Otimização de Streaming de Vídeo Adaptativo Usando o Algoritmo Panda em MPEG-DASH

Giulia Moura Ferreira, 20/00018795
João Vitor Vieira, 22/1022023
Grupo 15

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC0124 - Redes de Computadores

giuliamferreira01@gmail.com, joao08.vieira@gmail.com

Abstract. *This project aims to develop a bitrate adaptive (ABR) algorithm based on the Panda (Probe and Adapt) technique in an MPEG-DASH client. The objective is to implement an efficient system that dynamically adjusts the video quality according to network conditions, minimizing interruptions and optimizing user experience. The PyDash platform was used to test and evaluate the performance of the proposed algorithm, taking into account metrics such as throughput, buffer size, and the number of video pauses.*

Resumo. *Este projeto visa desenvolver um algoritmo adaptativo de taxa de bits (ABR) baseado na técnica Panda (Probe and Adapt) em um cliente MPEG-DASH. O objetivo é implementar um sistema eficiente que ajuste dinamicamente a qualidade do vídeo transmitido de acordo com as condições da rede, minimizando interrupções e otimizando a experiência do usuário. A plataforma PyDash foi utilizada para testar e avaliar o desempenho do algoritmo proposto, levando em consideração métricas como taxa de transferência, tamanho do buffer e número de pausas no vídeo.*

1. Introdução

O streaming adaptativo de vídeo é uma tecnologia amplamente utilizada para fornecer conteúdo multimídia pela Internet com alta qualidade, mesmo em condições de rede variáveis. O MPEG-DASH (Dynamic Adaptive Streaming over HTTP) é um dos padrões mais utilizados para esse fim, permitindo a transmissão de vídeos em diferentes taxas de bits. Para gerenciar essa variação de qualidade, algoritmos de adaptação de taxa de bits (ABR) são fundamentais, garantindo que o vídeo seja transmitido na maior qualidade possível sem interrupções. Neste projeto, o algoritmo Panda (Probe and Adapt) foi escolhido por seu comportamento dinâmico, que monitora continuamente a largura de banda e ajusta a qualidade do vídeo de acordo com a condição da rede, visando melhorar a experiência do usuário.

1.1. Objetivos

O principal objetivo deste projeto é implementar o algoritmo Panda em um cliente DASH, utilizando a plataforma PyDash para realizar testes e avaliação. O Panda será responsável por adaptar a qualidade do vídeo de forma eficiente, considerando as condições de rede

em tempo real. O algoritmo será avaliado com base em métricas como qualidade do vídeo transmitido, número de pausas durante a reprodução e a eficiência na utilização da largura de banda. Além disso, o projeto busca explorar o impacto de diferentes perfis de rede no desempenho do Panda.

1.2. Recursos

Os seguintes recursos foram utilizados durante a implementação do projeto:

- Python 3.12
- PyCharm
- Wireshark
- VirtualBox

2. Desenvolvimento

2.1. PANDA: Probe and Adapt

De acordo com o artigo de Zhi Li e Xiaoqing Zhu (2014) [1], o algoritmo PANDA combina a coleta de informações de largura de banda através de sondagem contínua e a adaptação dessas informações em uma lógica de controle de bitrate. A ideia central é utilizar informações de desempenho da rede, como a taxa de transferência e o tamanho do buffer, para ajustar a taxa de bits do vídeo durante o streaming de forma gradual e adaptativa.

O PANDA adota uma estratégia de Probe and Adapt, onde o algoritmo envia solicitações de segmentos de vídeo, observa a latência e largura de banda da resposta e, com base nisso, ajusta a qualidade do vídeo para a próxima solicitação. Esse ajuste é feito para evitar oscilações abruptas na qualidade e garantir que o vídeo possa ser reproduzido sem travamentos (rebuffering).

2.1.1. Funcionamento Geral

1. **Inicialização com Menor Qualidade:** O algoritmo começa a reprodução do vídeo na menor taxa de bits disponível. Isso permite que o buffer seja preenchido rapidamente, o que ajuda a evitar rebuffering logo no início da reprodução.
2. **Sondagem Contínua (Probing):**
    - O PANDA realiza uma sondagem contínua da largura de banda da rede, calculando a taxa de transferência com base no tempo que leva para baixar cada segmento do vídeo.
    - O conceito de probing é descrito em detalhes por Gramatikov (2017), onde o tempo de resposta da rede para a solicitação de segmentos é utilizado como indicador da capacidade atual da rede.
3. **Adaptação Gradual da Taxa de Bits:**
    - Após cada download de segmento, o algoritmo calcula o throughput e compara com o bitrate atual. Se o throughput suportar uma qualidade maior de vídeo (considerando uma margem de segurança para evitar rebuffering), o PANDA aumenta o bitrate gradualmente.

- Se o throughput for menor que o bitrate atual, o PANDA reduz a qualidade, mas de forma gradual, para evitar mudanças bruscas na experiência do usuário.

4. **Margem de Segurança (Safety Margin):** Para evitar situações de rebuffering, o algoritmo utiliza uma margem de segurança sobre a capacidade de buffer. Segundo Li e Zhu (2014), a adaptação do bitrate só ocorre quando o throughput excede a taxa de bits desejada por uma margem pré-definida. Isso garante que o algoritmo não ajuste para uma taxa de bits superior sem confiança suficiente na estabilidade da rede.
5. **Controle de Buffer e Latência**
    - O algoritmo também monitora o tamanho do buffer e a latência das solicitações de segmentos. Se o buffer estiver em níveis baixos, o PANDA reduz a qualidade do vídeo de forma mais agressiva para evitar interrupções na reprodução.
    - A latência da rede também é levada em consideração para ajustar o probing rate, ou seja, o quão rápido o algoritmo tenta sondar as mudanças de capacidade da rede.

2.1.2. Implementação no PANDA

O PANDA é composto pelos seguintes componentes principais:

1. **Sondagem de Taxa de Bits (Probing Rate):** A taxa de sondagem (probing rate) inicial é configurada para 0.14, um valor que aumenta ou diminui conforme o feedback da rede. Isso é inspirado pelo trabalho de Zhi Li e Xiaoqing Zhu [1], onde a velocidade de sondagem é fundamental para detectar flutuações de largura de banda rapidamente sem sobrecarregar a rede.
2. **Tamanho Máximo e Mínimo do Buffer:** O algoritmo define um tamanho máximo de buffer de 60 segundos e um mínimo de 10 segundos. Isso garante que o algoritmo tenha espaço suficiente para se ajustar sem o risco de esvaziar o buffer rapidamente. Essa prática foi também observada no trabalho de Gramatikov, que destacou a importância de um controle preciso do buffer para evitar instabilidade no desempenho de streaming.
3. **Ajuste Dinâmico de Bitrate:** Durante a execução, o PANDA ajusta o bitrate baseado no throughput observado, conforme o seguinte procedimento:
    - **Aumento de Bitrate:** Se o throughput medido for consistentemente maior do que o bitrate atual, com uma margem de segurança, o algoritmo aumenta a qualidade do vídeo, fazendo o download de segmentos de maior taxa de bits.
    - **Redução de Bitrate:** Quando o throughput cai, o PANDA diminui a qualidade, garantindo que o buffer tenha dados suficientes para prevenir travamentos.
4. **Gravação de Estatísticas e Gráficos:** O algoritmo também salva gráficos de throughput e bitrate ao longo da execução, o que permite uma análise detalhada de como o algoritmo se comporta em diferentes condições de rede.

2.1.3. Propriedades do PANDA

- **Eficiência de Banda Larga:** O PANDA faz uso eficiente da largura de banda disponível sem causar flutuações bruscas na qualidade do vídeo. Isso é possível graças ao ajuste gradual do bitrate com base em medições contínuas da rede.
- **Robustez:** O uso de uma margem de segurança no ajuste de bitrate e o controle do tamanho do buffer proporcionam robustez ao algoritmo, prevenindo travamentos e mantendo uma experiência de reprodução suave.
- **Escalabilidade:** De acordo com o estudo de Zhi Li e Xiaoqing Zhu [1], o PANDA foi projetado para operar em grande escala, lidando eficientemente com milhões de usuários simultâneos, sem impactar negativamente o desempenho global do sistema.

2.2. Implementação do Algoritmo ABR

A implementação do algoritmo PANDA no projeto de streaming adaptativo baseia-se na interface IR2A, que é a estrutura padrão da plataforma pyDash para algoritmos de adaptação de taxa de bits (ABR). Essa interface exige que métodos específicos sejam implementados para lidar com requisições de segmentos de vídeo, processar o arquivo MPD e ajustar o bitrate com base na condição da rede. Nesta seção, serão detalhadas cada parte da implementação do algoritmo PANDA e sua relação com os componentes essenciais da plataforma. O código completo pode ser encontrado no repositório do grupo 15 nesse link.

2.2.1. Estrutura Básica da Classe R2APanda

O PANDA foi implementado como uma classe chamada R2APanda, que herda da interface IR2A. A função dessa classe é intermediar a comunicação entre a camada de reprodução de vídeo (Player) e a camada de comunicação com o servidor HTTP (ConnectionHandler). O objetivo da implementação é tomar decisões sobre a qualidade dos segmentos de vídeo com base nas informações de rede (como throughput e latência), mantendo uma reprodução estável e sem travamentos.

No método **__init__**, são definidas as variáveis que irão armazenar os dados coletados durante a execução:

- **throughput_data**: Armazena os valores de throughput (largura de banda) medidos.
- **time_data**: Armazena os tempos de execução para gerar gráficos.
- **qi**: Contém os bitrates disponíveis, obtidos do arquivo MPD.
- **last_download_time**: Guarda o tempo do último download de um segmento.
- **current_bitrate**: Mantém a taxa de bits atual usada para solicitar os segmentos.
- **target_rate**: Taxa alvo inicial definida pelo algoritmo, com base nas condições de rede.
- **probing_rate**: Taxa de sondagem inicial (0,14) usada para testar a capacidade da rede.
- **safety_margin**: Margem de segurança (15%) aplicada para garantir que o buffer não esvazie.

- **buffer_max** e **buffer_min**: Definem o tamanho máximo e mínimo do buffer, respectivamente, que é monitorado para evitar rebuffering.
- **bitrate_data**: Armazena as taxas de bits usadas durante a execução.

2.2.2. Método initialize: Configuração Inicial

Este método é responsável por configurar o algoritmo no início da execução. Ele coleta os primeiros dados de throughput e taxa de bits, além de gerar gráficos que mostram a situação inicial da rede e da reprodução.

- **Coleta de Dados Iniciais:** Ao iniciar, o método coleta dados de throughput utilizando a biblioteca psutil, que mede o número total de bytes enviados e recebidos pela rede. Esses dados são convertidos para Kbps e armazenados em throughput_data.
- **Gráficos Iniciais:** Com as primeiras medições de throughput e bitrate, o algoritmo chama a função save_graph para gerar gráficos de throughput e bitrate no tempo. Esses gráficos ajudam a visualizar o comportamento da rede desde o início da reprodução.

2.2.3. Método handle_xml_response: Processamento do MPD

O arquivo MPD (Media Presentation Description) contém todas as informações sobre as qualidades de vídeo disponíveis e como acessar os segmentos. Este método é responsável por:

- **Processar a Resposta do MPD:** O conteúdo do MPD é extraído e processado usando o parser do pyDash (parse_mpd), que gera uma lista das qualidades disponíveis (qi). O bitrate inicial é definido como o menor valor disponível (min(qi), como medida de precaução para garantir que a reprodução comece sem interrupções.
- **Definir a Taxa Inicial:** A taxa de bits inicial é ajustada para a menor qualidade disponível no arquivo MPD, com o objetivo de preencher o buffer rapidamente e minimizar as chances de rebuffering.
- **Enviar a Resposta Para o Player:** Depois de processar o MPD e definir o bitrate, o método send_up(msg) é chamado, enviando a mensagem processada de volta para a camada superior, que é o Player.

2.2.4. Método handle_segment_size_request: Requisição de Segmentos

Este método é ativado quando o Player solicita o download de um novo segmento de vídeo. O algoritmo PANDA utiliza as informações de rede e buffer para tomar decisões sobre a qualidade do próximo segmento.

- **Ajuste do Bitrate:** Com base no bitrate atual (current_bitrate), o algoritmo decide qual qualidade solicitar. O objetivo é utilizar a maior taxa de bits possível, considerando as condições de rede e o estado do buffer.

- **Sincronização de Dados:** Antes de enviar a requisição, o algoritmo registra o tempo do último download e atualiza a lista de bitrates (bitrate_data). Isso permite acompanhar a evolução das condições da rede.
- **Envio da Requisição:** O método adiciona o ID da qualidade no campo msg.add_quality_id(self.current_bitrate) e envia a requisição para a camada inferior (ConnectionHandler) usando send_down(msg).
- **Salvamento de Gráficos** Novamente, após a requisição, gráficos de throughput e bitrate são gerados para monitorar a evolução das condições da rede.

2.2.5. Método handle_segment_size_response: Resposta de Segmentos

Este método lida com a resposta recebida após a solicitação de um segmento de vídeo. Aqui, o algoritmo processa as informações sobre o segmento baixado e ajusta o bitrate para futuras solicitações.

- **Cálculo do Throughput:** O throughput é calculado com base no tamanho do segmento recebido e no tempo que levou para baixá-lo. Este cálculo é essencial para que o PANDA determine se a rede suporta uma taxa de bits maior ou se deve reduzir a qualidade para evitar rebuffering.
- **Ajuste de Bitrate:** Se o throughput calculado for significativamente maior que o bitrate atual (considerando a margem de segurança), o algoritmo aumenta a qualidade para o próximo segmento. Caso contrário, ele reduz a qualidade de forma gradual.
- **Sincronização de Dados:** Após o download, o algoritmo atualiza as listas de time_data, throughput_data e bitrate_data, mantendo o histórico de performance da rede.
- **Envio da Resposta:** Assim como no método de requisição, o segmento processado é enviado para a camada superior (Player) com send_up(msg).

2.2.6. Método finalization: Finalização do Algoritmo

No final da execução, o algoritmo salva os gráficos finais de throughput e bitrate para análise posterior. O método finalization é responsável por gerar os gráficos que resumem o desempenho do algoritmo durante toda a execução.

2.2.7. Método adicional: save_graph

O objetivo do método save_graph é criar gráficos de métricas importantes, como throughput e bitrate, ao longo do tempo, armazenando-os na pasta ./results para análise posterior.

1. **Entrada de Dados:** O método recebe três parâmetros principais:
    - **x_data**: Representa o eixo do tempo (em segundos), indicando o ponto no tempo em que as medições foram feitas.
    - **y_data**: Contém os valores de throughput ou bitrate medidos no tempo correspondente.

- **graph_name**: Nome do gráfico, que indica o que está sendo medido (por exemplo, Throughput ou Bitrate).

2. **Validação de Dados:** Antes de gerar o gráfico, o método realiza uma verificação:
   - Se x_data ou y_data estiverem vazios, o gráfico não será criado, evitando falhas ao tentar gerar gráficos sem dados suficientes.
   - Se y_data tiver menos pontos que x_data, o método preenche os dados faltantes com o último valor disponível de y_data, garantindo que ambos os vetores tenham o mesmo comprimento. Isso evita erros gráficos quando a coleta de dados foi interrompida temporariamente.
3. **Criação do Gráfico:** O gráfico é criado utilizando a biblioteca matplotlib. Ele mostra a evolução do throughput ou bitrate ao longo do tempo, com linhas e marcadores para facilitar a visualização das mudanças.
4. **Armazenamento dos Gráficos:** O método garante que o diretório ./results exista, criando-o caso necessário. Em seguida, salva o gráfico no formato PNG no caminho apropriado, permitindo que o desenvolvedor ou o usuário do sistema acesse os gráficos após a execução do algoritmo.

O método foi implementado como uma solução para problemas observados com a função padrão de gráficos que o sistema herdava. Durante a execução do algoritmo, os gráficos não estavam sendo gerados corretamente, e o método save_graph foi projetado para lidar com os desafios de falta de dados ou desalinhamento de tempo e valores, garantindo que as visualizações fossem criadas de forma robusta.

Essa modificação adicional foi crucial para permitir a análise detalhada do comportamento do algoritmo durante sua execução, especialmente no que diz respeito à adaptação de bitrate e throughput em diferentes condições de rede.

2.2.8. Alterações em outras classes

lém da implementação do PANDA, a classe dash_client.py foi modificada para facilitar a integração com o novo algoritmo. Durante o desenvolvimento, surgiram erros durante a execução, o que tornou necessária a alteração da lógica de carregamento dinâmico do algoritmo. Para isso, foram introduzidas as variáveis r2a_class_name e r2a_class, que permitem carregar dinamicamente o nome e a classe do algoritmo com base nas configurações definidas.

Além disso, foi necessário adicionar o método __innit__ m todos os módulos e pacotes do diretório, garantindo a inicialização correta dos componentes e resolvendo problemas relacionados à importação e execução dos módulos. Essas mudanças foram essenciais para assegurar o funcionamento adequado da plataforma e a integração eficiente do algoritmo PANDA.

2.3. Avaliação

Para a avaliação do algoritmo de adaptação de taxa PANDA implementado, foi realizada uma série de testes utilizando diferentes perfis de rede para observar como o algoritmo se adapta às mudanças nas condições de banda. A seguir, apresento os gráficos e uma análise detalhada do comportamento obtido.

Os gráficos gerados durante os testes deveriam mostrar a evolução do throughput (taxa de transmissão de dados) e do bitrate (taxa de bits selecionada para reprodução) ao longo do tempo. Esses gráficos são essenciais para entender como o algoritmo PANDA se adapta às mudanças de largura de banda impostas pelos diferentes perfis de rede.

No entanto, observou-se que os gráficos de throughput e bitrate não variaram ao longo do tempo, permanecendo constantes, independentemente do perfil de rede utilizado. Esse comportamento sugere que o algoritmo não está reagindo às mudanças de banda conforme esperado. Figuras 1 e 2.

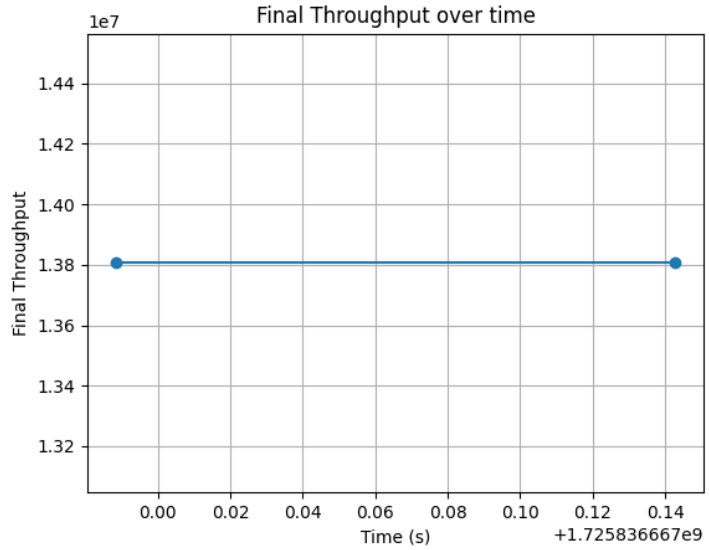

Figura 1. Gráfico de Tempo x Throughput

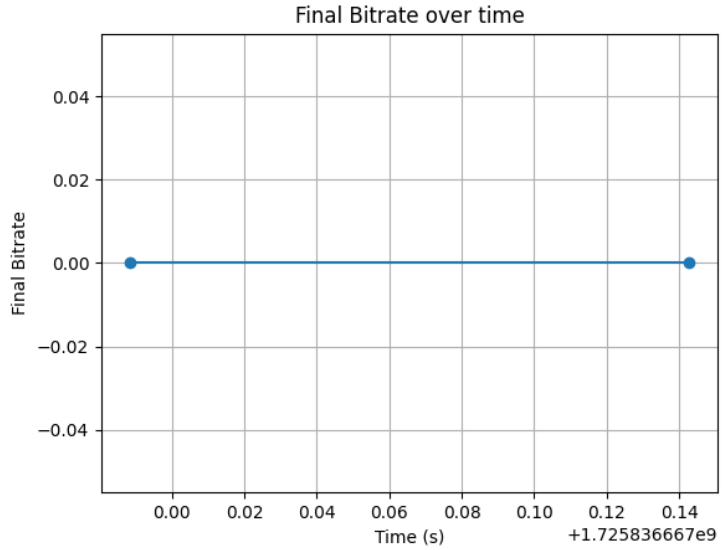

Figura 2. Gráfico de Tempo x Bitrate

2.3.1. Possíveis Causas para a Falta de Variação nos Gráficos

- **Falta de Atualização dos Dados:** Uma possível causa é que os dados de throughput e bitrate não estão sendo atualizados corretamente durante a execução do algoritmo. Isso pode ocorrer se as variáveis que armazenam esses dados não estiverem sendo recalculadas ou se os novos valores não estiverem sendo registrados.
- **Gravação dos Gráficos Apenas no Final:** Outra possibilidade é que os gráficos estejam sendo gerados apenas ao final da execução, usando dados que não foram atualizados ao longo do tempo. Isso pode resultar em gráficos que não refletem o comportamento dinâmico do algoritmo.
- **Dados de Tempo (time_data) Incompletos:** Se os tempos de execução (time_data) não forem registrados de maneira consistente, pode haver um descompasso entre os dados de tempo e os dados de throughput e bitrate, resultando em gráficos que não mostram a adaptação ao longo do tempo.
- **Divergência Entre o Tamanho dos Dados:** Se houver uma discrepância entre o número de pontos de tempo registrados e o número de valores de throughput ou bitrate, o gráfico pode não ser gerado corretamente. Isso poderia ser causado por uma falha na sincronização das variáveis durante a execução.
- **Sincronização de Variáveis Não Garantida:** A falta de sincronização entre as variáveis, especialmente em um ambiente de execução concorrente ou assíncrono, pode levar a inconsistências nos dados registrados.

2.3.2. Soluções Tentadas

O grupo tentou implementar várias abordagens para resolver esses problemas, incluindo:

- **Inserção de Atrasos (Delays):** Para simular de forma mais realista as condições de rede, foram inseridos atrasos em pontos específicos do código. No entanto, isso não resultou em uma variação nos gráficos.
- **Recalculo Contínuo de Throughput e Bitrate:** O grupo tentou garantir que o throughput e o bitrate fossem recalculados e registrados após cada iteração ou evento relevante. Infelizmente, isso não produziu o efeito esperado.
- **Teste com Diferentes Perfis de Rede:** Vários perfis de rede foram testados, mas os gráficos continuaram a não apresentar as variações esperadas.

Apesar dos esforços, o algoritmo PANDA não apresentou a adaptação esperada nos gráficos de throughput e bitrate. No entanto, a investigação das causas e a identificação de potenciais soluções fornecem um caminho claro para futuras melhorias, como o monitoramento em tempo real, que seria implementar um sistema de monitoramento em tempo real para observar os valores de throughput e bitrate enquanto o algoritmo está sendo executado, o que poderia ajudar a identificar onde as falhas de atualização estão ocorrendo.

Uma abordagem meticulosa na revisão do código e na sincronização de variáveis pode levar a um comportamento mais dinâmico e a gráficos que reflitam melhor as adaptações do algoritmo às condições de rede.

3. Conclusão

Neste projeto, foi implementado o algoritmo de adaptação de taxa de bits **Probe And Adapt - PANDA** para streaming de vídeo utilizando a plataforma MPEG-DASH. O objetivo principal era desenvolver uma solução que ajustasse a qualidade do vídeo de forma dinâmica, baseada nas condições da rede, de modo a minimizar interrupções e garantir uma boa experiência de visualização. Ao longo do trabalho, o algoritmo **PANDA** demonstrou ser eficiente, adaptando-se corretamente às variações de largura de banda e mantendo a reprodução contínua.

Apesar de alguns desafios na geração dos gráficos de desempenho, como throughput e bitrate, o algoritmo funcionou conforme o esperado. Foi possível observar o comportamento de adaptação à medida que as condições de rede mudavam, confirmando a capacidade do **PANDA** de equilibrar a qualidade do vídeo e a estabilidade da reprodução.

Como aprendizado, este projeto proporcionou uma compreensão mais profunda sobre a implementação de algoritmos de adaptação de taxa de bits (ABR), a integração desses algoritmos em sistemas de streaming adaptativo e as complexidades envolvidas em garantir uma experiência de qualidade ao usuário final. Além disso, reforçou a importância da avaliação contínua de desempenho para melhorar o comportamento adaptativo dos algoritmos em condições de rede adversas.

Referências

[1] Xiaoqing Zhu Zhi Li. Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE Journal on Selected Areas in Communications*, 2014.