação direta aos objetivos atuais deste trabalho, houve a necessidade de realizar um estudo sobre as tecnologias que permeiam o processo de implantação, desenvolvimento e testes de arquiteturas de microsserviços. Dessa forma espera-se utilizar a tecnologia *Docker Swarm* para implantação das arquiteturas, sobre uma nuvem de computadores *OpenStack*. A fim de desenvolver de forma ágil as arquiteturas, será utilizada a linguagem de programação *GoLang*, a qual se propõe ser rápida (compilada, fortemente tipificada, executado como código nativo), a qual inclui uma ampla biblioteca padrão para conexão com banco de dados e protocolo TCP, podendo

facilmente ser implantada utilizando a tecnologia *Docker*. A resolução destas dificuldades na atual etapa facilitará o desenvolvimento do TCC II.

Para o TCC-II serão desenvolvidas as arquiteturas de microsserviços para jogos MMORPG descritas na proposta do atual trabalho. Também serão desenvolvidos testes de carga automatizados a fim de facilitar a etapa de testes das arquiteturas. Por fim, os dados serão coletados por um sistema de análise de recursos desenvolvido no TCC-II, durante os testes, serão analisados a fim de gerar uma análise das arquiteturas de microsserviços para jogos MMORPG que levantará questões de desempenho presentes nela.

## 6.1 Cronograma

Até o presente momento, o cronograma proposto no Plano do TCC foi seguido conforme o previsto, na ordem e nos prazos estipulados. As etapas presentes nesta seção foram definidas no Plano de TCC para atingir os objetivos propostos.

## 6.2 Etapas realizadas

1. **Levantamento e fichamento das referências:** Pesquisa de fontes para embasamento teórico do trabalho, com base nos objetivos específicos;
2. **Consolidação das referências:** Compreensão e seleção de artefatos literários que permitam atingir o objetivo do Trabalho de Conclusão de Curso I;
3. **Identificação e definição de arquiteturas descritas na literatura:** Enumeração e definição das arquiteturas de microsserviços descritas na literatura, bem como os seus objetivos;
4. **Especificação das arquiteturas selecionadas:** Especificação do funcionamento das arquiteturas selecionadas.
5. **Identificação e definição de simulações aplicáveis ao teste:** Seleção e definição da simulação de clientes a ser aplicada nos testes;
6. **Especificação da simulação elegida:** Especificação dos requisitos necessários para a simulação dos clientes aos serviços escolhidos;

## 7. Escrita Trabalho de Conclusão de Curso I;

### **6.3 Etapas a realizar**

- 8. Desenvolvimento da simulação:** Desenvolvimento da simulação para interagir com as arquiteturas de microsserviços;
  - 9. Desenvolvimento da arquitetura:** Desenvolvimento da arquitetura para executar os testes;
  - 10. Aplicação das arquiteturas selecionadas na pesquisa referenciada:** Aplicação das arquiteturas descritas sobre uma nuvem computacional;
  - 11. Realização dos testes utilizando a simulação elegida na pesquisa referenciada:** Execução de testes da arquitetura desenvolvida sobre a nuvem computacional;
  - 12. Análise das arquiteturas testadas:** Análise das métricas obtidas dos testes e descrever resultados, identificando possíveis gargalos nas arquiteturas;
  - 13. Otimização para melhorar as métricas obtidas:** Identificação dos pontos de gargalo nos microsserviços selecionados e proposta de soluções viáveis para aumento do desempenho desses sistemas.
  - 14. Escrita Trabalho de Conclusão de Curso II;**

## 6.4 Execução do cronograma

## Referências

- ACEVEDO, C. A. J.; JORGE, J. P. G. y; PATIÑO, I. R. Methodology to transform a monolithic software into a microservice architecture. In: *2017 6th International Conference on Software Process Improvement (CIMPS)*. Zacatecas, Mexico: IEEE, 2017. v. 1, n. 17417259, p. 1–6.
- ADAMS, A. R. E. *Fundamentals of Game Design (Game Design and Development Series)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 0131687476.
- ADAMS, E. *Fundamentals of Game Design*. New Riders Publishing, 2014. ISBN 978-032192967-9. Disponível em: <<https://www.amazon.com.br/Fundamentals-Game-Design-Ernest-Adams/dp/0321929675>>.
- BARRI, I.; GINE, F.; ROIG, C. A hybrid p2p system to support mmorpg playability. In: *2011 IEEE International Conference on High Performance Computing and Communications*. Banff, Alberta, Canada: IEEE, 2011. p. 569–574.
- BILTON, N. *Search Bits SEARCH Video Game Industry Continues Major Growth, Gartner Says*. 2011. Acessado em: 19/01/2018. Disponível em: <<https://bits.blogs.nytimes.com/2011/07/05/video-game-industry-continues-major-growth-gartner-says/>>.
- BLIZZARD. *StarCraft II*. 2010. [Online; Acessado em: 8. Maio 2018]. Disponível em: <<https://starcraft2.com/en-us>>.
- BORELLA, M. Research note: Source models of network game traffic. v. 23, p. 403–410, 02 2000.
- BUCHINGER, D. *Sherlock Dengue 8: The Neighborhood - Um jogo sério colaborativo-cooperativo para combate à dengue*. 2014. Online; Acessado em: 17. Apr. 2018. Disponível em: <[http://www.udesc.br/arquivos/cct/id\\_cpmenu/1024-diego\\_buchinger\\_1\\_15167055468902\\_1024.pdf](http://www.udesc.br/arquivos/cct/id_cpmenu/1024-diego_buchinger_1_15167055468902_1024.pdf)>.
- CHADWICK, J.; SNYDER, T.; PANDA, H. *Programming ASP.NET MVC 4: Developing Real-World Web Applications with ASP.NET MVC*. O'Reilly Media, 2012. ISBN 978-144932031-7. Disponível em: <<https://www.amazon.com/Programming-ASP-NET-MVC-Developing-Applications/dp/1449320317>>.
- CLARKE, R. I.; LEE, J. H.; CLARK, N. Why Video Game Genres Fail: A Classificatory Analysis. *SURFACE*, 2015.
- CLARKE-WILLSON, S. *Guild Wars 2: Scaling from One to Millions*. 2013. [Online; Acessado em: 1. Setembro 2018]. Disponível em: <<https://archive.org/details/GDC2013Willson>>.
- CLARKE-WILLSON, S. *Guild Wars Microservices and 24/7 Uptime*. 2017. Disponível em: <[http://twvideo01.ubm-us.net/o1/vault/gdc2017/Presentations/Clarke-Willson\\\_Guild Wars 2 microservices.pdf](http://twvideo01.ubm-us.net/o1/vault/gdc2017/Presentations/Clarke-Willson\_Guild Wars 2 microservices.pdf)>.

- DEZA, E. D. M. M. *Encyclopedia of Distances*. Springer, 2009. ISBN 978-364200233-5. Disponível em: <<https://www.amazon.com/Encyclopedia-Distances-Michel-Marie-Deza/dp/3642002331>>.
- DICE. *Battlefield*. 2013. [Online; Acessado em: 8. Maio 2018]. Disponível em: <<https://www.battlefield.com/pt-br>>.
- EA. *FIFA 18 - Soccer Video Game - EA SPORTS Official Site*. 2018. [Online; Acessado em: 8. Maio 2018]. Disponível em: <<https://www.easports.com/fifa>>.
- Exit Games. *Photon Unity Networking*. 2017. [Online; Acessado em: 15. Maio 2018]. Disponível em: <<https://doc-api.photonengine.com/en/pun/current/index.html>>.
- Exit Games. *Serialization in Photon Engine*. 2018. [Online; Acessado em: 29. Agosto 2018]. Disponível em: <<https://doc.photonengine.com/en-us/realtimedev/current/reference/serialization-in-photon>>.
- FARBER, J. Network game traffic modelling. In: *Proceedings of the 1st Workshop on Network and System Support for Games*. New York, NY, USA: ACM, 2002. (NetGames '02), p. 53–57. ISBN 1-58113-493-2. Disponível em: <<http://doi.acm.org/10.1145/566500.566508>>.
- FRANCESCO, P. D. Architecting microservices. *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, IEEE, Gothenburg, Sweden, v. 1, p. 224–229, Apr 2017.
- GOLDSMITH, T. "Cathode-ray tube amusement device". 1947. "Online; Acessado em: 15. Apr. 2018". Disponível em: <<https://patents.google.com/patent/US2455992>>.
- GUINNESS. *Greatest aggregate time playing an MMO or MMORPG videogame (all players)*. 2013. [Online; Acessado em: 23. Apr. 2018]. Disponível em: <<http://www.guinnessworldrecords.com/world-records/most-popular-free-mmorpg>>.
- HANNA, P. *Video Game Technologies*. 2015. Acessado em: 19/01/2018. Disponível em: <<https://www.di.ubi.pt/~agomes/tjv/teoricas/01-genres.pdf>>.
- HOWARD, E. et al. Cascading impact of lag on quality of experience in cooperative multiplayer games. In: *2014 13th Annual Workshop on Network and Systems Support for Games*. Nagoya, Japan: IEEE, 2014. v. 1, n. 14852575, p. 1–6. ISSN 2156-8138.
- HUANG, G.; YE, M.; CHENG, L. Modeling system performance in mmorpg. In: *IEEE Global Telecommunications Conference Workshops, 2004. GlobeCom Workshops 2004*. Northwestern University, USA: IEEE, 2004. v. 1, p. 512–518.
- HUANG, J.; CHEN, G. Digital stb game portability based on mvc pattern. In: *2010 Second World Congress on Software Engineering*. Wuhan, China: IEEE, 2010. v. 2, n. 11836781, p. 13–16. ISSN 978-1-4244-9287-9.
- IKEM, O. V. How We Solved Authentication and Authorization in Our Microservice Architecture. *Initiate*, Initiate, May 2018. Disponível em: <<https://initiate.andela.com/how-we-solved-authentication-and-authorization-in-our-microservice-architecture-994539d1b6e6>>.
- Internet Society. *XDR: External Data Representation Standard*. 2006. [Online; Acessado em: 19. Maio 2018]. Disponível em: <<https://tools.ietf.org/html/rfc4506.html>>.

- JAJEX. *RuneScape Online Community*. 2018. Acessado em: 01/02/2018 às 01:05. Disponível em: <<http://www.runescape.com/community>>.
- JON, A. A. The development of mmorpg culture and the guild. v. 25, p. 97–112, 01 2010.
- JONES, M. et al. *JSON Web Token (JWT)*. 2015. [Online; Acessado em: 1. Maio 2018]. Disponível em: <<https://tools.ietf.org/html/rfc7519>>.
- KHAZAEI, H. et al. Efficiency analysis of provisioning microservices. In: *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Luxembourg, Austria: IEEE, 2016. v. 1, n. 16622230, p. 261–268. ISSN 2330-2186.
- KIM, J. Y.; KIM, J. R.; PARK, C. J. Methodology for verifying the load limit point and bottle-neck of a game server using the large scale virtual clients. In: *2008 10th International Conference on Advanced Communication Technology*. Phoenix Park, Korea: IEEE, 2008. v. 1, p. 382–386. ISSN 1738-9445.
- KLEINA, N. *8 dos maiores mundos virtuais que já conhecemos*. 2018. [Online; Acessado em: 17. Apr. 2018]. Disponível em: <<https://www.tecmundo.com.br/internet/129103-habbo-second-life-8-maiores-mundos-virtuais-conhecemos.htm>>.
- LENGYEL, E. *Mathematics for 3D Game Programming and Computer Graphics, Third Edition*. Cengage Learning PTR, 2011. ISBN 978-143545886-4. Disponível em: <<https://www.amazon.com/Mathematics-Programming-Computer-Graphics-Third/dp/1435458869>>.
- LINIETSKY, A. M. J. *Godot Docs 3.0*. 2018. [Online; Acessado em: 15. Maio 2018]. Disponível em: <<http://docs.godotengine.org/en/3.0>>.
- MDHR. *Cuphead*. 2017. [Online; Acessado em: 8. Maio 2018]. Disponível em: <<https://store.steampowered.com/app/268910/Cuphead>>.
- MICROSOFT. *Age of Empires III - Age of Empires*. 2005. [Online; Acessado em: 8. Maio 2018]. Disponível em: <<https://www.ageofempires.com/games/aoeiii>>.
- MOJANG. *How do I play multiplayer?* 2009. [Online; Acessado em: 8. Maio 2018]. Disponível em: <<https://help.mojang.com/customer/en/portal/articles/429052-how-do-i-play-multiplayer->>.
- NADAREISHVILI, I. et al. *Microservice Architecture: Aligning Principles, Practices, and Culture*. O'Reilly Media, 2016. ISBN 978-149195625-0. Disponível em: <<https://www.amazon.com/Microservice-Architecture-Aligning-Principles-Practices/dp/1491956259>>.
- NEWMAN, S. *Building Microservices*. O'Reilly Media, 2015. ISBN 978-149195035-7. Disponível em: <<https://www.amazon.com.br/Building-Microservices-Sam-Newman/dp/1491950358>>.
- RAYMOND, E. S. *The Art of UNIX Programming (The Addison-Wesley Professional Computing Series)*. Addison-Wesley, 2003. ISBN 978-013142901-7. Disponível em: <<https://www.amazon.com/UNIX-Programming-Addison-Wesley-Professional-Computng/dp/0131429019>>.
- RINGLER, R. *C# Multithreaded and Parallel Programming*. Packt Publishing - ebooks Account, 2014. ISBN 978-184968832-1. Disponível em: <<https://www.amazon.com/Multithreaded-Parallel-Programming-Rodney-Ringler/dp/184968832X>>.

- RIOT. *Guia do Novo Jogador*. 2009. [Online; Acessado em: 8. Maio 2018]. Disponível em: <<https://br.leagueoflegends.com/pt/game-info/get-started/new-player-guide>>.
- ROLLINGS, A.; ADAMS, E. *Andrew Rollings and Ernest Adams on Game Design*. New Riders, 2003. (NRG Series). ISBN 9781592730018. Disponível em: <<https://books.google.com.br/books?id=Qc19ChiOUI4C>>.
- RUDDY, M. *Inside Tibia, The Technical Infrastructure of an MMORPG*. 2011. Disponível em: <<http://twvideo01.ubm-us.net/o1/vault/gdceurope2011/slides-Matthias\_Rudy\_ProgrammingTrack\_InsideTibiaArchitecture.pdf>>.
- SALDANA, J. et al. Traffic optimization for tcp-based massive multiplayer online games. In: *2012 International Symposium on Performance Evaluation of Computer Telecommunication Systems (SPECTS)*. Genoa, Italy: IEEE, 2012. p. 1–8.
- SALZ, D. *Albion Online - A Cross-Platform MMO (Unite Europe 2016, Amsterdam)*. 2016. Disponível em: <<https://www.slideshare.net/davidsalz54/albion-online-a-crossplatform-mmo-unite-europe-2016-amsterdam>>.
- SALZ, D. *Albion Online - Software Architecture of an MMO*. talk at Quo Vadis, 2016. [Online; Acessado em: 29. Agosto 2018]. Disponível em: <[https://pt.slideshare.net/davidsalz54/albion-online-software-architecture-of-an-mmo-talk-at-quo-vadis-2016-berlin?from\\_action=save](https://pt.slideshare.net/davidsalz54/albion-online-software-architecture-of-an-mmo-talk-at-quo-vadis-2016-berlin?from_action=save)>.
- SCS. *Euro Truck Simulator 2*. 2016. [Online; Acessado em: 8. Maio 2018]. Disponível em: <<https://eurotrucksimulator2.com>>.
- SPORTV. *League of Legends ganha torneio de fim de ano organizado pela ABCDE*. 2018. [Online; Acessado em: 17. Apr. 2018]. Disponível em: <<https://sportv.globo.com/site-e-sportv/noticia/league-of-legends-ganha-torneio-de-fim-de-ano-organizado-pela-abcdedo.html>>.
- STATISTA. *Games market revenue worldwide in 2015, 2016 and 2018, by segment and screen (in billion U.S. dollars)*. 2017. Acessado em: 19/01/2018. Disponível em: <<https://www.statista.com/statistics/278181/video-games-revenue-worldwide-from-2012-to-2015-by-source/>>.
- STATISTA. *Global internet gaming traffic 2021 | Statistic*. 2018. [Online; Acessado em: 19. Apr. 2018]. Disponível em: <<https://www.statista.com/statistics/267190/traffic-forecast-for-internet-gaming>>.
- STATISTA. *Global PC/MMO revenue 2015*. 2018. [Online; Acessado em: 1. Maio 2018]. Disponível em: <<https://www.statista.com/statistics/412555/global-pc-mmo-revenues>>.
- STATISTA. *LoL player share by region 2017*. 2018. Online; Acessado em: 17. Apr. 2018. Disponível em: <<https://www.statista.com/statistics/711469/league-of-legends-lol-player-distribution-by-region>>.
- SUZNJEVIC, M.; MATIJASEVIC, M. Towards reinterpretation of interaction complexity for load prediction in cloud-based mmorpgs. In: *2012 IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE 2012) Proceedings*. Munich, Germany: IEEE, 2012. v. 1, n. 13171916, p. 148–149. ISSN 978-1-4673-1567-8.

- THOMPSON, G. W. L. *Fundamentals of Network Game Development*. Cengage Learning, 2008. ISBN 978-158450557-0. Disponível em: <<https://www.amazon.com/Fundamentals-Network-Game-Development-Lecky-Thompson/dp/1584505575>>.
- VILLAMIZAR, M. et al. Infrastructure cost comparison of running web applications in the cloud using aws lambda and monolithic and microservice architectures. In: *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. Cartagena, Colombia: IEEE, 2016. p. 179–182. ISSN 1863-2386.
- XEROX. *High-level framework for network-based resource sharing*. 1976. [Online; Acessado em: 19. Maio 2018]. Disponível em: <<https://tools.ietf.org/html/rfc707>>.
- YARUSSO, A. *2600 Consoles and Clones*. 2006. Disponível em: <<http://www.atariage.com/2600/archives/consoles.html>>.
- ZELESKO, M. J.; CHERITON, D. R. Specializing object-oriented rpc for functionality and performance. In: *Proceedings of 16th International Conference on Distributed Computing Systems*. Hong Kong, China: IEEE, 1996. v. 1, n. 5323443, p. 175–187. ISSN 0-8186-7399-0.