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, Junho 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*

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*

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*

TCP *Transmission Control Protocol*

TPS *Third-person Shooter*

UDP *User Datagram Protocol*

WAN *Wide Area Network*

WS *Web Services*

XDR *External Data Representation*

XML *Extensible Markup Language*

1 Introdução

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2 Fundamentação Teórica

O termo de jogos eletrônicos é amplamente difundido, entretanto as especificações, características e histórico deste termo não é de conhecimento popular. A Seção 2.1 tratará a definição de jogo eletrônico, um levante histórico e o impácto da evolução do hardware no desenvolvimento dos jogos. Este termo será tomado como introdução para o conceito de gênero de jogo, abordado na Seção 2.1.1, na qual serão 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 tenta 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 abordará sobre uma introdução ao impácto 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, se faz obrigatório o entendimento acerca da 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) serão 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 2.5) a arquitetura de jogos MMORPG desenvolvidos de forma distribuída ou sobre uma arquitetura de microsserviços. Esta seção em específico abordará exemplos de métodos e métricas utilizadas para menssurar o desempenho de tais arquiteturas, realizando por fim uma análise destes trabalhos (Seção 2.5.4).

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 desenhadas diretamente para algum hardware proprietário, sem sistema operacional, por muitas vezes sem utilizar comunicação por rede ou memória de disco (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 de máquinas distintas (STATISTA, 2018a). Jogos como EA Habitat[1], CipSoft Tibia[2] e Jajex Runescape[3] começam a utilizar, como requisito obrigatório do jogo, a conexão com a Internet para interagir em um mundo compartilhado com outos 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**

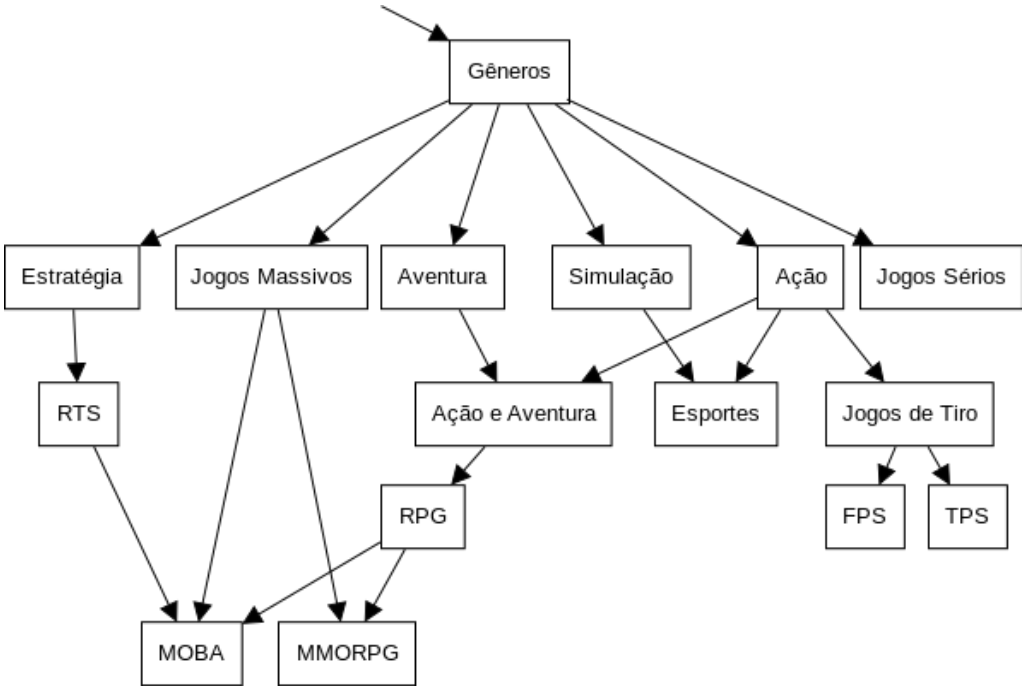
2013; HUANG; YE; CHENG, 2004), criando o gênero de jogos *Massively Multiplayer Online* (MMO) na árvore de gêneros.

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).

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 analisado (ADAMS, 2006; HANNA, 2015). A árvore pode ser visualizada pelo diagrama na Figura 2.1. O contexto breve de cada gênero é:

- 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 *Web Services* (WS).

  - *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 outros jogadores também estão jogando, na qual a interação entre outros jogadores (herdado dos jogos MMO), com o mundo (herdado dos jogos de ação

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

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 WS.
  - 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 WS.
  - 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

[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**

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 cameras 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 WS.

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).

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

	LAN	P2P	WAN	WS	Jogadores
ESTRATÉGIA	✓	✓	✓		até 10 (MICROSOFT, 2005)
RTS	✓				até 10 (BLIZZARD, 2010)
MMO			✓	✓	mais que 1000 (JAJEX, 2018)
MOBA	✓	✓	✓		até 10 (RIOT, 2009)
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.

Dentre todos os jogos, o gênero MMORPG é o mais impactado pela quantia de jogadores(KIM; KIM; PARK, 2008), visível na Tabela 2.1. Por esse motivo, a escolha por abordar o gênero MMORPG se torna interessante do ponto de vista computacional, a fim de analisar o comportamento de rede e processamento das arquiteturas desses jogos.

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 realizará as requisições com a interface do serviço, exibindo o estado de jogo de forma imersiva ao jogador. Este tema será melhor abordado na Seção 2.4.1.
- **Servidor**: Conjunto de computadores que recebe as requisições do cliente a fim de ser 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 será melhor abordado na Seção 2.4.2.

A maioria dos jogos MMORPG disponíveis no mercado estão implementados sobre uma arquitetura que executa sobre diversos servidores (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.

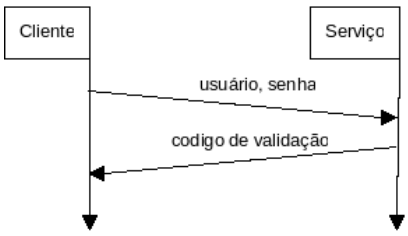
Nesse sentido torna-se necessário descrever as características de jogabilidade de jogos MMORPG a fim de melhor compreender o funcionamento da arquitetura de um cliente e de um serviço para jogos MMORPG nas seções seguintes.

2.3 Jogabilidade de jogos MMORPG

É comum em jogos MMORPG ter sistemas com funcionalidades parecidas. Por esse motivo é fácil identificar os sistemas em jogos do mercado após a compreensão dos sistemas identificados. Essa seção está presente a fim de 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 que inicia o cliente de algum jogo MMORPG (SALZ, 2016; 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.

Figura 2.2: Sistema de autenticação para jogos



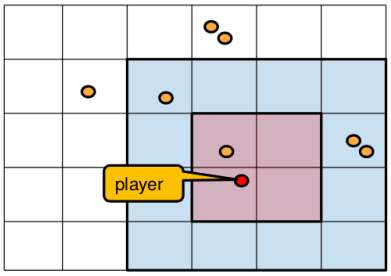
Fonte: Adaptado de (THOMPSON, 2008)

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, ele 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, 2016).

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



Fonte: (SALZ, 2016)

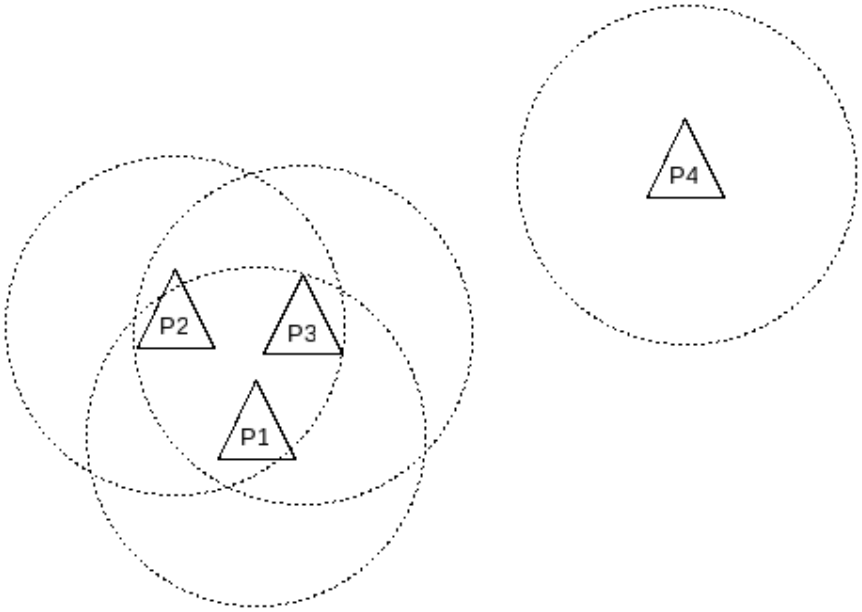
Esse caso pode ser visualizado na Figura 2.3, na qual o personagem selecionado (destacado em vermelho) tem uma área de interesse de baixa distância e uma área de interesse de longa distância (SALZ, 2016), sendo que o jogador não tem informações dos demais objetos e jogadores fora de sua área de interesse. 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, 2016).

Algumas ações comuns dentro do ambiente de um jogo MMORPG (JON, 2010):

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

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

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



Fonte: Adaptado de (SALZ, 2016)

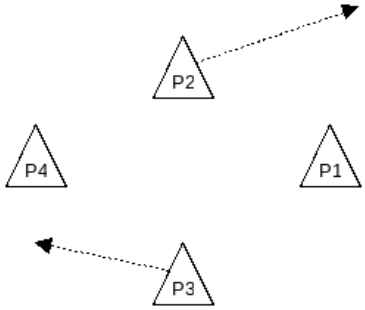
Essa distância (Figura 2.4) pode ser calculada utilizando Distância Euclidiana (DEZA, 2009), na qual a distância entre dois personagens podem ser calculadas pela equação $d(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2}$. Para diminuir a complexidade das comparações, a fim de decidir quais personagens P_n 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.

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 inúmeras consultas de proximidade, além de necessitar uma frequência de atualização contante pela qualidade da jogabilidade (SALZ, 2016).

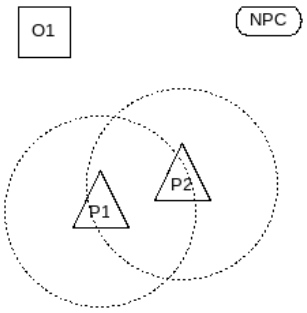
A interação com o ambiente, itens, NPCs e outros jogadores também são afetados pela área de interesse, na qual o personagem terá um raio limitante para cada tipo de interação no ambiente. Um exemplo de ambiente com personagens (P_1 e P_2), objetos (O_1) e NPCs (NPC somente) pode ser visualizado na Figura 2.6 (SALZ, 2016).

Figura 2.5: Personagens e os seus pontos de destino



Fonte: O próprio autor.

Figura 2.6: Personagens, objetos e NPCs em no ambiente



Fonte: O próprio autor.

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

2.4 Problemas em jogos MMORPG

Uma métrica popular para mensurar o desempenho de um serviço MMORPG é o número de conexões (HUANG; YE; CHENG, 2004) simultâneas suportadas. Em geral, caso o serviço ultrapasse o limite para o qual 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.

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 pilar fundamental 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, 2016). Um exemplo de cliente MMORPG é o jogo Sandbox-Interactive Albion[10], que pode ser visualizado na Figura 2.7.

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

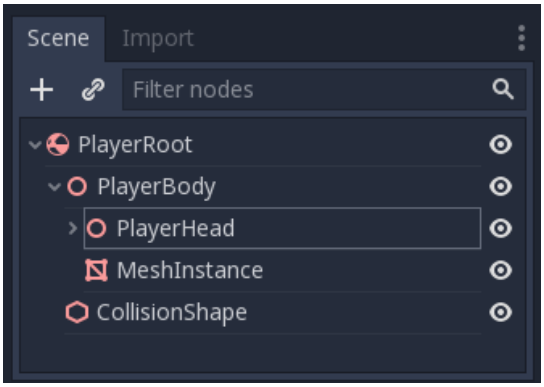
Fonte: (SALZ, 2016)

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 intersão e deleção, 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

[10]Sandbox-Interactive Albion: **https://albiononline.com/en/home**

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.

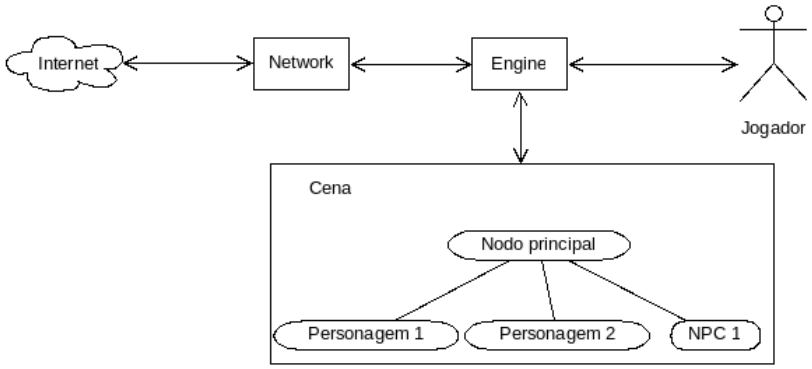
Dentro dessa árvore, cada nodo tem uma funcionalidade específica. Essas funcionalidades são específicas a cada motor gráfico, podendo existir nodos específicos a física, renderização ou controles (LINIETSKY, 2018). Em um jogo MMORPG, o serviço será responsável por enviar atualizações dos parâmetros aos 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, 2016). 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, 2016; WILLSON, 2017).

O módulo de *Network* implementado em um cliente de jogo MMORPG é responsável por realizar as requisições conforme as ações requiridas pelo jogador ao serviço, além de aplicar os parâmetros na *Scene Tree* ou chamar métodos remotos no cliente por ordens do serviço (SALZ, 2016). Ele pode ser encontrado na Figura 2.9.

A Figura 2.9 refere-se a uma visão macro de um cliente MMORPG. Nesta figura é possível ver o ator *player* na 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).

Figura 2.9: Modelo de um cliente genérico.



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

Para facilitar o desenvolvimento, a aplicação de cliente é dividida em diversos módulos, entretanto são relevantes ao atual trabalho (SALZ, 2016):

- **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[11] e GodotEngine[12] 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, se faz 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 (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).

[11]Unity3D: **https://www.unity3d.com**
[12]GodotEngine: **https://www.godotengine.org**

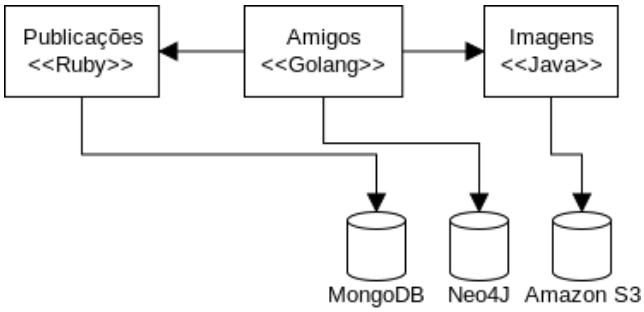
Arquiteturas de microsserviços iniciam uma nova linha de desenvolvimento de aplicações preparadas para executar sobre nuvens computacionais, promovendo maior flexibilidade, escalabilidade, gerenciamento e desempenho, sendo a principal escolha de arquitetura de grandes empresas como Amazon, Netflix e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

- Deve possibilitar a implementação como uma peça individual do macrosserviço.
- Deve funcionar individualmente.
- Cada serviço deve ter uma interface. Essa interface deve ser o suficiente para utilizar o microsserviço.
- 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 como um sistema independente em um *Platform as a Service* (PaaS). 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 contenha tecnologias distintas que melhor resolvam os problemas relacionados ao contexto deste microsserviço. Isso pode ser visualizado na Figura 2.10.

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

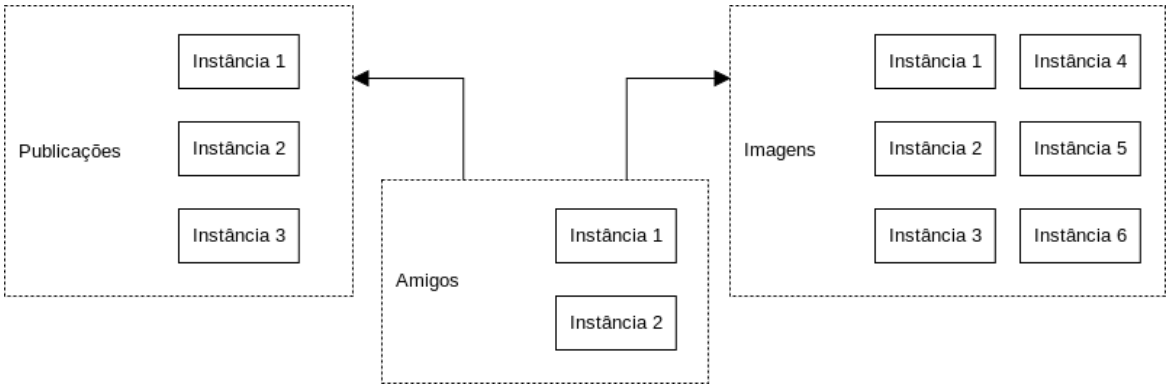
Figura 2.10: Microsserviços podem ter diferentes tecnologias



Adaptado de: (NEWMAN, 2015)

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 se faz necessário ver as diferenças entre uma arquitetura de microsserviços para MMORPG comparados a microsserviços web.

2.4.3 Microsserviços 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, 2016).

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* teremos operações padrões aplicados sobre *Models* da aplicação. Esses métodos padrões seguem o protocolo CRUD, contendo 4 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.

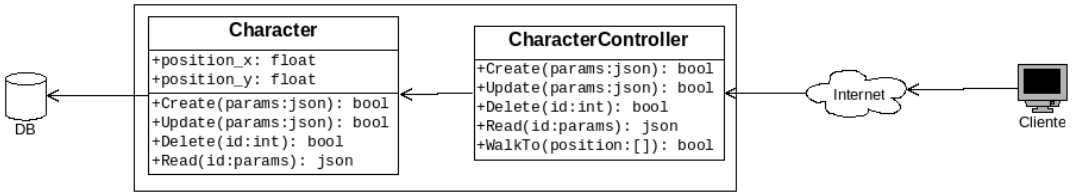
2. *Update*: Representa a atualização de um objeto no banco.

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

4. *Read*: Representa a consulta sobre este objeto no banco.

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

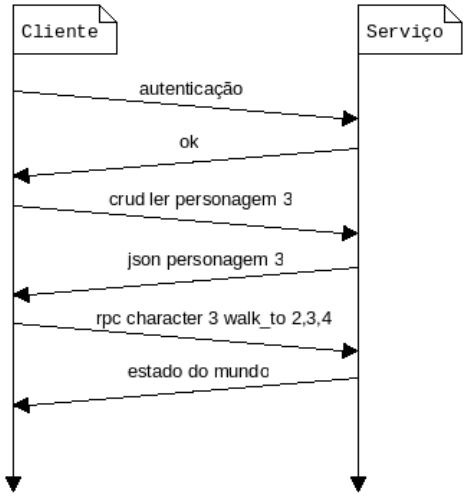
Figura 2.12: Cliente pode realizar requisições CRUD ao serviço



Fonte: Adaptado de (SALZ, 2016).

Essas operações são executadas utilizando protocolo RPC (SALZ, 2016). 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 2.13: Diagrama de requisições entre serviço e cliente com operações CRUD e RPC em uma arquitetura monolítico.



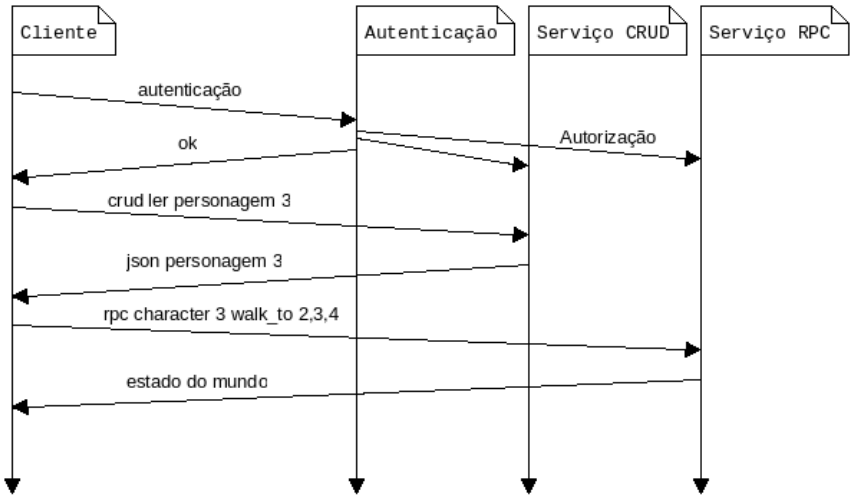
Fonte: Adaptado de (THOMPSON, 2008)

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 requiridas 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 compreendida pela Figura 2.14.

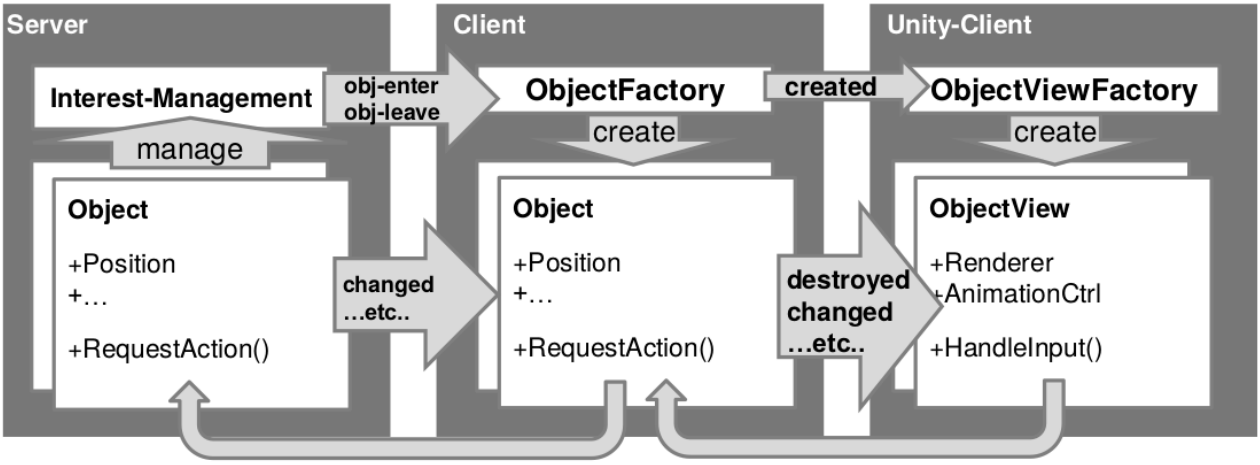
Figura 2.14: Diagrama de requisições entre serviço e cliente com operações CRUD e RPC em uma arquitetura de microsserviços.



Fonte: O próprio autor.

Como resultado da integração entre Cliente, Serviço e Motor Gráfico, o resultado final obtido é descrito pelo diagrama presente na Figura 2.15.

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



Fonte: (SALZ, 2016)

A integração descrita na Figura 2.15 está presente no jogo Sandbox-Interactive Albion[13] (SALZ, 2016). Utilizando esse embasamento teórico sobre microsserviços e arquiteturas de jogos MMORPG, pode-se analisar os trabalhos (Seção 2.5) relacionados com o tema proposto no atual documento.

[13]Sandbox-Interactive Albion: **https://albiononline.com/en/home**

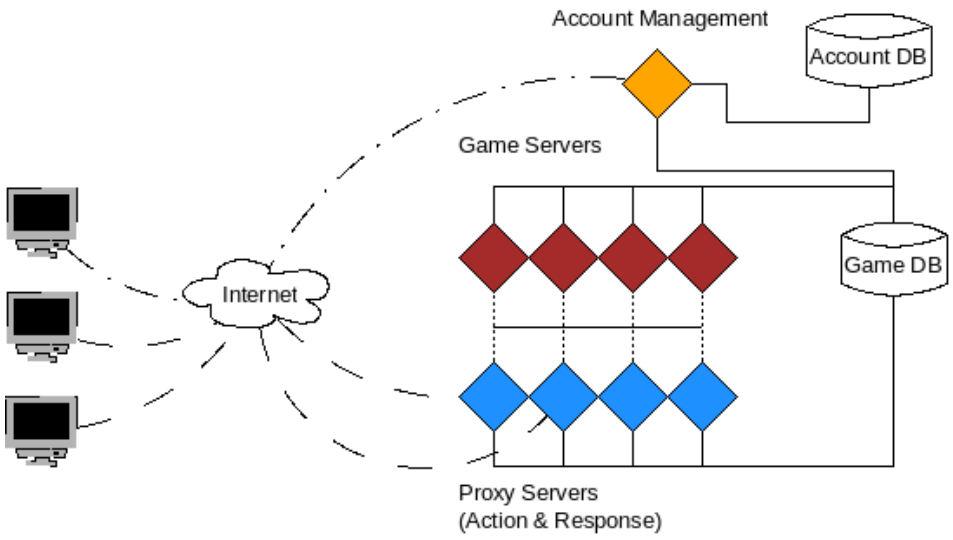
2.5 Trabalhos Relacionados

Para nortear o desenvolvimento da análise se 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.

2.5.1 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 2.16: Arquitetura distribuída utilizando proxy

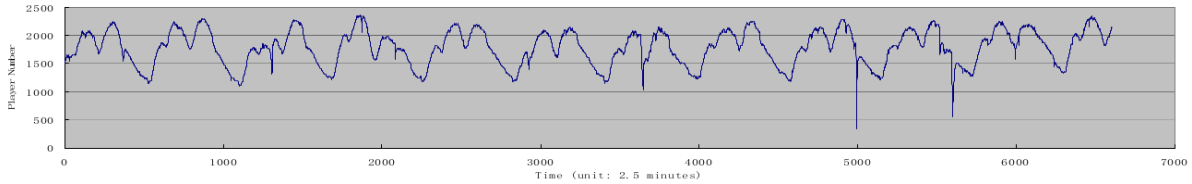


Adaptado de: (HUANG; YE; CHENG, 2004)

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 2.16, na qual obteve seus dados obtidos durante 100 dias para realizar as análises. A Figura 2.17 demonstra uma amostra do número de conexões pelo tempo no serviço obtido.

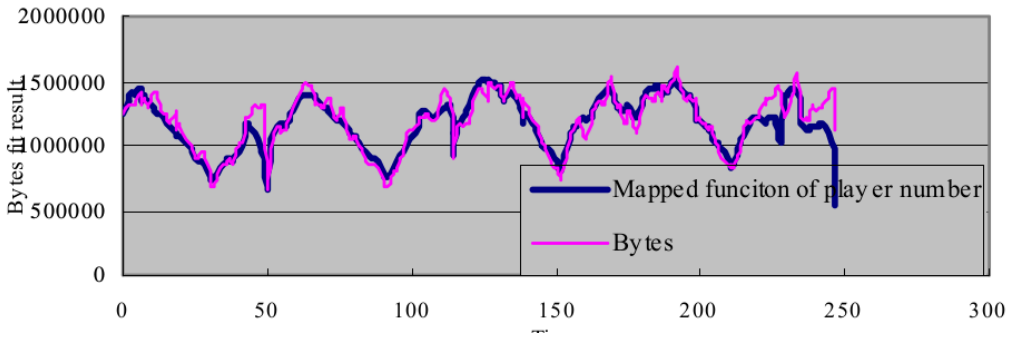
Figura 2.17: Número de conexões no serviço pelo tempo decorrido.



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

Figura 2.18: Regressão linear comparado ao consumo de banda real do servidor.



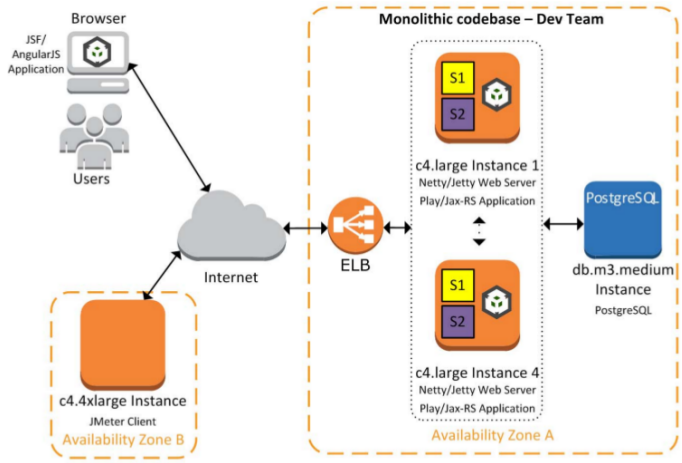
Fonte: (HUANG; YE; CHENG, 2004)

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.

2.5.2 Villamizar et al. (2016)

O trabalho de (VILLAMIZAR et al., 2016) investiga o custo de arquiteturas de microsserviços, arquiteturas 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.

Figura 2.19: Arquitetura monolítica web implementada na AWS



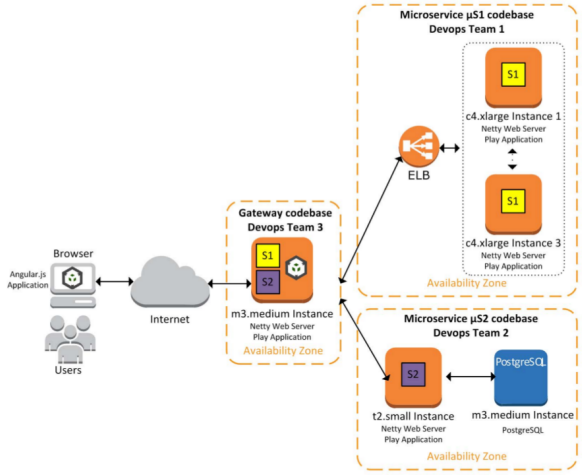
Fonte: (VILLAMIZAR et al., 2016)

Instância I. Utilizando quatro instâncias AWS *c4.large*, uma instância AWS *c4.xlarge* e uma instância AWS *db.m3.medium*. A Figura 2.19 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 2.20 exibe a implantação de uma aplicação de microsserviços web. Essa arquitetura foi implementada utilizando *Play Framework*.

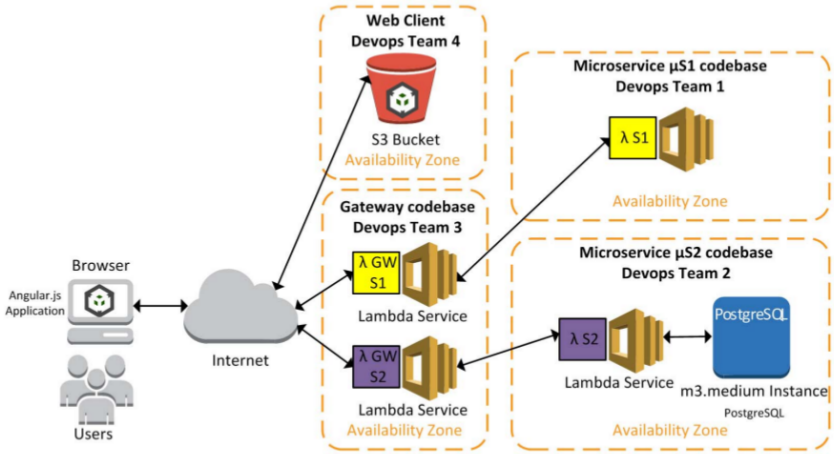
Instância III. Utilizando duas instâncias AWS *lambda S1*, duas instâncias AWS *lambda S2*, uma instância AWS *S3 Bucket* e uma instância AWS *db.m3.medium*. A Figura 2.21 exibe a implantação de uma aplicação de microsserviços web utilizando a tecnologia AWS *lambda*. Essa arquitetura foi implementada em *Node.js*, onde as funções de *gateway* foram implementadas em quatro funções independentes do tipo *microservice-http-endpoint*.

Figura 2.20: Arquitetura de microsserviços web implementada na AWS



Fonte: (VILLAMIZAR et al., 2016)

Figura 2.21: Arquitetura de microsserviços web implementada na AWS utilizando a tecnologia *lambda*



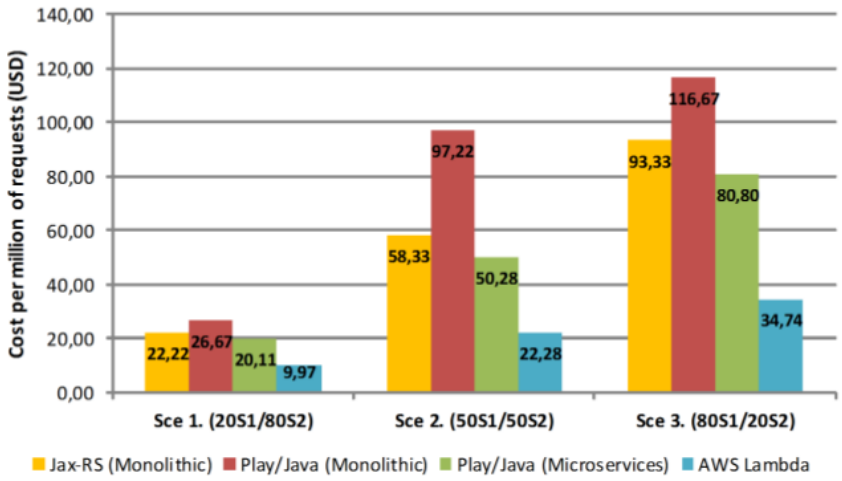
Fonte: (VILLAMIZAR et al., 2016)

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 2.22. O autor alerta sobre tolerância a falhas, transações distribuidas, distribuição de dados e versionamento de serviço.

2.5.3 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 perso-

Figura 2.22: Custo por um milhão de requisições em dólares utilizando diferentes arquiteturas sobre a AWS



Fonte: (VILLAMIZAR et al., 2016)

nagem pode realizar em um ambiente. Este trabalho usa com base o modelo descrito na Seção 2.5.1, um modelo distribuído em serviços na qual efetuam o processamento de uma região do ambiente virtual.

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

Contexto da ação	PvP	PvNPCs	Number of 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

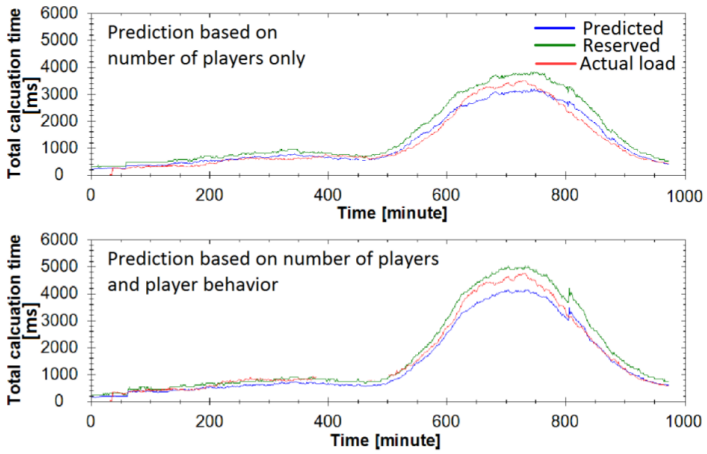
Fonte: (SUZNJEVIC; MATIJASEVIC, 2012)

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 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, onde multiplos jogadores unem forças a fim de combater outro grupo de jogadores ou NPCs.

Figura 2.23: Regressão levando em conta a complexidade das ações e contexto dos personagens



Fonte: (SUZNJEVIC; MATIJASEVIC, 2012)

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 2.23.

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.

2.5.4 Análise dos trabalhos relacionados

Em específico, nesta seção são apontadas características relevantes a cada trabalho e por fim é feita uma comparação entre estas características. Ao levantar o problema de consumo de recursos, existem duas vertentes de pesquisas, sendo descritas dessa forma:

- **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 alocações ociosas.

Dessa forma, podemos 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.

2.6 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 analises 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 de microsserviços MMORPG

Para realizar a análise de consumo de recursos de uma arquitetura de jogos MMORPG é necessário coletar / medir os recursos utilizados para posterior análise. Na fase de coleta de informação, é possível utilizar ferramentas existentes para coletar informações sobre o consumo da rede (*e.g.*, Tcpdump, Wireshark, etc.) e consumo de recursos computacionais do serviço (e.g., Golang Flame, Ruby Thread profiler tool, Golang profiler, *etc.*) Estas ferramentas contribuem com a análise, sendo de extrema importância para a coleta de dados. Este tópico será abordado na Seção refsec:metricas.

Contudo, se faz necessário uma arquitetura de microsserviços para jogos MMORPG a fim de ser utilizado para análise. Esta arquitetura será descrita na Seção 3.2.

3.1 Ferramentas de análise de consumo de recursos

Para efetuar a análise de uma arquitetura, de faz necessário um sistema de monitoramento a fim de obter métricas do uso de recursos do sistema. Para este fim, se faz necessário obter três classes de aplicação:

1. Banco de dados para métricas
2. Sistema de alimentação do banco de dados
3. Sistema de visualização do banco de dados

Nesse sentido, se faz necessário obter um banco de dados específico ao armazenamento de métricas. O banco *Graphite*[1] é específico a esta operação. Ele permite o envio de registros utilizando texto plano por chamadas de sistema, UDP ou TCP. Um exemplo de chamada de sistema pode ser visualizado na Figura 3.1.

[1]Graphite: **https://github.com/graphite-project**

Figura 3.1: Inserindo dados no Graphite utilizando chamada de sistema.

```
# Exemplo de chamada de sistema em Bash
$ echo "foo.bar 1 `date +%s`" | nc localhost 2003
```

Fonte: (Graphite, 2017).

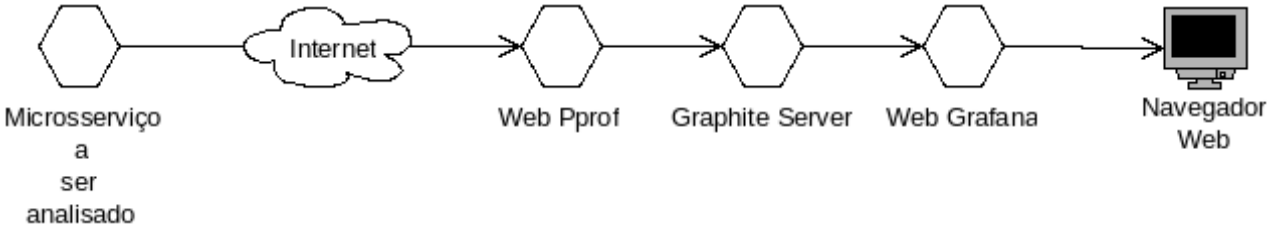
Para visualização dos dados, o sistema *web Grafana*[2] exibe os dados armazenados pelo *Graphite* em um *dashboard*. Este *dashboard* pode ser visualizado na Figura 3.2.

Figura 3.2: Dashboard de análise do Grafana.

Fonte: (Grafana, 2018).

Entretanto, se faz necessário o envio de métricas de uso de recursos pelos próprios microsserviços. Nesse sentidio, ferramentas de análise (*e.g.*, TCPDump, Go Pprofit, etc.) e informações do sistema operacional serão utilizadas para popular o banco de dados. A Figura 3.3 informa o funcionamento do serviço que receberá as métricas dos microsserviços.

Figura 3.3: Dashboard de análise do Grafana.



Fonte: (Grafana, 2018).

O microsserviço *Web Pprof* exibido na Figura 3.3 receberá requisições web com os seguintes parâmetros:

1. *service_id*: Referenciará qual microsserviço é o emissor desta requisição.

[2]Grafana: **https://github.com/grafana/**

2. *service_hash*: Servirá para autenticar a requisição do microsserviço.

3. *service_ram*: Valor de ponto flutuante que indicará a memória consumida, em Megabytes.

4. *service_bw*: Valor de ponto flutuante que indicará a o consumo de banda pelo serviço, em Megabytes.

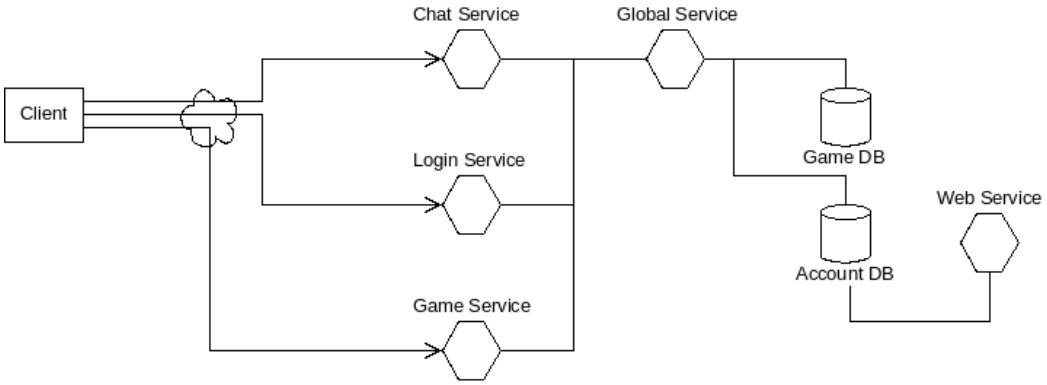
5. *service_disk*: Valor de ponto flutuante que indicará o consumo de disco, em Gigabytes.

Com estes parâmetros, será possível analisar o consumo de recursos e custo de operação da arquitetura analisada. Entretanto, faz necessário simular esta arquitetura. Nesse sentido, a descrição da arquitetura analisada é importante, a fim de descrever sua funcionalidade para que se comporte igual a uma arquitetura de microsserviços para jogos MMORPG.

3.2 Arquitetura proposta para análise

Esta seção tem como objetivo descrever a arquitetura implementada para análise, baseada na arquitetura Salz (SALZ, 2016). A mesma arquitetura é utilizada no jogo Sandbox-Interactive Albion[3]. Ela pode ser vista em uma visão macro na Figura 3.4.

Figura 3.4: Arquitetura de microsserviços proposta para análise.



Adaptado de: (SALZ, 2016).

Todos os elementos descritos na Figura 3.4 são microsserviços, e por este motivo devem funcionar de forma independente dos demais serviços. Essa arquitetura contém os seguintes microsserviços:

[3]Sandbox-Interactive Albion: **https://albiononline.com/en/home**

- **Login Service**: Responsável por gerenciar as conexões do jogo e fornecer autorização aos clientes para os demais microsserviços.

- **Chat Service**: Receberá mensagens e distribuirá aos demais jogadores da mesma região ou enviará mensagens diretas a outros jogadores. Ele receberá a conexão diretamente do jogador, e por sua vez será necessário utilizar a tecnologia JWT assinado pelo serviço *Login Service* para autenticar a conexão.

- **Game Service**: Gerenciará um pedaço do ambiente do jogo. Cada microsserviço desta categoria será responsável por um pedaço de ambiente do jogo, na qual controlará as ações dos personagens em relação a sua área de interesse. Ele receberá a conexão diretamente do cliente, e por sua vez usará a mesma técnica de autenticação do *Chat Service* para autenticar a conexão.

- **Global Service**: Este microsserviço resolve as requisições globais requisitadas pelo *chat service*, *login service* ou *game service*. Será responsável por orquestrar os microsserviços, gerenciar as mudanças de microsserviços dos clientes e obter métricas referentes ao jogo, como número de personagens, ações principais e coordenação geral do jogo. Além disso, ele será o gerente de operações CRUD ao banco de dados *Account DB*.

- **Web Service**: Servirá para gerênciar o banco de dados utilizando operações CRUD.

- **Game DB**: Banco de dados em memória utilizado pelos microsserviços para compartilhamento do estado de mundo.

- **Account DB**: Banco de dados persistente para armazenar o estado de jogo em disco.

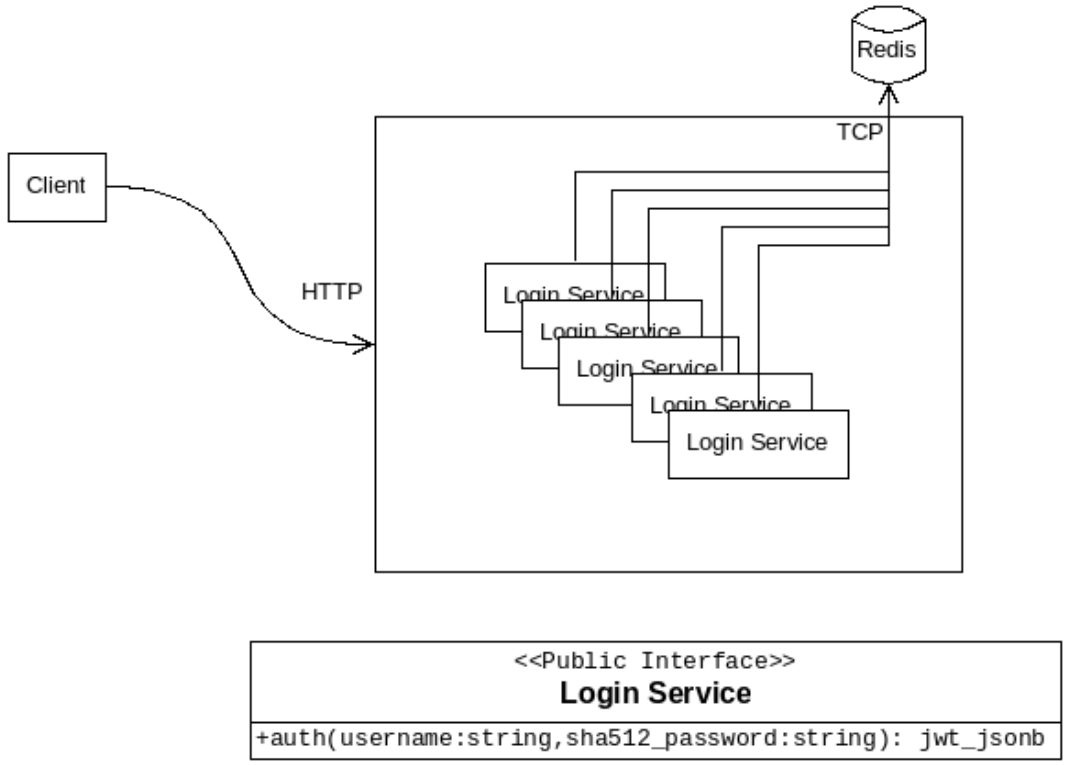
Para melhor compreendimento se faz necessário a descrição específica de funcionamento em cada microsserviço, tecnologias e protocolos utilizados e esquema de funcionamento.

3.2.1 Login Service

O microsserviço respoonsável por efetuar a autenticação do usuário aos demais microsserviços é nomeado *Login Service*, descrito na Figura 3.5. Ele será responsável por impedir

multiplas conexões utilizando um mesmo usuário, e assinar um token que será utilizado para autenticação nos demais microsserviços.

Figura 3.5: *Login Service*



Elaborado pelo autor.

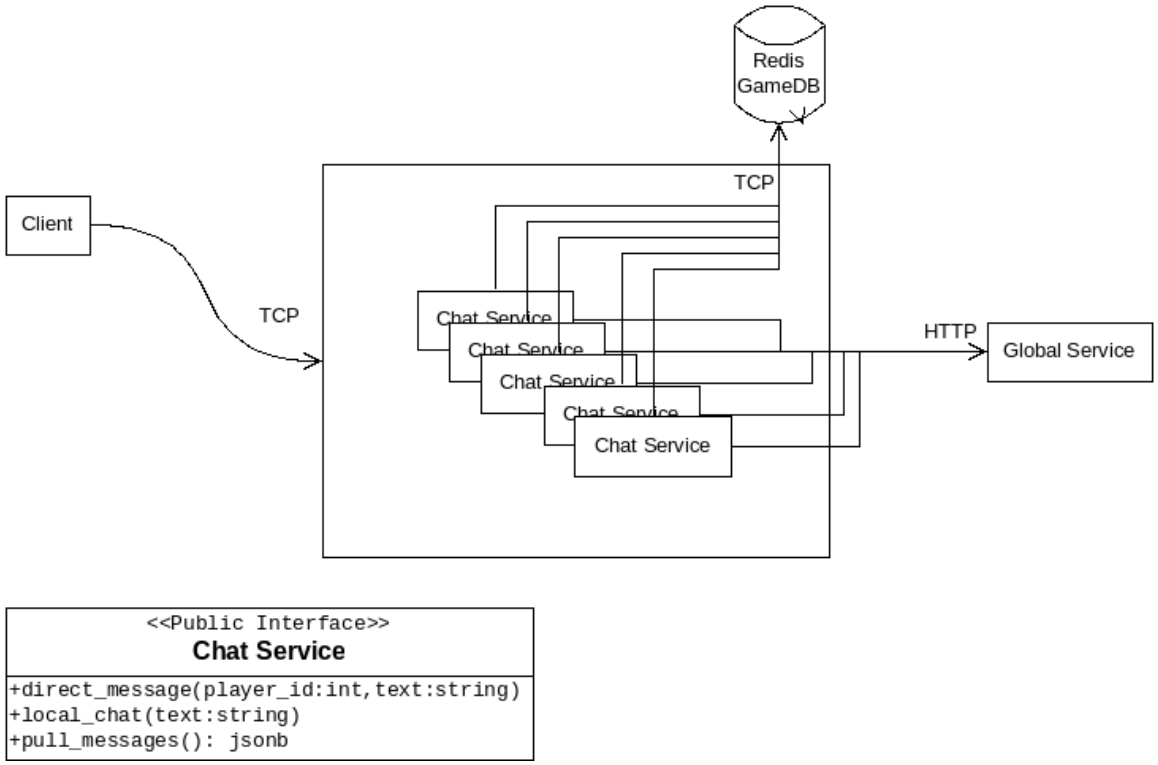
Além dessa funcionalidade, outro objetivo deste microsserviço é descrever o status de conexão de um jogador, a fim de sinalizar os demais microsserviços que o mesmo desconectou do serviço. Ele será o responsável pela descrição de status da atual conexão do jogador com o serviço visto que os demais microsserviços estão trabalhando de forma isolada. Isto servirá para evitar requisições maliciosas ao serviço. Nesse sentido, faz necessário que o *Login Service* comunique frequentemente os demais serviços de interesse para que atualize o status de conexão dos jogadores e suas credenciais de conexão, além de obrigar ao cliente uma requisição em um espaço determinado de tempo.

Este microsserviço usará Redis, um banco NoSQL baseado em chave-valor de armazenamento em memória para armazenar tokens de autenticação de uma conexão. Para gerar este token, será utilizado a tecnologia JWT, a fim que permita a assinatura de dados pelo serviço de autenticação e a validação nos demais microsserviços que tem interface pública na rede.

3.2.2 Chat Service

Este microsserviço será responsável por redirecionar mensagens a jogadores, e pode ser visualizado na Figura 3.6.

Figura 3.6: *Chat Service*



Elaborado pelo autor.

Este serviço terá três principais funções na interface pública:

- *local_chat*: Envia mensagens aos personagens que estão na área de interesse do remetente.
- *direct_message*: Envia mensagens diretas a outros personagens.
- *pull_messages*: Recebe mensagens não lidas até então.

Para compartilhamento de informações do ambiente de jogo, o microsserviço terá acesso ao GameDB a fim de obter informações de posicionamento dos personagens. Entretanto, a abordagem para mensagens diretas é mais complexa. Se faz necessário o redirecionamento das mensagens entre microsserviços a fim de entregar ao microsserviço onde o destinatário esteja conectado. Para este redirecionamento funcionar, será necessário uma tabela de localização da conexão deste jogador, a qual armazenará a relação entre

o personagem e o IP:Porta do microsserviço a qual ele está conectado. Por este motivo, o microsserviço terá acesso ao *GameDB*. No mesmo sentido, mensagens a jogadores offline não são entregues com roteamento, sendo necessário o armazenamento para entrega posterior, tendo esta requisição armazenada junto ao *Global Service*.

3.2.3 Game Service

Este microsserviço gerenciará conexões de uma determinada região do ambiente do jogo, nomeado *chunk*. Ele servirá para atualizar a árvore de cena do cliente e executar operações sobre a árvore de cena do chunk a qual o serviço ficou responsável. A abordagem de divisão em *chunks* serve para minimizar a influência de conexões sobre as operações no mundo, e assim permitir mais conexões sobre um único mundo compartilhado. Entretando é necessário tomar uma abordagem para definir a troca de contexto entre duas *chunks* no jogo. Neste microsserviço foi escolhida a abordagem sobre banco de dados (SALZ, 2016), onde o cliente terá que trocar de microsserviço conforme a sua localidade a troca de mapa e armazenar estra troca de dados no banco para que o microsserviço que controla esta região possa resgatar seus dados e prosseguir com o jogo.

3.2.4 Global Service

Este microsserviço realizará o meio de campo entre todos os microsserviços. Ele será responsável por sincronizar as posições dos personagens entre o Chat Service e Game Service, atualizar informações do GameService no AccountDB além de gerenciar o GameDB. Ele é uma aplicação *Web* e pode ser um gargalo na aplicação.

3.2.5 Web Service

Este microsserviço será responsável por exibir dados do banco aos jogadores. No contexto de simulação ele servirá para realizar operações CRUD manuais e automatizadas no banco, além de gerenciar migrações no mesmo.

3.2.6 GameDB

3.2.7 AccountDB

4 Considerações & Próximos passos

CONCLUSÃO

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>.

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.

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>.

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>.

Grafana. *Grafana*. 2018. [Online; accessed 28. May 2018]. Disponível em: <https:/-/github.com/grafana/grafana>.

Graphite. *Graphite*. 2017. [Online; accessed 28. May 2018]. Disponível em: <http:/-/graphiteapp.org/quick-start-guides/feeding-metrics.html>.

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>.

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>.

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>.

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.

WILLSON, S. C. *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>.

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.