Relatório Técnico: Dueling Protocol

Lucas Damasceno

[1]Universidade Estadual de Feira de Santana (UEFS)
Av. Transnordestina, s/n, Novo Horizonte
Feira de Santana – BA, Brasil – 44036-900

lucas.damasceno.dev@gmail.com

Resumo. *O mercado de jogos multiplayer online tem crescido exponencialmente, demandando arquiteturas de servidor cada vez mais robustas e de baixa latência para garantir uma experiência de usuário competitiva e justa. Este relatório detalha o "Dueling Protocol", uma solução de backend para um jogo de cartas 1v1, projetada para ser escalável, concorrente e confiável. A arquitetura cliente-servidor foi implementada em Java, utilizando comunicação nativa via sockets TCP para ações críticas de jogo e UDP para um sistema de ping, visando a monitoração de latência. A persistência de dados dos jogadores é realizada de forma leve e legível através de arquivos JSON. A lógica de negócio foi centralizada utilizando o padrão Facade, e a consistência em um ambiente de alta concorrência foi garantida pelo uso de estruturas de dados thread-safe. Para validar a solução, uma suíte de testes automatizados foi desenvolvida com Docker e Docker Compose, permitindo a simulação de múltiplos clientes e a identificação de condições de corrida. Testes de estresse demonstraram a capacidade do servidor de gerenciar dezenas de conexões simultâneas sem falhas, e os testes de cenário foram cruciais para identificar e corrigir uma condição de corrida no sistema de matchmaking, validando a robustez da implementação final.*

1. Introdução

Com a ascensão das plataformas de distribuição digital, o mercado de jogos independentes (*indie*) tornou-se um campo fértil para a inovação. No entanto, para se destacar, especialmente no gênero multiplayer, é imperativo que a infraestrutura de backend seja performática e confiável. Um servidor lento ou instável pode arruinar a experiência de jogo e afastar a base de jogadores. O desafio proposto neste projeto foi o desenvolvimento do backend para um jogo de cartas tático 1v1, que exigia pareamento de jogadores, comunicação bidirecional em tempo real e a gestão de múltiplas sessões de jogo simultâneas.

Para resolver este problema, foi criado o "Dueling Protocol", um servidor de jogo robusto e concorrente. A arquitetura da solução tem como ponto de entrada o `GameServer`, que aceita novas conexões de clientes e instancia um `ClientHandler` dedicado para cada um, adotando o modelo "uma thread por cliente". Cada `ClientHandler` é responsável por toda a comunicação com um cliente específico, processando seus comandos. As operações e a lógica de jogo são centralizadas no `GameFacade`, que atua como um orquestrador, delegando tarefas a serviços especializados, como o `ConcurrentMatchmakingService` para o pareamento de jogadores

e o `StoreServiceImpl` para a loja de pacotes de cartas. Essa abordagem desacopla a lógica de rede da lógica de negócio, facilitando a manutenção e a escalabilidade do sistema.

Este relatório está organizado da seguinte forma: a Seção 2 detalha a fundamentação teórica por trás das tecnologias e conceitos utilizados. A Seção 3 descreve a metodologia de implementação, alinhada com os requisitos do projeto. A Seção 4 apresenta e analisa os resultados dos testes executados. Finalmente, a Seção 5 conclui o trabalho, recapitulando os resultados e sugerindo direções para trabalhos futuros.

2. Fundamentação Teórica

Esta seção aborda os conceitos essenciais que serviram de alicerce para o desenvolvimento do Dueling Protocol.

2.1. Arquitetura Cliente-Servidor

O modelo cliente-servidor é um paradigma de arquitetura de software que distribui as responsabilidades entre dois processos independentes: o servidor, que provê recursos ou serviços, e o cliente, que os solicita [Tanenbaum and Wetherall 2011]. Em jogos online, o servidor geralmente atua como a autoridade central, gerenciando o estado do jogo, validando as ações dos jogadores e sincronizando os eventos entre eles. Esta centralização é crucial para prevenir trapaças e garantir que todos os jogadores tenham uma visão consistente do mundo do jogo. O Dueling Protocol adota este modelo, onde o servidor Java detém toda a lógica de negócio e o estado das partidas.

2.2. Comunicação de Rede com Sockets

Sockets são a interface fundamental para a programação de rede, representando um ponto de extremidade em uma comunicação bidirecional entre dois programas na rede [Kurose and Ross 2016]. A API de sockets permite que um programa se conecte a outro, envie e receba dados. No Dueling Protocol, a biblioteca de sockets nativa do Java (`java.net.Socket` e `java.net.ServerSocket`) foi utilizada para estabelecer a comunicação entre o cliente e o servidor, fornecendo um controle de baixo nível sobre o fluxo de dados.

2.3. Protocolos TCP e UDP

Os protocolos TCP (Transmission Control Protocol) e UDP (User Datagram Protocol) são os dois principais protocolos da camada de transporte da Internet. O TCP é orientado à conexão e oferece entrega de dados confiável e ordenada. Em contrapartida, o UDP não possui garantia de entrega, o que o torna mais rápido e com menor sobrecarga [Kurose and Ross 2016]. No projeto, o TCP foi escolhido para as ações críticas do jogo (como jogar uma carta), onde a perda de um comando é inaceitável. O UDP, por sua vez, foi utilizado para o `PingServer`, um serviço secundário que permite medir a latência do cliente de forma rápida.

2.4. Multithreading em Java

Multithreading é a capacidade de um processo de executar múltiplas threads concorrentemente. Em um servidor, isso é essencial para atender a vários clientes simultaneamente

sem que um tenha que esperar o outro ser atendido [Goetz et al. 2006]. O Dueling Protocol implementa o modelo ”uma thread por cliente”, onde o `GameServer` cria uma nova thread para cada `ClientHandler`. Embora simples, este modelo é eficaz para um número moderado de conexões e isola o trabalho de cada cliente.

2.5. Serialização de Dados com JSON

JSON (JavaScript Object Notation) é um formato leve e legível por humanos para intercâmbio de dados. Sua simplicidade o tornou um padrão para a serialização de objetos e comunicação em APIs [Bray 2017]. No projeto, o JSON foi utilizado para a persistência dos dados dos jogadores no `PlayerRepositoryJson`, permitindo que os dados sejam salvos em disco de forma simples.

2.6. Virtualização com Contêineres (Docker)

Contêineres são uma forma de virtualização que permite empacotar uma aplicação e suas dependências em uma unidade isolada e portátil [Merkel 2014]. Docker foi fundamental para criar um ambiente de teste consistente. Utilizando o Docker Compose, foi possível orquestrar a execução de múltiplas instâncias de clientes e do servidor com um único comando, automatizando a suíte de testes.

3. Metodologia

Esta seção detalha a implementação técnica do Dueling Protocol.

3.1. Arquitetura

A arquitetura do sistema foi projetada para ser modular e escalável, como ilustrado na Figura 1.

- **GameServer**: Ponto de entrada que escuta por conexões e instancia um `ClientHandler` por cliente.
- **ClientHandler**: Gerencia a comunicação com um único cliente, fazendo o parsing dos comandos e delegando para o `GameFacade`.
- **GameFacade**: Orquestra a lógica de negócio, atuando como uma fachada para os serviços de matchmaking, loja e persistência.
- **Serviços**: Componentes como `ConcurrentMatchmakingService` e `StoreServiceImpl` implementam lógicas de negócio específicas.
- **Repositórios**: A camada de persistência, com `PlayerRepositoryJson` e `CardRepository`, abstrai o acesso aos dados.

3.2. Comunicação e API Remota

A comunicação utiliza uma API textual sobre TCP, com comandos no formato `COMANDO:ARG1:ARG2:...`. A Tabela 1 detalha os principais comandos da API.

O diagrama de sequência na Figura 2 ilustra o fluxo de mensagens para a operação de matchmaking.

3.3. Encapsulamento e Concorrência

O `ClientHandler` valida e faz o parsing dos comandos, rejeitando mensagens malformadas. Para a concorrência, o sistema utiliza estruturas de dados thread-safe, como `ConcurrentLinkedQueue` para a fila de matchmaking e `ConcurrentHashMap` para gerenciar clientes e jogos ativos, garantindo acesso seguro por múltiplas threads.

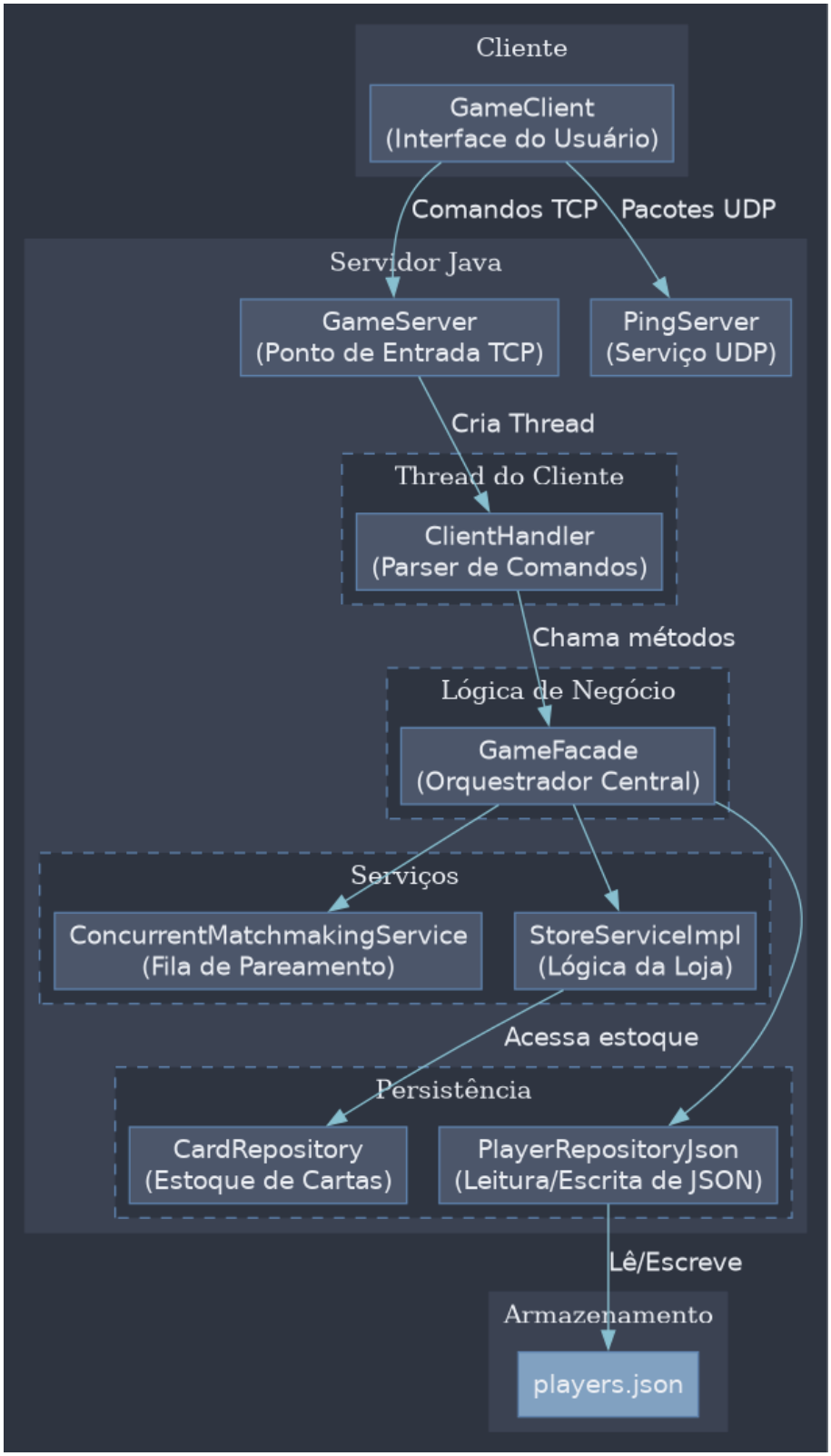

Figura 1. Diagrama de componentes da arquitetura do Dueling Protocol.

Tabela 1. Principais comandos da API remota.

Comando	Descrição
`CHARACTER_SETUP`	Configura a raça e classe do personagem.
`MATCHMAKING`	Adiciona o jogador à fila de pareamento.
`STORE:BUY`	Compra um pacote de cartas.
`GAME:PLAY_CARD`	Realiza uma jogada de carta na partida.
`UPGRADE`	Usa pontos para melhorar um atributo base.

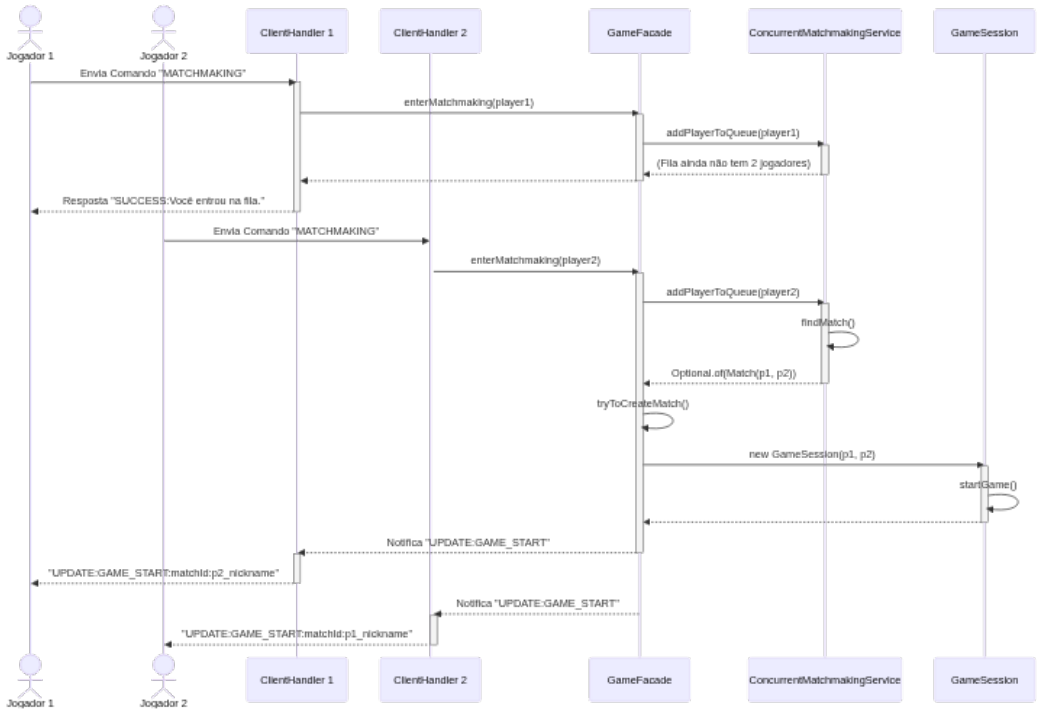

Figura 2. Diagrama de sequência para a criação de uma partida.

3.4. Latência, Partidas e Pacotes

- **Latência**: Um `PingServer` UDP ecoa pacotes, permitindo que o cliente calcule o tempo de ida e volta (RTT).
- **Partidas**: Uma condição de corrida no matchmaking foi corrigida: o `GameFacade` agora verifica se ambos os jogadores estão conectados antes de criar a partida. Se um desconectou, o outro é devolvido à fila.
- **Pacotes**: A compra de pacotes é atômica. O `StoreServiceImpl` usa um `ReentrantLock` com política ”fair”para enfileirar as solicitações de compra, garantindo que o acesso ao estoque de cartas do `CardRepository` seja justo e sequencial.

3.5. Testes e Emulação

Uma suíte de testes automatizada, orquestrada pelo script `run_all_tests.sh`, foi usada para validar a robustez do servidor. O Docker Compose cria um ambiente de emulação com um servidor e múltiplos clientes. Os cenários testam desconexões,

condições de corrida, comandos malformados e estresse com 10 clientes simultâneos por 30 segundos.

4. Resultados

A execução da suíte de testes validou a funcionalidade e a robustez do servidor. A Tabela 2 resume os resultados.

Tabela 2. Resumo dos resultados dos testes automatizados.

Teste	Objetivo	Resultado
Desconexão na Fila	Lidar com desconexão de cliente antes da partida.	Sucesso
Jogada Simultânea	Testar o comportamento com ações concorrentes.	Sucesso
Malicious Bot	Validar a robustez contra comandos malformados.	Sucesso
Teste de Estresse	Avaliar desempenho com 10 clientes simultâneos.	Sucesso

A análise dos logs foi fundamental. O teste de ”Desconexão na Fila”foi crucial para identificar e corrigir a condição de corrida no matchmaking. Os logs confirmaram que, após a correção, o servidor cancela a partida e devolve o jogador remanescente à fila.

O teste de estresse validou a eficácia do `ReentrantLock` no `StoreServiceImpl`, onde múltiplos clientes compraram pacotes simultaneamente sem inconsistências.

5. Conclusão

Este trabalho alcançou o objetivo de implementar um servidor de jogo multiplayer robusto e concorrente. A arquitetura, baseada no padrão Facade e em estruturas de dados thread-safe, provou ser eficaz. A principal contribuição foi o desenvolvimento de uma suíte de testes automatizada com Docker, que foi indispensável para identificar e corrigir bugs críticos de concorrência.

Como trabalhos futuros, sugere-se a expansão das mecânicas de jogo, como um sistema de ”Energia”para ações em tempo real e a implementação de um sistema de montagem de baralhos.

6. Referências

Referências

Bray, T. (2017). The JavaScript Object Notation (JSON) Data Interchange Format.

Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Holmes, D., and Lea, D. (2006). *Java Concurrency in Practice*. Addison-Wesley Professional.

Kurose, J. F. and Ross, K. W. (2016). *Redes de computadores e a internet: uma abordagem top-down*. Pearson, 7 edition.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. In *Linux Journal*, volume 2014.

Tanenbaum, A. S. and Wetherall, D. J. (2011). *Redes de computadores*. Pearson Prentice Hall, 5 edition.