Marlon Henry Schweigert

Análise de arquiteturas de microsserviços empregados a jogos MMORPG voltada a otimização do uso de recursos de gerenciamento de mundos virtuais

Joinville

2018

UNIVERSIDADE DO ESTADO DE SANTA CATARINA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Marlon Henry Schweigert

ANÁLISE DE ARQUITETURAS DE MICROSSERVIÇOS EMPREGADOS A JOGOS MMORPG VOLTADA A OTIMIZAÇÃO DO USO DE RECURSOS DE GERENCIAMENTO DE MUNDOS VIRTUAIS

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 DE GERENCIAMENTO DE MUNDOS VIRTUAIS

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 Ciência da Computação Integral do CCT/UDESC.

Banca Examinadora

Charles Christian Miers - Doutor (orientador)

Débora Cabral Nazário - Doutora

Guilherme Piêgas Koslovski - Doutor

Agradecimentos

AGRADECIMENTOS

Resumo

A crescente popularização de jogos massivos 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 algo pertinente e impactante para o sucesso desses jogos. O objetivo deste trabalho é propor uma análise voltada a identificar abordagens para otimização dos recursos computacionais consumidos pelas arquiteturas identificadas. Esse objetivo será atingido após realizar uma pesquisa referenciada, seguida de uma análise das principais arquiteturas e, preferencialmente, a execução de simulações usando uma nuvem computacional para auxiliar na identificação de gargalos de recursos. Os resultados obtidos auxiliarão provedores de serviços *Massively Multiplayer Online Role-Playing Game* (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, Nuvens computacionais

Abstract

The increasing popularization of mass games demands new technological approaches in order to meet the needs 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 identified architectures. This objective will be achieved after conducting a referenced search, followed by an analysis of the main architectures and, preferably, the execution of simulations using a computational cloud to assist in the identification of resource bottlenecks. The results obtained will help service providers to reduce maintenance costs and improve the quality of such services.

Keywords: Cloud computing, Traffic characterization, Management network, Traffic monitoring system, Performance analysis, OpenStack.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

API *Application Programming Interface*

AWS *Amazon Web Services*

CRUD *Create Read Update Delete*

CPU *Central Processing Unit*

C/S Cliente/Servidor

FPS *First-person shooter*

HTTP *Hypertext Transfer Protocol*

IDE *Integrated Development Environment*

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*

NPCs *Non-Playable Characters*

P2P *Peer-to-Peer*

PvP *Player vs Player*

PvNPCs *Player vs Non-Playable Characters (NPCs)*

PaaS *Platform as a Service*

POF *Point of View*

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*

VM *Virtual Machine*

WAN *Wide Area Network*

XDR *External Data Representation*

XML *Extensible Markup Language*

1 Introdução

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 para essa área de estudo (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. Reduzir a carga e impacto da latência em serviços para jogos massivos resulta em uma melhor experiência de jogabilidade aos usuários finais (HUANG; YE; CHENG, 2004).

Jogos MMORPG são utilizados como negócio viável e lucrativo, sendo que experiência de jogabilidade na qual o usuário final será submitido é um fator crítico para o sucesso. Tais jogos são de interpretação de papéis massivos. 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 duas partes (KIM; KIM; PARK, 2008):

- **Serviço**: É o macrosserviço que implementa as regras de negócio e requisitos do jogo. O serviço disponibiliza uma interface com ações possíveis ao cliente sobre algum protocolo de rede.
- **Cliente**: Cliente é a aplicação que realizará as requisições com a interface do macrosserviço, exibindo o estado de jogo de forma imersiva ao jogador.

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). Modelar um sistema de alto desempenho torna-se um trabalho essencial para

a satisfação do usuário final (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.

1.0.1 Ocorrências de serviços massivos

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 este 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 os microsserviços**: 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.

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).

1.0.2 Arquiteturas de microsserviços

Entende-se por microsserviço as aplicações que executam operações menores de um macrosserviço, da melhor forma possível (CLARKE-WILLSON, 2017). Microsserviços devem funcionar separadamente e de forma autônoma. 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).

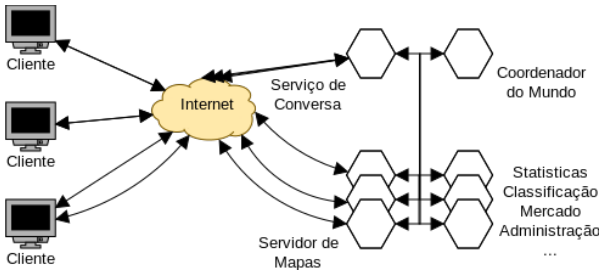
Arquiteturas de microsserviços iniciam uma nova linha de desenvolvimento de aplicações preparadas para executar sobre nuvens computacionais, promovendo maior flexibilidade, escalabilidade, gerenciamento e desempenho, sendo a principal escolha de arquitetura de grandes empresas como Amazon, Netflix e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

- Deve posibilitar 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.
- A interface deve estar disponível na rede para chamada de processamento remoto.
- 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.

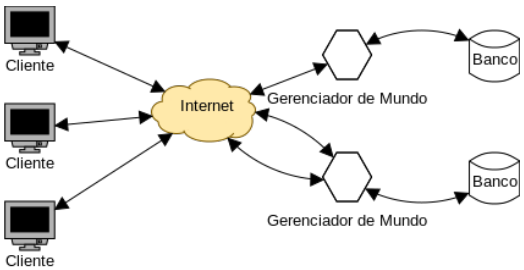
Alguns exemplos de arquitetura de microsserviços para jogos MMORPG são as arquiteturas apresentadas por Rudy (Figura 1.2), Salz (Figura 1.1) e a arquitetura escrita por Willson (Figura 1.3).

Figura 1.1: Arquitetura Salz, utilizada no jogo Albion.



Adaptado de: (SALZ, 2016a)

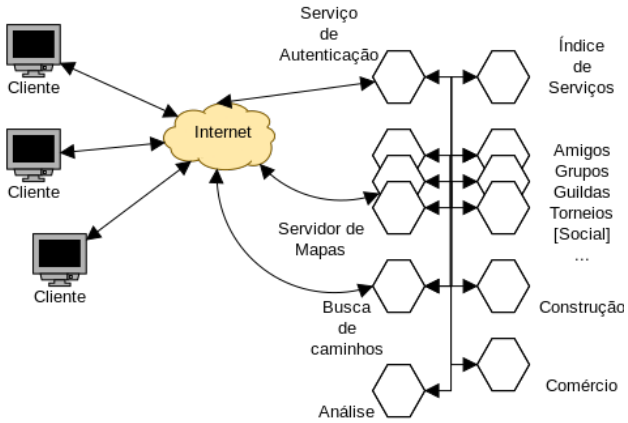
Figura 1.2: Arquitetura Rudy, utilizada no jogo Tibia.



Adaptado de: (RUDDY, 2011)

A arquitetura Rudy (Figura 1.2) é formada por um sistema cliente-servidor monolítico, na qual cada microsserviço individual gerencia um mundo mútuo dos demais gerenciadores de mundo (RUDDY, 2011). Essa arquitetura dificulta a escalabilidade, modificações e manutenção (ACEVEDO; JORGE; PATIñO, 2017), além de segregar a comunidade de jogadores em servidores menores (RUDDY, 2011). Inicialmente essa arquitetura

Figura 1.3: Arquitetura Willson, utilizada no jogo Guild Wars 2.



Adaptado de: (CLARKE-WILLSON, 2017)

foi pensada para ser um sistema Cliente-Servidor monolítico. A arquitetura Rudy é uma arquitetura de microsserviços adaptada de um serviço cliente-servidor (RUDDY, 2011). O jogo Tibia[1], operante sobre essa arquitetura, possui 68 mundos oficiais (Sendo 2 servidores de teste) (RUDDY, 2011), com capacidade para 1.050 clientes em cada servidor, na qual encontra-se restringido pelo gerenciador de mundo.

A arquitetura Salz (Figura 1.1) é formada por diversos microsserviços (SALZ, 2016a). O principal objetivo dessa arquitetura é modularizar o serviço visando melhorar a escalabilidade. Ela é atualmente utilizada no jogo Albion Online[2]. A arquitetura é planejada para funcionar conforme a seguinte especificação(SALZ, 2016a):

- O mundo é distribuído sobre os vários servidores de mapas. Cada microsserviço gerencia uma região do mundo, denominado *chunk*.
- Jogadores mudam a conexão com os microsserviços quando estão posicionados na borda de um *chunk*.
- A autorização de acesso aos microsserviços é obtido pelo banco de dados.
- O coordenador do mundo é responsável por tudo que seja de escopo global (*e. g.,* Grupos, chat global, guildas, *etc.*).

A arquitetura Willson (Figura 1.3) é distribuída em diversos microsserviços, assim como a arquitetura Salz (Figura 1.1). A diferença em comparação a arquitetura

[1]Tibia: `http://www.tibia.com`
[2]Albion Online: `https://albiononline.com`

Salz (Figura 1.1) está na decomposição da arquitetura para outros microsserviços e a conexão direta entre esses microsserviços e o cliente. O principal objetivo dessa arquitetura é facilitar a manutenção e desempenho de reinicialização do macrosserviço (CLARKE-WILLSON, 2017). Guild Wars 2[3] é um jogo que executa sobre a arquitetura Willson. Ele é popularmente conhecido por ter seus serviços sempre ativos, visto que a arquitetura possibilita desativar pequenos pedaços do serviço para manutenções básicas e a sua reinicialização é rápida para manutenções críticas.

Conforme as arquiteturas de microsserviços dos jogos MMORPG aumentam para atender novas demandas do mercado, o seu custo computacional e complexidade de nós na nuvem computacional aumentam. Por esse motivo a análise e otimização dessas arquiteturas é importante para impactar em um melhor desempenho tanto ao usuário final, quanto a otimização de lucros das empresas que disponibilizam esses serviços.

[3]Guild Wars 2: `https://www.guildwars2.com`

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 será tomado como introdução para o conceito de gênero de jogo, abordado na Subseção 2.1.1, 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 2.7) 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 2.7.4).

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).

O crescente recurso computacional disponível 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 hardwares, iniciou-se o uso da rede de computadores para proporcionar a interação entre jogadores em equipamentos distintas (STATISTA, 2018a). Jogos como EA Habitat[1], CipSoft Tibia[2] e Jajex Runescape[3] 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 de jogabilidade e desafio proposto ao jogar com milhares de jogadores (GUINNESS,

[1]EA Habitat: `http://www.mobygames.com/game/c64/habitat/credits`
[2]CipSoft Tibia: `http://www.tibia.com/`
[3]Jajex Runescape: `https://www.runescape.com`

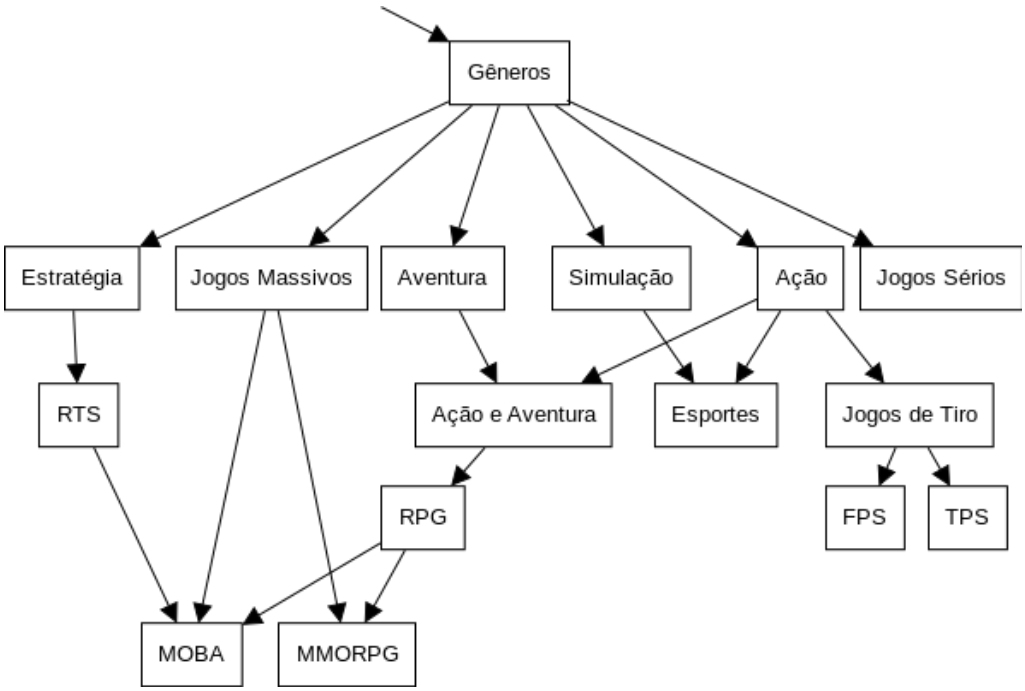
2013; HUANG; YE; CHENG, 2004), criando o gênero de jogos *Massively Multiplayer Online* (MMO).

Como o gênero MMO, muitos outros foram criados devido ao aumento da quantidade de desenvolvedores e as novas tecnologias. Nesse sentido, as redes de computadores serviram como impulsionador 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.

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 comprem jogos com características conforme o seu interesse (CLARKE; LEE; CLARK, 2015). A árvore pode ser visualizada pelo diagrama 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, os gêneros não definem de forma explícita o conteúdo expresso em algum jogo eletrônico, mas sim um desafio comum presente no jogo anali-

sado (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[4] 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[5] e Riot League of Legends[6]. O jogo League of Legends obteve 100 milhões de usuários ativos em 2016 (STATISTA, 2018c), além de ter um torneio nacional e mundial (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 se faz permitido interações em um mundo na qual

[4]SecondLife: `https://www.secondlife.com/`
[5]Blizzard Dota 2: `http://br.dota2.com/`
[6]Riot League of Legends: `https://br.leagueoflegends.com/pt/`

outros jogadores também estão jogando, 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 Word of WarCraft[7]. A grande parte dos jogos MMORPG utilizam tecnologia WAN.

- 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 Ação e 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[8]. 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[9] é um título popular nesse segmento. É comum encontrar tecnologias P2P e LAN.
- Ação: Preza pela habilidade de coordenação motora e reflexos do jogador, para tomar uma atitude a fim de passar seus objetivos no cenário. Nesse gênero os objetivos são passar por uma série de desafios que incluam movimentação e posicionamento de outros objetos no cenário (ADAMS, 2006). É comum encontrar tecnologias LAN, P2P, WAN e C/S.

[7]Word of WarCraft: `https://worldofwarcraft.com/pt-br/`
[8]Nintendo The Legend of Zelda: `https://www.zelda.com/`
[9]FIFA: `https://www.easports.com/br/fifa`

  - Jogos de Tiro: Usa um número finito de armas para executar ações a distância. O posicionamento, movimentação estratégia 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* (POF). 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 também 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. 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 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

Tabela 2.1: Tipos de comunicação e quantia de jogadores populares em gêneros

	LAN	WAN	P2P	C/S	Jogadores
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.

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 é de mais de 30 bilhões de dólares americanos com 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 duas 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.

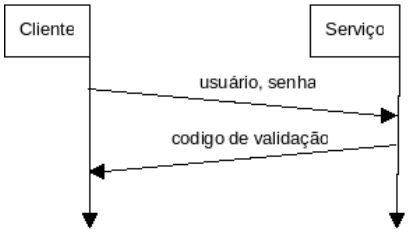
2.3 Jogabilidade de jogos MMORPG

É comum serviços MMORPG terem com funcionalidades parecidas. Dessa forma, facilita a compreenção básica de forma genérica de um jogo MMORPG, e auxilia a compreender o modelo computacional implementado nestes serviços. 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, a fim de disponibilizar um código para validar todas as futuras ações da seção do usuário. Ele pode ser visualizado de forma macro na Figura 2.2.

O sistema de autenticação visualizado na Figura 2.2 é o principal, porém não

Figura 2.2: Sistema de autenticação para jogos



Fonte: Adaptado de (THOMPSON, 2008)

o único. O jogo Realm of the Mad God[10] 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 auditorado em qualquer serviço pertencente ao jogo, visto que ele foi assinado pelo sistema de autenticação do serviço (IKEM, 2018).

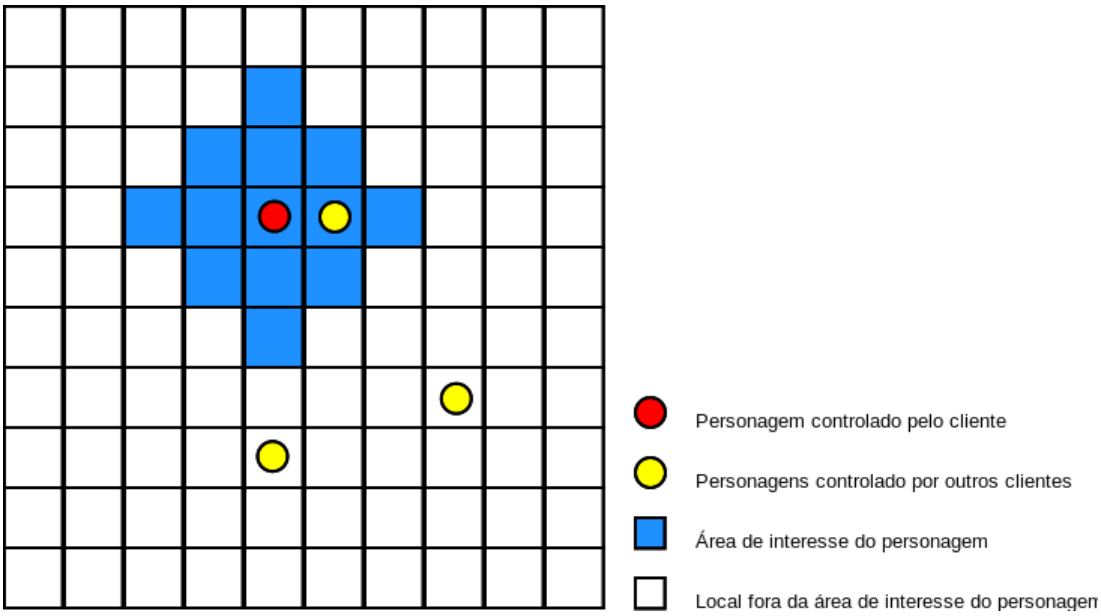
Após a autenticação, é comum existir um sistema para seleção de personagem, caso o jogo seja desenhado com este objetivo. Efetuada a seleção ou criação de um personagem, este será imerso no mundo compartilhado do jogo com os demais jogadores (RUDDY, 2011). Nos jogos MMORPG é comum a restrição da visão 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 e uma área de interesse de longa distância (SALZ, 2016a), sendo que o jogador não tem informações dos demais objetos e jogadores fora de sua área de interesse. Esta característica impede trapaças (visto que o cliente não tem informações que só estão contidas no serviço) 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):

[10]Realm of the Mad God: `https://www.realmofthemadgod.com/`

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



Fonte: (SALZ, 2016a)

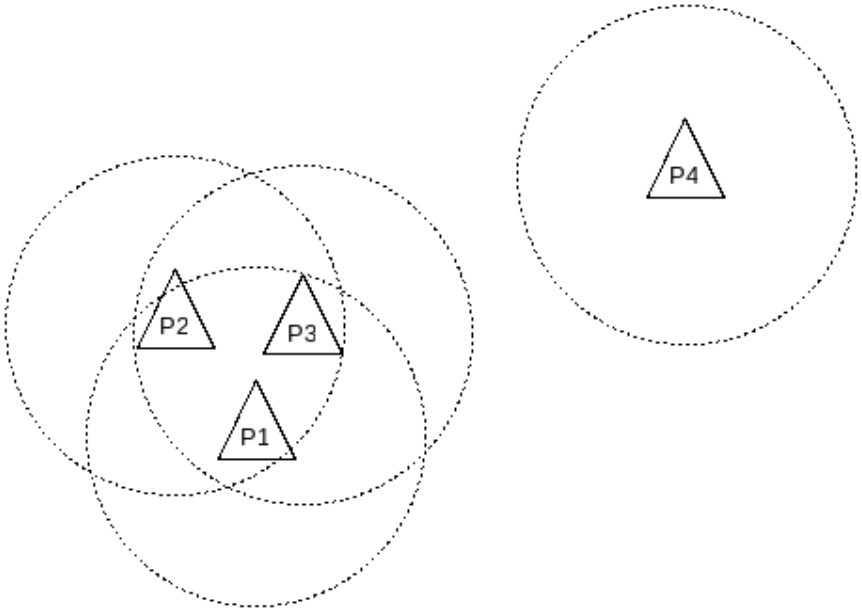
- Enviar e receber mensagem no chat;
- Mover-se pelo ambiente;
- Interagir com outros jogadores, NPCs ou objetos fixos do ambiente; e
- Obter itens do ambiente.

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 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 a 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.

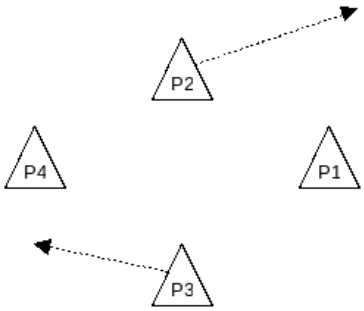
Figura 2.4: Chat baseado em contexto de posicionamento, utilizando Distância Euclidiana



Fonte: Adaptado de (SALZ, 2016a)

O sistema de movimento pelo ambiente do jogo possibilita que cada jogador movimente o 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 contante pela qualidade da jogabilidade (SALZ, 2016a).

Figura 2.5: Personagens e os seus pontos de destino

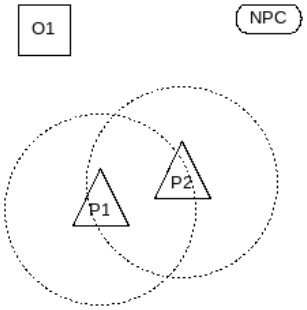


Fonte: O próprio autor.

A interação com o ambiente, itens, NPCs e outros jogadores 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 2.6 (SALZ, 2016a).

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.

Figura 2.6: Personagens, objetos e NPCs em no ambiente



Fonte: O próprio autor.

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 este 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[11], que pode ser visualizado na Figura 2.7.

A Figura 2.7 representa na prática, como será exibido ao jogador o ambiente

[11]Sandbox-Interactive Albion: `https://albiononline.com/en/home`

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

Fonte: (SALZ, 2016a)

do jogo ao cliente final. 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 formado 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*.

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

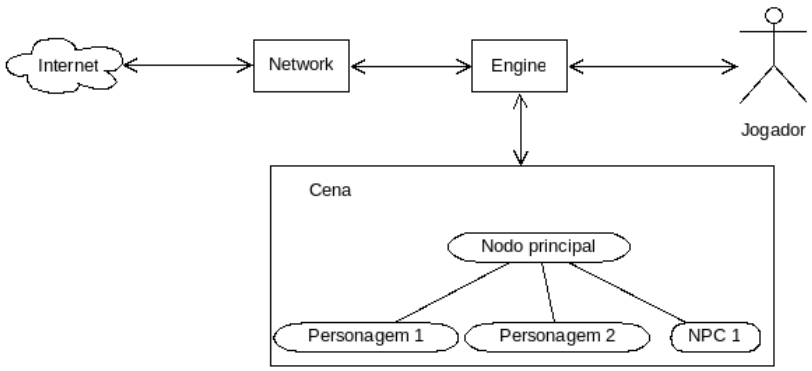
Fonte: O próprio autor.

Dentro dessa árvore, cada nodo tem uma funcionalidade específica. Essas funcionalidades são específicas a cada motor gráfico, podendo existir nodos específicos à parte física, renderização ou controles (LINIETSKY, 2018). Em um jogo MMORPG, o serviço 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 possui uma *Thread* dedicada ao seu funcionamento (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 Figura 2.9 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 também 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[12] e GodotEngine[13] 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):

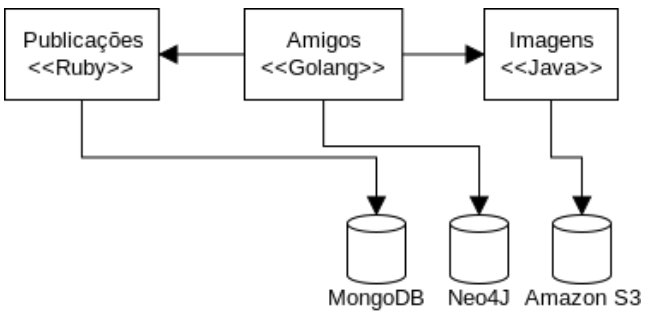
- Deve possibilitar a implementação como uma peça individual do macrosserviço.

[12]Unity3D: `https://www.unity3d.com`
[13]GodotEngine: `https://www.godotengine.org`

- Deve funcionar individualmente.
- Cada serviço deve ter uma interface. Essa interface deve ser o suficiente para utilizar o microsserviço.
- 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 [14], *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 um 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



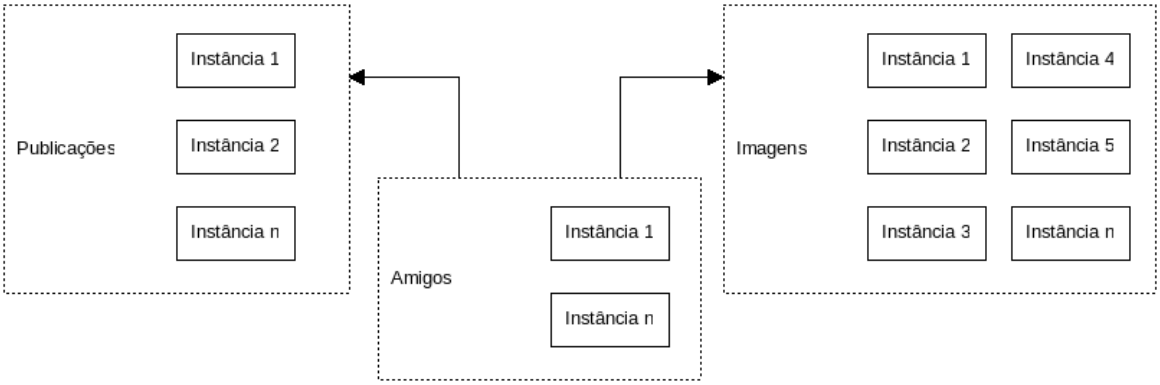
Adaptado de: (NEWMAN, 2015)

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

[14]Docker: `https://www.docker.com/`

a necessidade de escalabilidade. Este modelo computacional obtém maior desempenho, principalmente se executar sobre plataformas de computação elástica, na qual o orquestrador do macrosserviço pode aumentar o número de instâncias conforme a necessidade de requisições (NADAREISHVILI et al., 2016).

Figura 2.11: Microsserviços são escaláveis



Adaptado de: (NEWMAN, 2015)

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, comparado-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, e esta redução de contexto pode ser aplicada na visualização).

3. *Controller*: Representa as operações sobre modelos que serão requiridos 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 *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.

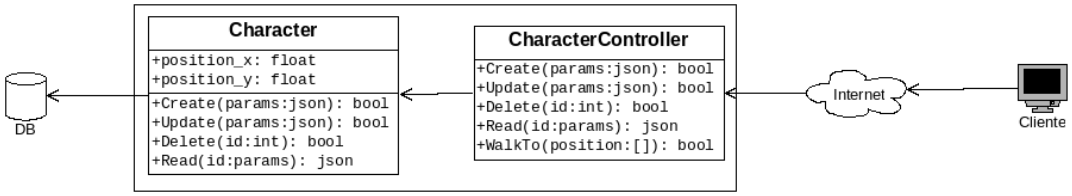
3. *Delete*: Representa a exclusão de um objeto no banco.

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.

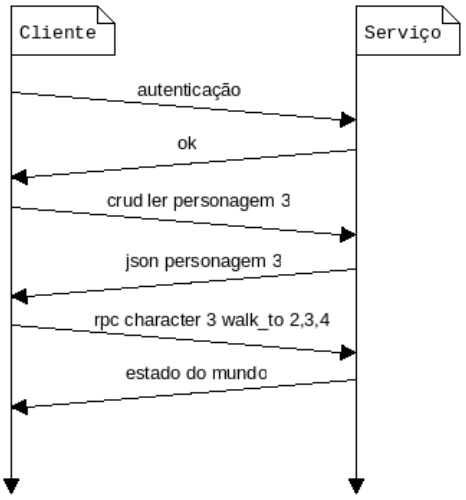
Essas operações são executadas utilizando protocolo RPC (SALZ, 2016a). As requisições de métodos remotos são realizadas entre dois processos distintos, a fim de gerar uma computação distribuída (XEROX, 1976). Entretanto, a base do protocolo não é legível como em servidores web que utilizam JSON para transmissão de estrutura de dados, mas sim uma codificação binária nomeado *External Data Representation* (XDR) (Internet Society, 2006).

Figura 2.12: Cliente pode realizar requisições CRUD ao serviço.



Fonte: Adaptado de (SALZ, 2016a).

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



Fonte: Adaptado de (THOMPSON, 2008)

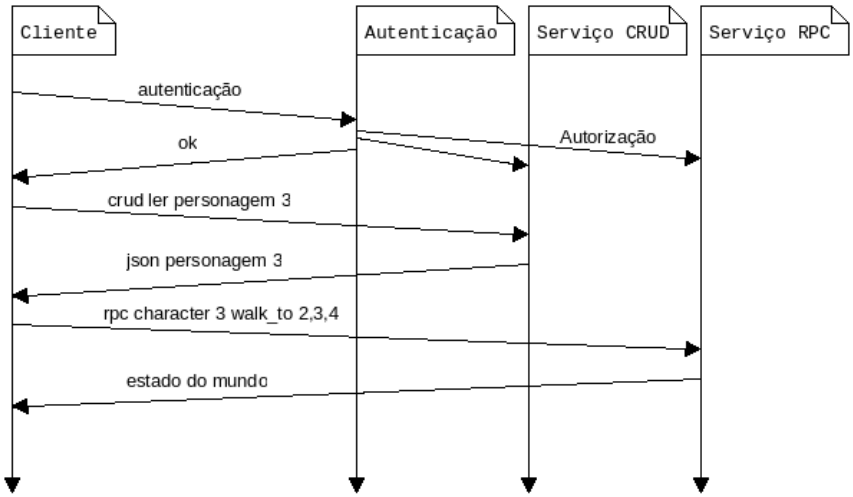
Uma técnica comum em jogos é a compressão de pacotes utilizando mapeamento *hash* de bytes (THOMPSON, 2008). Tanto o cliente quanto o serviço precisam ter a mesma estrutura de dados. Dessa forma, é possível trocar o nome das funções solicitadas em RPC por poucos bytes para transitar na rede. Já para 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).

Como relatado na Seção 2.4.2, uma arquitetura de microsserviços permite múltiplas tecnologias, pois a comunicação entre todos os elementos de um microsserviço 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.

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[15] (SALZ, 2016a). Percebe-se, neste contexto,

[15]Sandbox-Interactive Albion: `https://albiononline.com/en/home`

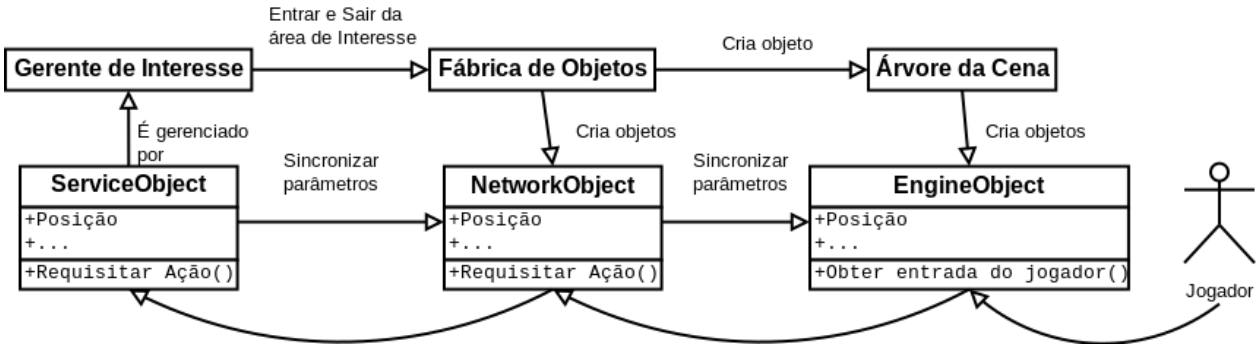
Figura 2.14: Diagrama de requisições entre serviço e cliente com operações CRUD e RPC em uma arquitetura de microsserviços.



Fonte: O próprio autor.

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.



Fonte:Adaptado de (SALZ, 2016a)

A Figura 2.15 demonstra 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 populares 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 Principais arquiteturas MMORPG

Dentre o modelo C/S, encontra-se modelos diferentes para processamento das ações dos jogadores. Dentre essas arquiteturas, as que se destacam com maior popularidade entre os jogos 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 gerencia 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 a funcionalidades do jogo. Dessa forma, o ambiente do jogo torna-se escalável ao número de jogadores, porém a latência tende a aumentar.

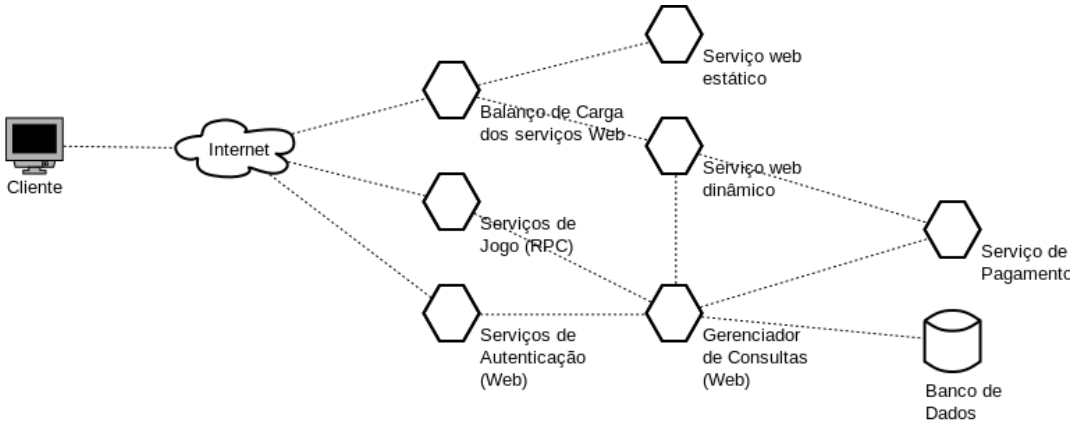
A arquitetura Willson (Subseção 2.5.3) utiliza um intermediário, evitando a divisão em multiplos serviços para o gerente de jogo e utilizando um modelo de paralelismo próximo a arquitetura Salz.

Entretanto, se faz necessário conhecer a topologia de microsserviços, características e abordagens para solução de cada serviço MMORPG, visando classificar os protocolos utilizados por cada microsserviço e o seu objetivo na arquitetura.

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):

1. **Serviço web estático**: Armazena documentos estáticos para o serviço web (*e.g.,* imagens, executáveis do jogo, páginas web fixas, etc). Responde sobre o protocolo HTTP.

2. **Serviço web dinâmico**: Sistema web para cadastro de contas, guias, informações sobre atualizações, compras e demais demanda de páginas dinâmicas. Responde sobre o protocolo HTTP.

3. **Balanço de carga web**: Realiza a distribuição de carga sobre o *Serviço web estático* e o *Serviço web dinâmico*. Responde sobre o protocolo HTTP.

4. **Serviço de Jogo**: Gerencia um mundo inteiro, em um único serviço. Esta abordagem segrega 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.

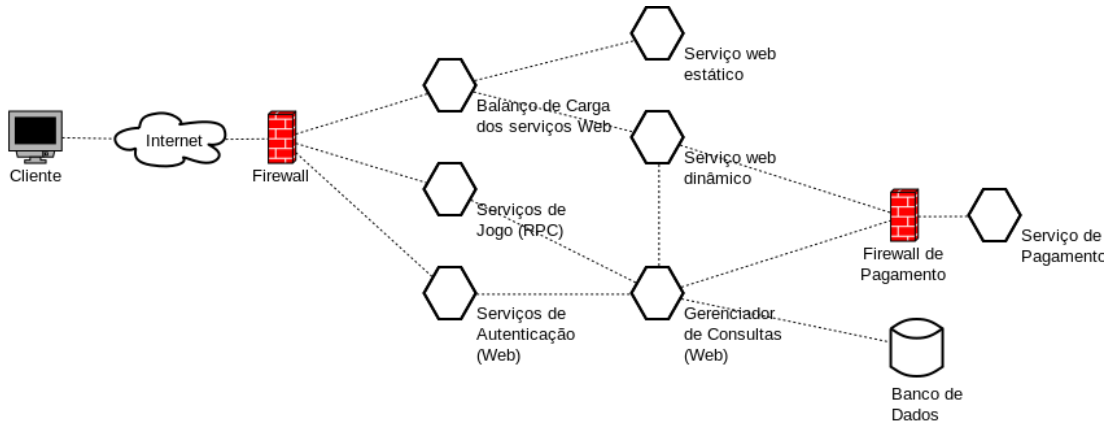
5. **Serviço de Autenticação**: Gerencia a autenticação das conexões ao *Serviço de jogo*. Este serviço opera sobre o protocolo RPC.

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[16], MySQL[17], 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.



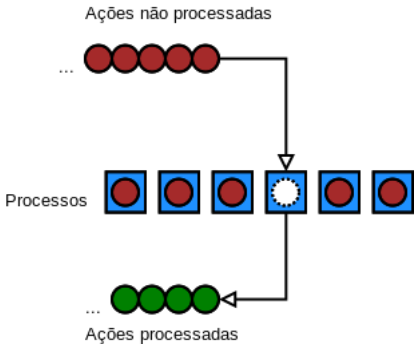
Adaptado de: (RUDDY, 2011).

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 ser consumido pelo ciclo de processamento do gerente de 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.

O modelo de processamento do gerente de mundo, visualizado na Figura 2.18 trabalha com um a 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.

[16]PostgreSQL: `https://www.postgresql.org`
[17]MySQL: `https://www.mysql.com/`

Figura 2.18: Modelo de processos *Thread Pool*



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

Um serviço que chama 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 é (RUDDY, 2011):

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.

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

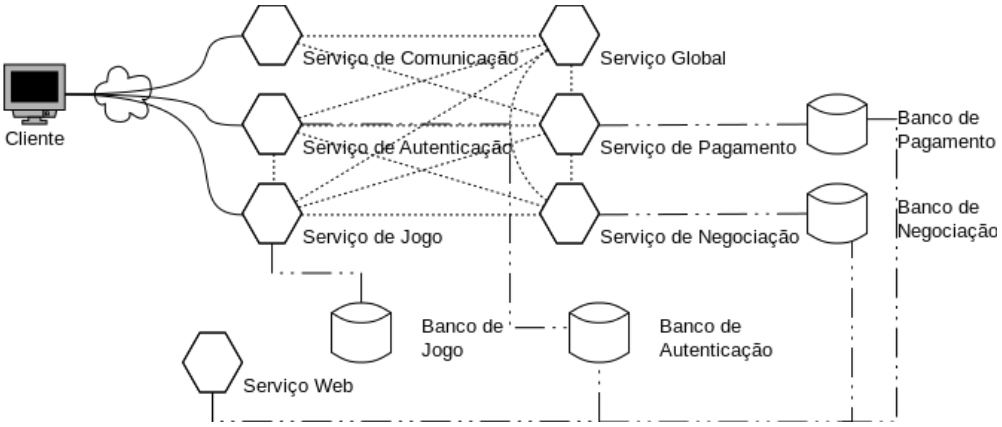
1. Aumenta a complexidade de implementação, teste, administração e ponto de falha.
2. Adiciona limites como número de conexões, número de requisições, etc.

Entretanto, esta arquitetura não permite escalar um único ambiente para um número de jogadores simultâneos maior a qual o hardware que hospeda o serviço é designado. Para 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).

Figura 2.19: Arquitetura Salz completa.



Adaptado de: (SALZ, 2016a).

A Figura 2.19 exibe a arquitetura Salz de forma completa. Na Figura 2.19 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), sendo os demais bancos com 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**: Recepciona 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.

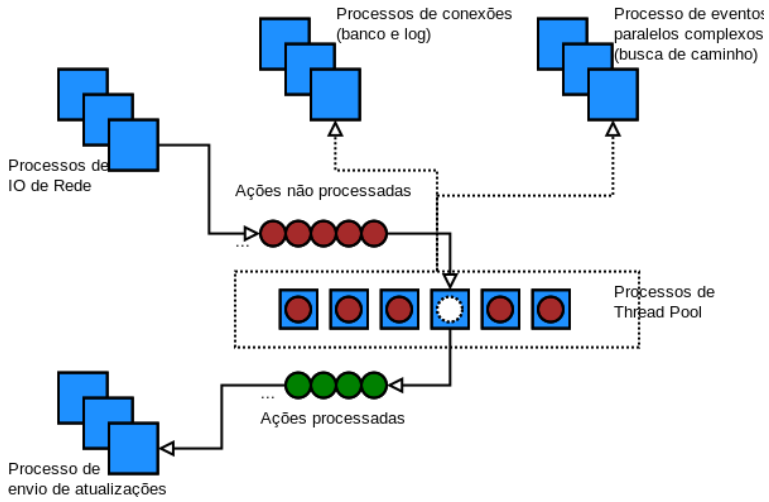
4. **Serviço Global**: Gerencia operações globais (*e.g,* interações entre grupos, procedimentos recorrentes globais, *etc.*). Opera sobre HTTP.

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 utiliza em grande parte serviços *web*, utilizando somente o protocolo RPC para o serviço de jogo. No serviço de jogo, tanto o cliente quando 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 possúi 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 solução de requesições, evitando a repetição da mesma conexão, na arquitetura Salz (Figura 2.19) as chamadas de processo remoto são executadas por ordem de chegada (SALZ, 2016b). Desta forma, a melhor conexão com o serviço terá maior vantagem, porém não existe custo de recursos para este gerenciamento no serviço.

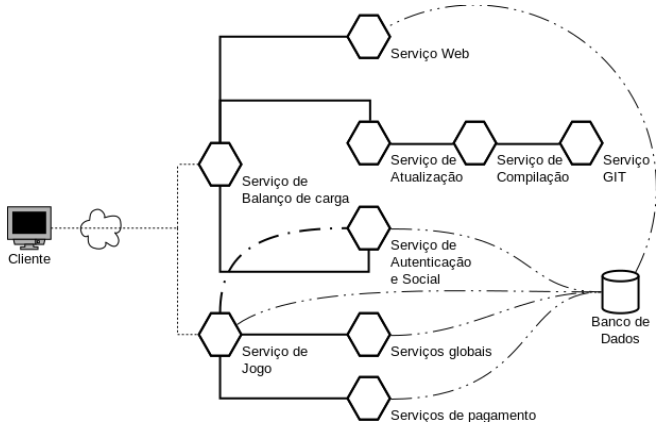
Na arquitetura salz, são necessários 3 conexões com o servidor, de forma contínua (SALZ, 2016a). O custo desta operação é alto, e por isso nem sempre é viável utilizar esta abordagem, principalmente para jogos onde a demanda de banda seja menor. Para evitar esta decisão, também é notório a abordagem 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 também 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.

A arquitura 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 arquitetura híbrida, onde divide funcionalidades em outros microsserviços, porém ainda mantem 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-

Figura 2.21: Arquitetura Willson



Fonte: (CLARKE-WILLSON, 2017).

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 e Social**: 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 Salz 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 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* 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

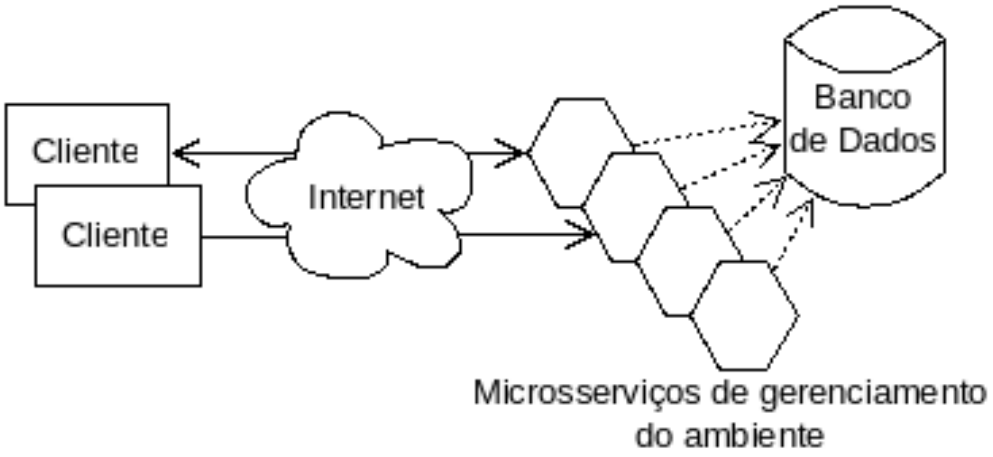
A escolha de uma arquitetura para serviços MMORPG que não suportem uma elevada carga de trabalho podem ser um eventual problema na escalabilidade do negócio de empresas que forneçam tais serviços. Em alguns serviços, a divisão da carga é dada pela área de interesse, dividindo o ambiente em pedaços a fim de diminuir a carga no serviço. Entretanto, 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. Entretanto, 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 comandos. 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, comércio e estatísticas, aumentando o grau de complexibilidade 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 descrever o seu comportamento e desempenho.

A literatura aborda a 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 2.7. Torna-se de interesse para empresas 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. Dentro do cenário de testes espera-se analisar por exemplo a estabilidade, limite de conexões e custo de processamento de

tais arquiteturas. Além disso, espera-se simular a carga de uma arquitetura utilizada em produção, a fim de possibilitar a quantificação do custo de operação de tal serviço. Estas são expectativas gerais do que espera-se analisar, mas conforme a análise dos dados obtidos for realizada, é possível que outras características surjam. Para guiar a proposta, um dos passos importantes são os trabalhos relacionados com este tema, na qual analisa-se estudos que abordam arquiteturas de jogos MMORPG e/ou arquiteturas de microsserviços.

2.7 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. 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.

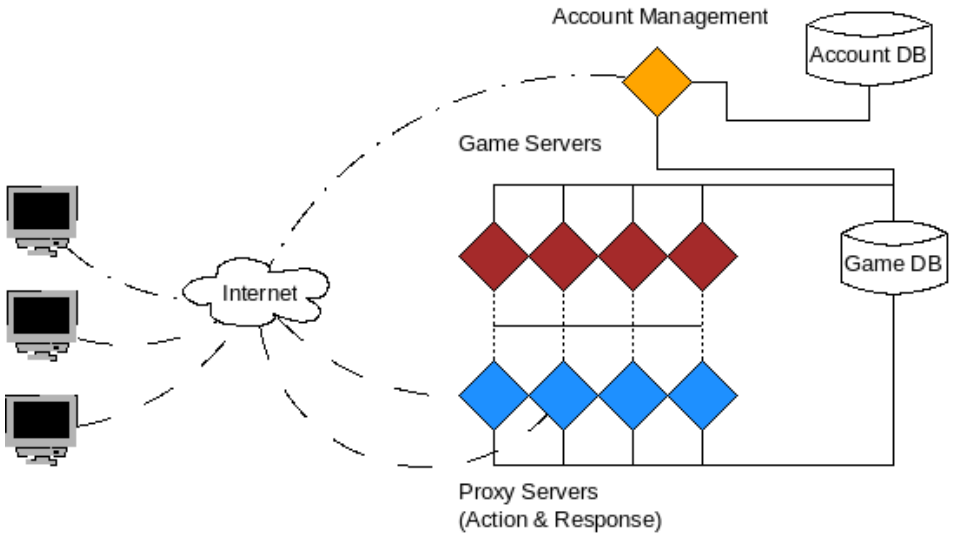
Para encontrar os trabalhos relacionados, foi efetuada uma busca pelos termos MMORPG ou *microsservices*. Entretanto, os dois termos dificilmente se correlacionam nos meios de busca disponíveis para a elaboração deste TCC. Nesse sentido, os trabalhos relacionados abordam arquiteturas de jogos MMORPG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

2.7.1 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.

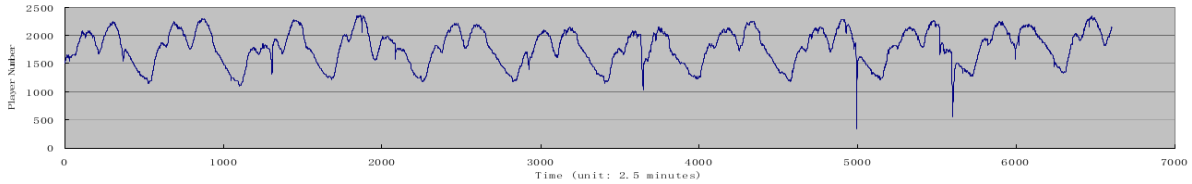
Figura 2.23: Arquitetura distribuída utilizando proxy.



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

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 utilizado entre o Cliente e *Proxy Server* é baseado em RPC (FARBER, 2002; BORELLA, 2000), porém não é relatado sobre o o protocolo de comunicação utilizado entre o *Proxy Server* e o *Game Server*. A sua arquitetura pode ser observada na Figura 2.23, na qual obteve seus dados obtidos durante 100 dias para realizar as análises. A Figura 2.24 demonstra uma amostra do número de conexões pelo tempo no serviço obtido.

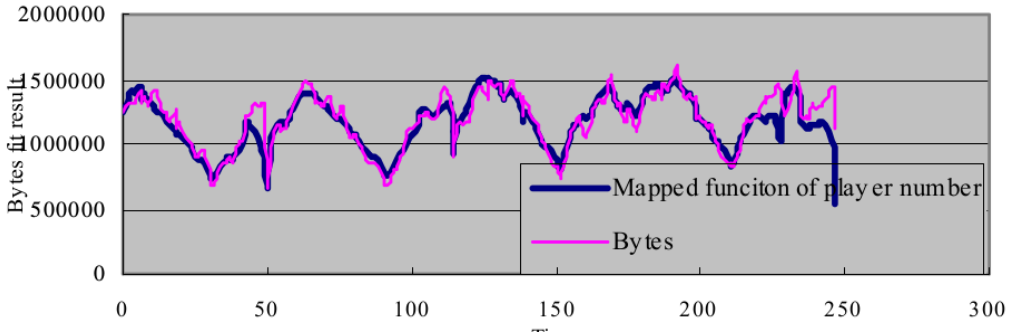
Figura 2.24: Número de conexões no serviço pelo tempo decorrido.



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 2.25, no qual o autor compara o consumo de banda real comparado a regressão linear.

Figura 2.25: Regressão linear comparado ao consumo de banda real do servidor.



Fonte: (HUANG; YE; CHENG, 2004)

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

2.7.2 Villamizar et al. (2016)

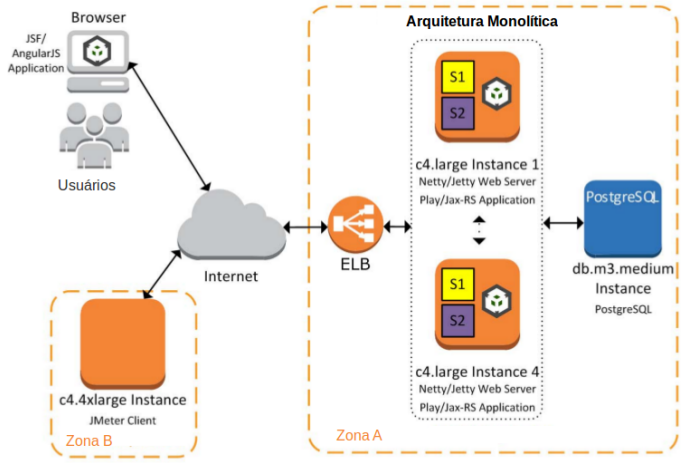
O trabalho de (VILLAMIZAR et al., 2016) investiga o custo de arquiteturas de microsserviços, arquiteturas *Platform as a Service* (PaaS) orientadas a eventos e aplicações monolíticas para aplicações web. A sua principal motivação é a comparação de custos para a tradução de sistemas legados para arquiteturas 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 2.26 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 2.27 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 [18].

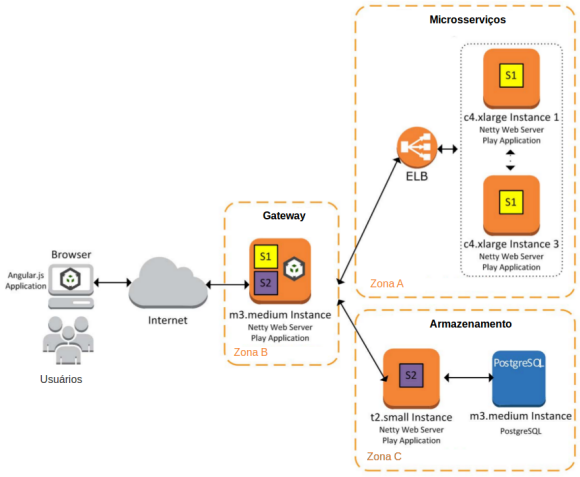
[18]Play Framework: `https://www.playframework.com/`

Figura 2.26: Arquitetura monolítica web implementada na AWS



Fonte: (VILLAMIZAR et al., 2016)

Figura 2.27: Arquitetura de microsserviços web implementada na AWS



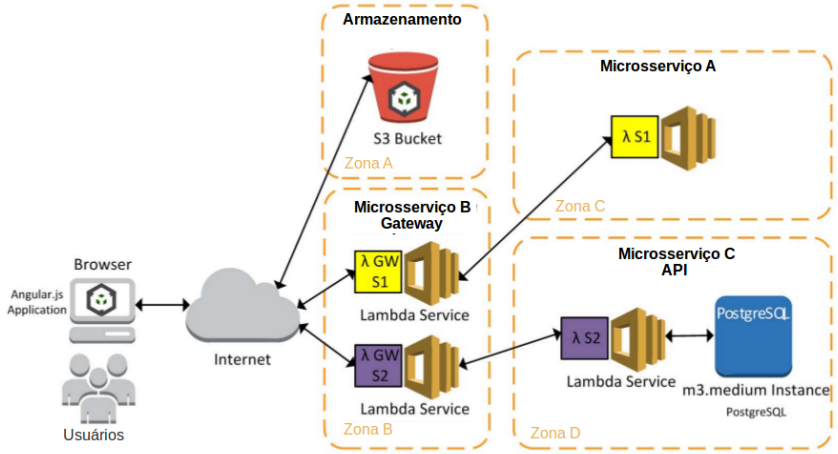
Fonte: (VILLAMIZAR et al., 2016)

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 2.28 exibe a implantação de uma aplicação de microsserviços web utilizando a tecnologia AWS *lambda*. Essa arquitetura foi implementada utilizando o *framework Node.js*[19], 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 2.29. O autor alerta sobre tolerância a falhas, transações distribuídas, distribuição de dados e versionamento de serviço.

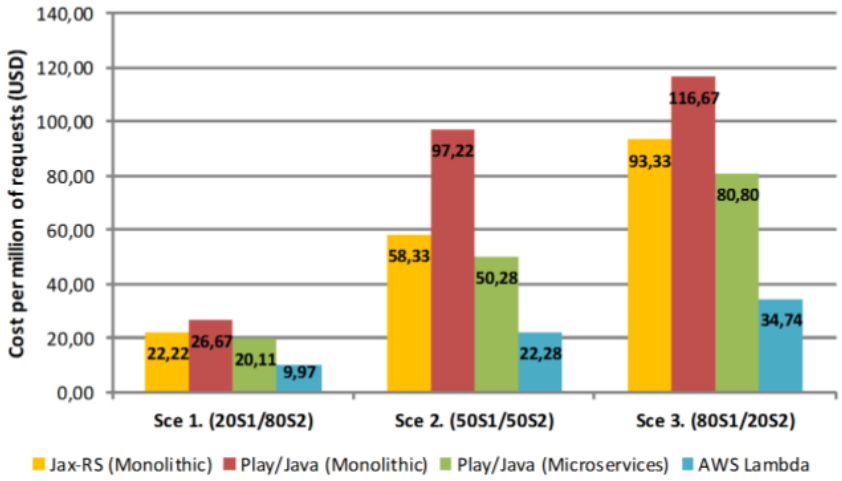
[19]Node.js: `https://nodejs.org/en/`

Figura 2.28: Arquitetura de microsserviços web implementada na AWS utilizando a tecnologia *lambda*



Fonte: (VILLAMIZAR et al., 2016)

Figura 2.29: Custo por um milhão de requisições em dólares utilizando diferentes arquiteturas sobre a AWS



Fonte: (VILLAMIZAR et al., 2016)

2.7.3 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 2.7.1, 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 2.2. Essa tabela exibe as ações que podem ser executadas para

Tabela 2.2: Complexidade da interação com o ambiente, por contexto da interação

Contexto da ação	PvP	PvNPCs	Número de NPCs	Network [kbits/s]
Questing	$O(n)$	$O(nlog(n))$	$N \leq 6$	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	$O(n^2log(n))$	$O(n^3)$	$N \leq 40$	32.0

Fonte: (SUZNJEVIC; MATIJASEVIC, 2012)

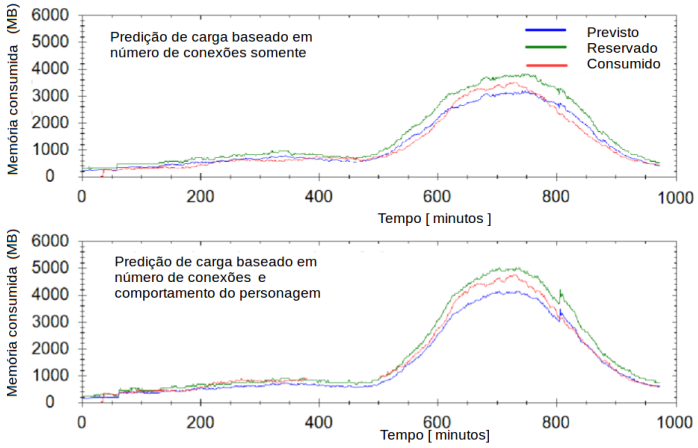
interagir com o ambiente, seja essa interação com PvP (jogador com outro jogador) ou PvNPCs (jogador com um personagem ou objeto gerenciado pelo serviço). Os contextos analisados nessa tabela são:

- *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 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.

Utilizando 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 2.30.

O autor conclui que o contexto de interação com o ambiente de cada personagem tem relevância com o consumo de *Central Processing Unit* (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.

Figura 2.30: Regressão levando em conta a complexidade das ações e contexto dos personagens



Fonte: (SUZNJEVIC; MATIJASEVIC, 2012)

2.7.4 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 2.3.

Tabela 2.3: 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 2.4. Nenhum trabalho relatou limite de conexões ou latência do sistema.

Tabela 2.4: Trabalhos relacionados por recurso utilizado

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 sí e precisavam um dos outros para o seu funcionamento. A relação de arquiteturas abordadas pode ser visualizado na Tabela 2.5, 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 2.5: 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.

2.8 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, a qual serão utilizados no problema deste Trabalho de Conclusão de Curso.

Por fim, são apresentados trabalhos relacionados a fim de guiar por métricas, métodos de coleta de dados e métodos de desenvolvimento e implantação de tais arquiteturas a fim de contribuir com o atual escopo do trabalho. Entretanto, houve uma dificuldade em encontrar na literatura análises sobre arquiteturas de microsserviços focado a jogos, principalmente em específico o gênero MMORPG. Dessa forma, espera-se introduzir a analise da viabilidade e do custo de recursos computacionais para manter um serviço MMORPG sobre uma arquitetura de microsserviços.

3 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 sí, 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 na escolha de implementações arquiteturas de microsserviços, em específico para serviços MMORPG.

Neste capítulo é descrito a proposta para análise de consumo de recursos computacionais em arquiteturas MMORPG. Inicialmente, é descrita a proposta em alto nível (Seção 3.1), trazendo os objetivos desta análise, quais recursos e métricas serão analisadas (TCC-II). Os Critérios de Análise (Seção 3.2) exibem como os dados obtidos devem ser interpretados, baseando-se nos objetivos da análise das arquiteturas. O Plano de Testes (Seção 3.3) exibe como será realizada a coleta dos dados, descrevendo o ambiente, cenário, critérios e testes os quais serão realizados para obter os objetivos deste trabalho.

3.1 Proposta

Tendo analisado os trabalhos relacionados (Seção 2.7) 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 recursos computacionais das arquiteturas Rudy (Subseção 2.5.1), Salz (Subseção 2.5.2) e Willson (Subseção 2.5.3),usando os seguintes critérios:

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 microsserviço, utilizando valores absolutos. Também 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 3.3.2). 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.

3.2 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 serão guiadas 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 na qual exija armazenamento em memória.
3. **Vazão Rede - Entrada**: Espera-se estressar com requisições na qual tenham uma carga de dados elevada.
4. **Vazão Rede - Saída**: Espera-se estressar com respostas na qual 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 padrão, todos os recursos não são estressáveis, com um número de conexões simultâneas elevado. Porém, espera-se para o atual 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 3.1.

Tabela 3.1: 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 a qual poderiam estar em outros microsserviços
		↑		↓			Uma entrada de dados elevada pode indicar o uso de um protocolo indapropriado para o serviço
			↑	↓			Caso a saída esteja muito elevada pode indicar uma má configuração de elementos que são transitados na rede ou uso inadequado de protocolos
				↓	↑		Pode estar relacionado a 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
↑	↑	↑	↑	↑	↑	↑	Limite de processamento da arquitetura
↓	↓	↓	↓	↑	↓	↓	Teste padrão

Fonte: O próprio autor.

A Tabela 3.1 relaciona os recursos conforme dois padrões:

- Valores acima da média (↑).
- Valores baixo da média (↓).

Espera-se encontrar problemas mais detalhados, além de problemas padronizados de forma genérica na Tabela 3.1. Pode ser possível identificar eventuais problemas

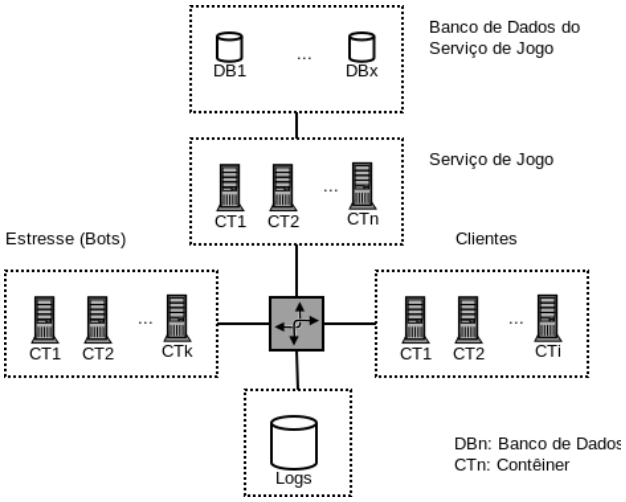
conforme o tipo de requisição e projeto da arquitetura. Estes problemas servirão para guiar a análise final das arquiteturas.

Seguindo estes critérios da análise, faz-se necessário definir um plano de testes para obter os dados conforme os cenários e casos de uso do atual trabalho.

3.3 Plano de testes

O plano de testes define os cenários que serão aplicados sobre todas as arquiteturas de microsserviços para jogos MMORPG. O plano de testes serve para descrever formas de estressar as arquiteturas, a fim de obter valores para análise. Entretanto, antes de relatar os cenários de teste, faz-se necessário descrever o ambiente no qual serão realizados os experimentos. A Figura 3.1 descreve a infraestrutura utilizada para execução das camadas de aplicação utilizados nos testes.

Figura 3.1: Ambiente de testes definido para a coleta de dados.



Fonte: O próprio autor.

Como visível na Figura 3.1, o ambiente de testes planejado está organizado em cinco regiões. Essas regiões foram segregadas 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.

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 suíte de estresse não interfira nas métricas obtidas no lado do cliente.

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, para utilizar na análise.

Tais regiões da infraestrutura utilizada no ambiente de testes devem manter um padrão a fim que não exista interferência entre os testes, além do desempenho das arquiteturas do serviço. 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 ambas as 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 basea-se na segregação de jogadores por canais;

2. Arquitetura Salz (Subseção 2.5.2), na qual basea-se em gerar muitos serviços escaláveis; e

3. Arquitetura Willson (Subseção 2.5.3), na qual basea-se em extrair microsserviços de regras de negócio mais custosas.

Tais arquiteturas vão impactar o serviço de jogo, banco de dados e as requisições a qual 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.

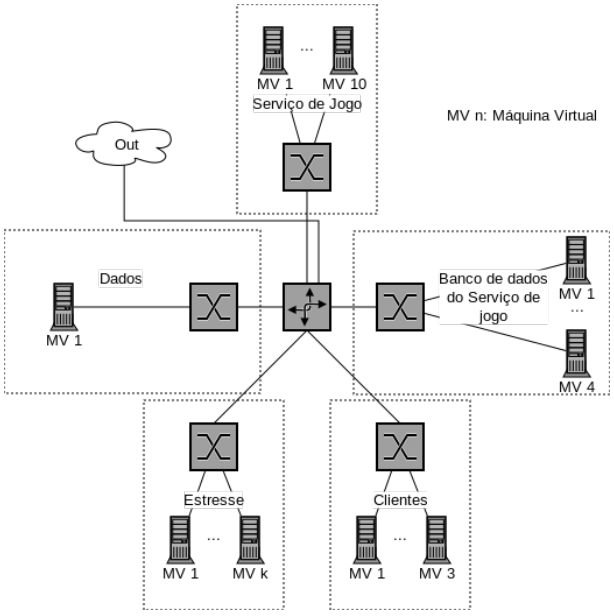
Com o objetivo de obter dados, faz-se necessário estressar as arquiteturas em casos diferentes para garantir a confiabilidade dos dados obtidos. Dessa forma, foi desenvolvido um cenário que comporte ambas as arquiteturas de microsserviços propostas a análise. Este cenário possibilita a execução do experimento junto a simulação de clientes.

3.3.1 Cenário

O cenário reflete diretamente sobre o ambiente proposto para esta análise. Ela será executada sobre 5 camadas, nas quais cada camada estará isolada em sub-redes em uma nuvem de computadores.

O cenário será composto por 5 sub-redes, na qual cada uma será responsável pela operação de uma região do ambiente planejado (Figura 3.2). 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.

Figura 3.2: Rede de execução dos testes



Fonte: O próprio autor.

O cenário, exibido na Figura 3.2 descreve cinco sub-redes. Estas redes são descritas da seguinte forma:

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 do 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 3.2.

Tabela 3.2: Tabela de interdependência das sub-redes.

Linha depende de Coluna	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.

A Tabela 3.2 refere-se a dependência das redes, conforme os serviços necessários para o seu funcionamento. Dessa forma, espera-se bloquear o tráfego de pacotes entre redes que não são dependentes.

A implantação do serviço de jogo utilizará métodos de implantação de micros-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 feitas diretas no sistema operacional da máquina virtual. Dessa forma, segue-se 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 requiridos 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 3.3. Nesta Tabela 3.3 também é definido a faixa de endereços de cada rede esperado.

Tabela 3.3: 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 3.3, 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. Neste sentido, se faz necessário definir características mínimas de regra de negócio, padrão de comportamento e interface esperada para o serviço e cliente.

3.3.2 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 comportamento aleatório.

Porém, como requisito para estipular as requisições, faz-se obrigatório reali-

zar um levantamento de requisitos na qual tanto o serviço quanto o cliente devem implementar. Nesse sentido, a Tabela 3.4 relaciona a funcionalidade com o impacto de implementação de tal funcionalidade.

Tabela 3.4: Requisitos funcionais e impacto de implementação

Requisito	Descrição	Implementação
Identificação	Gera uma numeração única (token) com base em uma tupla de dados.	Será implementado utilizando algoritmo de hash (MD5), 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.
Selecionar Personagem	Uma conexão deve requirir 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.
Chat	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.
Movimentar	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.
Atacar	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.
Consumir	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.

A Tabela 3.4 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 3.5.

Tabela 3.5: Requisitos mínimos para a implementação da simulação descrita

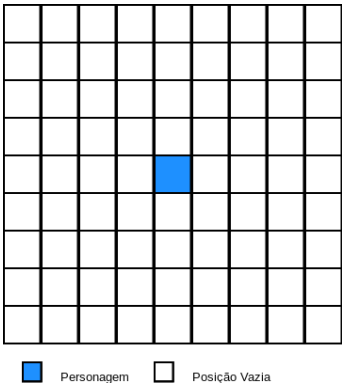
Nome	Argumentos	Retorno	Protocolo	Direção
Auth	*Username, Password*	JSON	*Web*	Cliente para Serviço
CreateAccount	*Username, Password*	JSON	*Web*	Cliente para Serviço
UpdateAccount	*Username, Password*	JSON	*Web*	Cliente para Serviço
CreateCharacter	*Token, Character Name*	JSON	*Web*	Cliente para Serviço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

A Tabela 3.5 descreve todos os comando que estarão disponíveis na rede. Além destes, serão implementados outros comandos para API privada do serviço. No entanto, os

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, a qual facilite calculos internos do serviço e cliente, promovam 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 3.3.

Figura 3.3: Área de interesse da simulação com raio de 4 células



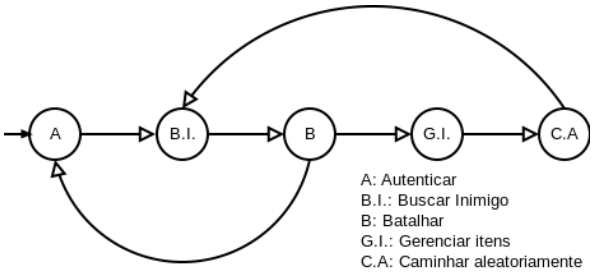
Fonte: O próprio autor.

A partir da área de interesse do jogo, como o da Figura 3.3, o *bot* poderá decidir suas ações baseado em um automato. 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 automato. Nesse sentido, espera-se obter um padrão de movimentação a fim de evitar ciclos a qual 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 automato descrito na Figura 3.4.

A Figura 3.4 descreve o comportamento de um *bot* no ambiente do jogo simulado. 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 é

Figura 3.4: Automato de movimentação dos personagens simulados



Fonte: O próprio autor.

definido 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 *Non-Playable Character* (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 explorar o 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% 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 automato 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.

3.3.3 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 a arquitetura Rudy com zero jogadores simultâneos, por 5 minutos;
2. Executar a arquitetura Rudy com zero jogadores simultâneos, aumentando a cada minuto 10 jogadores ao serviço, até o serviço obter algum microsserviço terminar a sua execução por erro interno;
3. Executar a arquitetura Salz com zero jogadores simultâneos, por 5 minutos;
4. Executar a arquitetura Salz com zero jogadores simultâneos, aumentando a cada minuto 10 jogadores ao serviço, até o serviço obter algum microsserviço terminar a sua execução por erro interno;
5. Executar a arquitetura Willson com zero jogadores simultâneos, por 5 minutos;
6. Executar a arquitetura Willson com zero jogadores simultâneos, aumentando a cada minuto 10 jogadores ao serviço, até o serviço obter algum microsserviço terminar a sua execução por erro interno.

A partir desta sequência de testes, é obtido valores suficientes para a análise das arquiteturas de microsserviços específico a jogos MMORPG. Eles serão utilizados para análise conforme os critérios definidos, buscando possíveis gargalos e problemas de escalabilidade do sistema.

3.4 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 podem ser 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.

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. Foi projetado 5 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. Por fim, foi definido regras de negócio básicas para os clientes simulados, baseados em um automato.

Os testes que serão executados sobre as redes são baseados obter valores dos recursos mínimos para execução dos serviços e o seu crescimento conforme um número de clientes simultâneos.

4 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 elementos em sua arquitetura a fim de suprir a demanda do mercado para estes jogos. Sendo assim, para entender melhor as características de tais arquiteturas, este trabalho propõe a realização de uma análise sobre arquiteturas distintas expressas na literatura.

O levantamento bibliográfico é um dos passos iniciais para guiar este objetivo. Nessa etapa apresentou-se conceitos de jogos MMORPG, padrões de projetos para tais jogos, protocolos utilizados e técnicas de desenvolvimento. Sobre estes conceitos foi descrito o problema referenciado neste TCC, no qual aplicaram-se alguns dos conceitos descritos anteriormente.

Após, foram analisados trabalhos relacionados cujos os objetivos são de análise de arquiteturas de microsserviços ou análise de arquiteturas de jogos MMORPG. Para guiar o inicio da proposta, foi descrito as arquiteturas encontradas na literatura, na qual serão utilizadas no plano de testes do atual trabalho. Na segunda parte da proposta, foi apresentado o plano de testes a fim de dar suporte a análise das arquiteturas.

Para o TCC-II será desenvolvido as arquiteturas de microsserviços para jogos MMORPG descritas na proposta do atual trabalho. Também será desenvolvido testes de carga automatizado a fim de facilitar a etapa de testes das arquiteturas. Por fim, os dados 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. Espera-se obter características específicas diferentes de cada arquitetura durante esta análise, casos de uso onde cada arquitetura seja aplicável e solução de gargalos encontrados ns arquiteturas descritas.

4.1 Cronograma

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

4.2 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 definir das arquiteturas de microsserviços descritas na literatura, bem como os seus objetivos;
4. **Especificação das arquiteturas selecionadas:** Especificar o funcionamento das arquiteturas selecionadas.
5. **Identificação e definição de simulações aplicáveis ao teste:** Eleger e definir a simulação a ser aplicada nos testes;
6. **Especificação da simulação elegida:** Especificar os requisitos;
7. **Escrita Trabalho de Conclusão de Curso I**;

4.3 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 referênciada:** Aplicação das arquiteturas descritas sobre uma nuvem computacional;

11. **Realização dos testes utilizando a simulação elegida na pesquisa referênciada:** Execução de testes da arquitetura desenvolvida sobre a nuvem computacional;

12. **Análise das arquiteturas testadas:** Analisar as métricas obtidas dos testes e descrever resultados, identificando possíveis gargalos nas arquiteturas;

13. **Otimização para melhorar as métricas obtidas:** Identificar pontos de gargalo nos microsserviços identificados e propor soluções viáveis para aumentar o desempenho desses sistemas.

14. **Escrita Trabalho de Conclusão de Curso II**;

4.4 Execução do cronograma

Etapas	2018												2019											
	J	F	M	A	M	J	J	A	S	O	N	D	J	F	M	A	M	J	J	A	S	O	N	D
1								X																
2								X																
3								X	X															
4								X	X															
5										X														
6										X														
7								X	X	X	X													
8																								
9																								
10																								
11																								
12																								
13																								
14																								

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. 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; accessed 8. May 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; accessed 17. Apr. 2018. Disponível em: <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; accessed 1. Sep. 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>.

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; accessed 8. May 2018]. Disponível em: <https://www-.battlefield.com/pt-br>.

EA. *FIFA 18 - Soccer Video Game - EA SPORTS Official Site*. 2018. [Online; accessed 8. May 2018]. Disponível em: <https://www.easports.com/fifa>.

Exit Games. *Photon Unity Networking*. 2017. [Online; accessed 15. May 2018]. Disponível em: <https://doc-api.photonengine.com/en/pun/current/index.html>.

Exit Games. *Serialization in Photon Engine*. 2018. [Online; accessed 29. Aug. 2018]. Disponível em: <https://doc.photonengine.com/en-us/realtime/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>.

GOLDSMITH, T. *"Cathode-ray tube amusement device"*. 1947. "Online; accessed 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; accessed 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*. [S.l.: s.n.], 2014. 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. p. 512–518.

HUANG, J.; CHEN, G. Digtal stb game portability based on mvc pattern. In: *2010 Second World Congress on Software Engineering*. [S.l.: s.n.], 2010. v. 2, p. 13–16.

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; accessed 19. May 2018]. Disponível em: <https://tools.ietf.org/html/rfc4506.html>.

JAJEX. *RuneScape Online Community*. 2018. Acessado em: 01/02/2018 ás 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; accessed 1. May 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. p. 261–268.

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; accessed 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; accessed 15. May 2018]. Disponível em: <http://docs.godotengine.org/en/3.0>.

MDHR. *Cuphead*. 2017. [Online; accessed 8. May 2018]. Disponível em: <https://store-.steampowered.com/app/268910/Cuphead>.

MICROSOFT. *Age of Empires III - Age of Empires*. 2005. [Online; accessed 8. May 2018]. Disponível em: <https://www.ageofempires.com/games/aoeiii>.

MOJANG. *How do I play multiplayer?* 2009. [Online; accessed 8. May 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>.

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; accessed 8. May 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; accessed 29. Aug. 2018]. Disponível em: <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; accessed 8. May 2018]. Disponível em: <https://eurotrucksimulator2.com>.

SPORTV. *League of Legends ganha torneio de fim de ano organizado pela ABCDE*. 2018. [Online; accessed 17. Apr. 2018]. Disponível em: <https://sportv.globo.com/site-/e-sportv/noticia/league-of-legends-ganha-torneio-de-fim-de-ano-organizado-pela-abcde-.ghtml>.

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; accessed 19. Apr. 2018]. Disponível em: <https://www.statista.com/statistics/267190/traffic-forecast-for-internet-gaming>.

STATISTA. *Global PC/MMO revenue 2015*. 2018. [Online; accessed 1. May 2018]. Disponível em: <https://www.statista.com/statistics/412555/global-pc-mmo-revenues>.

STATISTA. *LoL player share by region 2017*. 2018. Online; accessed 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*. [S.l.: s.n.], 2012. p. 148–149.

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.

XEROX. *High-level framework for network-based resource sharing*. 1976. [Online; accessed 19. May 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*. [S.l.: s.n.], 1996. p. 175–187.