UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Juliano Reis Fernandes Matos

Construção de um serviço conteinerizado de vídeo sob demanda baseado em DASH para experimentação e coleta de métricas de desempenho

Uberlândia, Brasil

2018

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Juliano Reis Fernandes Matos

Construção de um serviço conteinerizado de vídeo sob demanda baseado em DASH para experimentação e coleta de métricas de desempenho

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito parcial exigido à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Rafael Pasquini

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2018

Agradecimentos

Aos meus pais, por todo o apoio e incentivo durante minha trajetória nesta universidade e por sempre buscarem proporcionar a melhor educação a seus filhos.

Aos professores da Faculdade de Computação, por todos os ensinamentos compartilhados ao longo do curso. Em especial, ao meu orientador Rafael Pasquini, por fornecer a oportunidade de participar deste projeto e todo o suporte provido no decorrer do desenvolvimento do trabalho.

A todos os colegas da graduação, mestrado e doutorado que participaram do projeto e proveram auxílio e novas oportunidades de aprendizado.

Aos meus colegas de trabalho, pelo encorajamento para com minhas atividades acadêmicas.

Resumo

Esta pesquisa tem como objetivo o estudo de um ambiente de distribuição de vídeo sob demanda. À medida que o serviço implantado passa por diferentes condições durante seu funcionamento, este trabalho efetua a coleta de um conjunto de dados referente ao seu desempenho. Este conjunto de dados será adequado para investigações futuras de aprendizado de máquina. A metodologia se baseia na realização de experimentos controlados com a utilização de um serviço de transmissão de vídeos com adaptação dinâmica pelo método DASH, implantado em um ambiente conteinerizado, sendo acessado por um cliente de vídeo capaz de capturar métricas de qualidade de serviço. Ao mesmo tempo que o cliente está capturando as métricas, há um gerador de carga buscando afetar, de maneira controlada, o funcionamento da plataforma. Demonstramos que a plataforma foi capaz de realizar a transmissão apropriada dos vídeos configurados para o padrão DASH, ao mesmo tempo que o gerador de carga fez capaz a alteração entre os níveis de qualidade dos vídeos ao longo do experimento, resultando na diversificação dos valores contidos nos conjuntos de dados obtidos.

Palavras-chave: **adaptação dinâmica**, **DASH**, **Kubernetes**, **conteinerização**.

Abstract

The aim of this research is to study an on-demand video distribution environment. As the deployed service goes through distinct conditions during the time it is running, a platform component gathers data related to its overall performance. The collected data set will then be made suitable for future investigations where machine learning will be used. The methodology is based on carrying out controlled experiments using a video streaming service with support for dynamic adaptation by applying the DASH method, deployed in a conteinerized environment, accessed by a video client that is able to register a set of pre-defined metrics. As the video client captures the aforementioned metrics, a load generator tries to affect, in a controlled manner, the operation of the plaftorm. We show that the built platform was able to appropriately stream every DASH configured video available, while the load generator made the dynamic change in video quality possible during the experiment, resulting in a broader diversification of the data held within each newly obtained data set.

Key words: ***dynamic adaptation***, ***DASH***, ***Kubernetes***, ***containerization***

Lista de abreviaturas e siglas

DASH	Dynamic Adaptive Streaming over HTTP
ML	Machine Learning
MPD	Media Presentation Description
NFS	Network File System
PV	Persistent Volume
PVC	Persistent Volume Claim
QoS	Quality of Service
VM	Virtual Machine

Sumário

1 Introdução

A atual prática de hospedagem de serviços em servidores remotos, disponíveis por acesso na nuvem, implica em uma constante busca por arquiteturas de rede que mantenham alto desempenho em meio ao crescente tráfego de dados acessados simultaneamente por incontáveis usuários. Para que isso seja possível, é imprescindível que se tenha uma infraestrutura confiável, focada em retornar a melhor relação entre utilização de recursos e estabilidade da rede em *data centers*, seguindo metas que reduzam custos, mas, ao mesmo tempo, não sacrifiquem o poder de processamento necessário pelos procedimentos que gerenciam a disponibilização dos serviços ofertados [Greenberg et al. 2008a]. Com isso, são aplicados conceitos de computação em nuvem *(cloud computing)* que auxiliam na alocação de recursos virtuais para que serviços armazenados em um *data center* possam ser acessados a qualquer momento, necessitando apenas de acesso à Internet.

Buscar por inovações em *data centers* estabelecidos sobre uma estrutura de computação em nuvem *(cloud data centers)* significa identificar soluções para formulação de topologias de rede e infraestruturas de comunicação mais robustas [Verdi et al. 2010], levando em conta a necessidade de equipamentos (switches, roteadores, etc.) que forneçam estabilidade e, ao mesmo tempo, não causem impacto maior nos custos já elevados.

Conceitos como *scale-out* são aplicados de modo que hardware mais barato, também referido como comoditizado [Greenberg et al. 2008b], são instalados em grande quantidade. Essa abordagem é uma boa prática ao invés da utilização de equipamentos mais caros que, independente de apresentarem vantagens em diferentes aspectos, aumentam consideravelmente o custo de manutenção ao longo prazo. Além disso, muitas vezes esses equipamentos são compostos por software proprietário que limita as opções de gerenciamento de recursos da rede. O equipamento comoditizado, por sua vez, é mais barato e oferece adaptabilidade ao *data center* com planos de controle personalizáveis e facilidade na troca por novos equipamentos [Verdi et al. 2010].

Neste trabalho, um serviço de vídeo sob demanda será implantado em uma plataforma de computação em nuvem, utilizando contêineres disponíveis em nosso laboratório.

Com a ajuda do método DASH (*Dynamic Adaptive Streaming over HTTP*) [ISO/IEC 23009-1:2014], teremos um cliente capaz de acessar um arquivo de vídeo localizado em um servidor remoto e visualizar o conteúdo de vídeo em diferentes taxas de bits com resoluções pré-definidas em um documento de configuração denominado MPD (*Media Presentation Description*). Utilizando o formato MPD para codificar uma série de segmentos que formam componentes do conteúdo de mídia, é apenas necessário que o cliente DASH envie requisições HTTP de acordo com os tipos de segmentos disponibilizados. Dessa forma, o

cliente consegue realizar a adaptação da taxa de bits para uma transmissão contínua à medida que consegue evitar o esvaziamento do *buffer* de vídeo. [Chiariotti 2015]. Como o cliente DASH é capaz de fazer o controle da transmissão, não é necessário que o servidor que oferece o conteúdo de vídeo lide com a carga de gerenciamento de adaptação de fluxo.

O nível de qualidade da transmissão será dependente de fatores como a capacidade computacional dos servidores de processar as requisições à medida que os conteúdos disponíveis são servidos aos clientes conectados, a largura de banda disponível na conexão entre cliente e servidor ou o equipamento usado pelo usuário, que podem se tornar um obstáculo caso este também possua recursos computacionais insuficientes para a tarefa de transmissão do conteúdo. A adaptação do vídeo - neste trabalho realizada pelos mecanismos do padrão DASH - ao longo da transmissão será o ponto focal para o desenvolvimento de um mecanismo de extração de métricas referentes ao nível de qualidade do serviço de vídeo.

Em conclusão, o escopo deste trabalho é a construção do mecanismo de coleta de métricas relativas ao nível do serviço, conforme mencionado anteriormente. Essa abordagem visa suportar o estudo de técnicas eficazes de gerenciamento dos serviços. Como fruto deste trabalho, será possível investigar em trabalhos futuros o uso de aprendizado de máquina (ML - *Machine Learning*) como mecanismo auxiliar da gerência.

1.1 Objetivos

Temos como objetivo geral a construção um serviço conteinerizado de vídeo sob demanda baseado em DASH para experimentação em uma plataforma de computação em nuvem e coleta de métricas para possibilitar o treinamento de algoritmos de aprendizado de máquina através da construção de conjuntos de dados.

Objetivos Específicos:

- Implantar um conjunto de vídeos em diferentes resoluções obedecendo os critérios impostos pelo DASH;
- Alterar o cliente de vídeo para possibilitar a obtenção de métricas de qualidade de serviço (QoS - *Quality of Service* a partir da execução de vídeos;
- Estudar e construir uma plataforma de serviços baseada em conteinerização com a utilização do sistema de orquestração Kubernetes;
- Adaptar o gerador de carga para operar com a plataforma construída e auxiliar nos experimentos para possibilitar que a influência causada pela carga seja refletida diretamente nas informações retornadas pelo cliente de vídeo;

- Analisar os resultados obtidos através da execução do serviço de vídeos DASH em conjunto com o gerador de carga de modo a garantir que os resultados observados serão úteis para o aprendizado do ML.

1.2 Organização do texto

Este texto está organizado da seguinte forma. No Capítulo 2 falamos de trabalhos com temas relacionados ao que é apresentado ao longo do projeto, além de apresentarmos os principais conceitos e tecnologias envolvendo o desenvolvimento da plataforma. No Capítulo 3 descrevemos a configuração do conteúdo de vídeo a ser distribuído aos clientes e como as tecnologias escolhidas foram aplicadas no desenvolvimento da plataforma conteinerizada e do serviço de transmissão de vídeos. O Capítulo 4 aborda o processo de experimentação realizado no ambiente construído, destacando todo o conteúdo de vídeo disponível, os geradores de carga utilizados e a configuração das máquinas onde a plataforma foi hospedada e, por fim, apresenta os resultados obtidos após a execução dos experimentos através de uma análise das métricas de serviço obtidas. No Capítulo 5 é apresentada a conclusão obtida após avaliação dos resultados alcançados, os obstáculos enfrentados ao longo do trabalho realizado e os próximos passos para a continuação do projeto.

2 Fundamentação Teórica

2.1 Trabalhos Correlatos

O trabalho aqui realizado apresenta uma pesquisa fortemente relacionada ao que foi apresentado em [Pasquini and Staddler 2017], onde são feitas estimativas de métricas de QoS entre cliente e servidor utilizando uma rede OpenFlow. A partir de testes com um serviço de transmissão de vídeos sob demanda em funcionamento com clientes VLC e um sistema de armazenamento chave-valor *Voldemort*, os autores observaram que as estatísticas da rede em si são suficientes para realizar o aprendizado das métricas, podendo assim reduzir a quantidade de características necessárias de forma não prejudicial aos resultados. Foram aplicados dois métodos de aprendizado de máquina por regressão, *regression tree* e *random forest*, e um conjunto de características denominado X composto pelo agrupamento de servidores (*cluster*), conjunto de portas (*port*) e fluxo de dados (*flow*), cada um contendo seus respectivos subconjuntos, de uma rede OpenFlow virtualizada.

Os autores constataram que a estimativa de métricas a nível de serviço puderam ser feitas com uma média de erros abaixo de 10%, a partir de estatísticas providas pelo subconjunto de portas da rede, onde o método de *random forest* se mostrou mais eficaz nas estimativas, porém contendo um custo computacional consideravelmente maior. O principal fator que observaram foi que a redução do conjunto de características utilizadas, a partir de cada subconjunto existente, possibilitou aumentar a eficiência de aprendizado do algoritmo ao mesmo tempo que manteve um nível aceitável de precisão em relação ao conjunto X.

Nosso trabalho busca avançar nos conceitos estudados por estes autores à medida que são utilizadas novas tecnologias para complementar o ambiente construído que, aqui, se aproxima mais de um cenário onde os servidores têm maior chance de esgotar seus recursos computacionais. Utilizando técnicas mais modernas de transmissão de vídeos, vídeos com apenas um nível de qualidade são descartados com a aplicação de um método dinâmico de transmissão, além do uso de conteinerização para a exposição do serviço.

Os autores em [Bodík et al. 2009] discutem como modelos de ML podem ajudar a extrair estatísticas para encontrar políticas de controle automáticas otimizadas para melhor performance de serviços providos em nuvem, prática pouco utilizada em *data centers* por dúvidas quanto a adaptação destes modelos em exemplos reais.

Em [Qazi et al. 2013] é apresentado o Atlas, um *framework* que utiliza ML para classificação de aplicações Android com princípios em escalabilidade e precisão a partir do tráfego em uma rede OpenFlow. O trabalho desenvolvido possibilitou a incorporação de

reconhecimento de aplicações em camadas mais altas da rede a partir da automatização da coleta de dados para o processo de classificação na detecção de aplicações.

[Giotis et al. 2014] utilizam uma rede OpenFlow com adição de monitoramento de dados sFlow para realizar detecção de anomalias em arquiteturas SDN. É trabalhado em cima das funcionalidades existentes do protocolo OpenFlow, tendo em vista que ele é capaz de detectar certos fluxos anomalísticos, como *Distributed Denial of Service* (DDoS), mas pode causar sobrecarga do plano de controle durante o processamento das estatísticas desses fluxos, buscando uma implementação voltada à escalabilidade através da modularização das funcionalidades da rede OpenFlow e alternando a coleta de dados dos fluxos para o protocolo sFlow, eliminando a necessidade de utilizar o processamento de fluxos nativo da rede.

[Khan et al. 2009] fazem utilidade do *Mean Opinion Score* (MOS) para medir a qualidade de reprodução dos vídeos ofertados, utilizando parâmetros relacionados ao QoS para definir como a qualidade é afetada, classificando o conteúdo geral ofertado através do método de análise de *clusters* e aplicando *Principal Component Analysis* (PCA) para verificar a influência dos parâmetros QoS sobre a qualidade de vídeo em cada classificação criada. Dessa forma, os autores encontram um padrão de ajuste da taxa de bits ligado ao tipo de conteúdo do vídeo através do MOS.

Procuramos desenvolver um serviço de transmissão de vídeos com DASH para que cenários de experimentação semelhantes àqueles listados nestes trabalhos relacionados pudessem ser reproduzidos. O serviço de vídeo baseado em DASH é naturalmente mais complexo, pois o artifício de transição de resolução dos vídeos torna o serviço mutável, certamente desafiando os algoritmos de aprendizado de máquina.

2.2 Protocolo DASH

Dynamic Adaptive Streaming over HTTP é um protocolo de transmissão adaptativa de conteúdo de mídia ao-vivo ou sob-demanda, sob publicação ISO/IEC 23009-1, com a premissa de manter a reprodução de vídeos ininterrupta ao possibilitar a transição entre segmentos de vídeo de diferentes qualidades de visualização enviados através de requisições HTTP. O protocolo foi desenvolvido pela *Moving Picture Expert Group* [MPEG] com o intuito de servir como um padrão sólido para transmissão de vídeos em rede.

Diferentes implementações são usadas nos protocolos existentes para realizar esta tarefa. DASH é *codec-agnostic*, ou seja, implementado de modo a aceitar qualquer tipo de codificação de vídeo, como o H.264, utilizado no conteúdo deste trabalho.

O funcionamento de protocolos de adaptação dinâmica de taxa de bits se dá pela detecção da capacidade do sistema do usuário de consumir o conteúdo, como a largura

de banda e capacidade de processamento da CPU, ao mesmo tempo em que o vídeo está sendo disponibilizado ao usuário. Com essas informações, o cliente de reprodução de vídeo continuamente envia requisições ao servidor que contém o conteúdo de mídia para recuperar novos segmentos de vídeo.

Este meio de transmissão traz vantagens como a facilidade de envio de pacotes HTTP, utilização de cache para o conteúdo transmitido e capacidade de manter a lógica de adaptação do conteúdo inteiramente no cliente, diminuindo a complexidade do trabalho do servidor de disponibilizar o conteúdo e eliminando sua necessidade de manter o estado da sessão para cada cliente conectado, facilitando a capacidade de escalabilidade do servidor.

Para adaptar o conteúdo, DASH faz uso do arquivo de manifesto MPD e dos respectivos segmentos de vídeo que serão reproduzidos pelo cliente. O MPD é um arquivo no formato XML que possui todas as informações pertinentes aos arquivos de segmentos de vídeo necessárias ao cliente para que a adaptação dinâmica do conteúdo ocorra. O manifesto é composto pelos seguintes elementos:

- *Period*: descrição de alto nível do conteúdo incorporado, com duração total de vídeo e tempo de início do conteúdo;
- *Adaptation Set*: possui o agrupamento dos segmentos do conteúdo de áudio ou vídeo a ser transmitido. O manifesto pode conter todo o conteúdo dentro de um mesmo *adaptation set*, ou dividir cada arquivo de segmentos entre diferentes *adaptation sets*;
- *Representation*: divide o conteúdo incluído no manifesto entre suas diferentes configurações, identificando dados como a resolução do vídeo, largura de banda alvo para transmissão do segmento, taxa de quadros e codificação do segmento. *Representations* podem ser quebradas em *SubRepresentations*, estas individualmente complementam informações sobre um dos conteúdos incluídos numa *Representations*;
- *BaseURL*: identifica a localização do conteúdo a ser acessado;
- *SegmentBase*: possui o alcance de indexação de segmentos para permitir a obtenção do ponto de acesso do conteúdo contido num segmento. A partir do ponto de acesso o cliente identifica em que parte do segmento a decodificação do conteúdo pode iniciar, utilizando os dados a partir daquele ponto para continuar a transmissão.

Os segmentos de vídeo incluídos no MPD fazem parte de um ou mais arquivos de vídeo. A partir do vídeo original são gerados novos arquivos de vídeo com diferentes configurações, entre elas resolução, taxa de quadros, tamanho de buffer e codificação de

vídeo, definindo a variação de qualidade do conteúdo capaz de ser reproduzida pelo cliente. Cada arquivo de vídeo têm seu conteúdo segmentado em pedaços com uma duração em segundos estabelecida na geração dos segmentos, que serão posteriormente disponibilizados através de um único arquivo de vídeo contendo todos os segmentos ou vários arquivos de vídeo gerados para cada segmento processado.

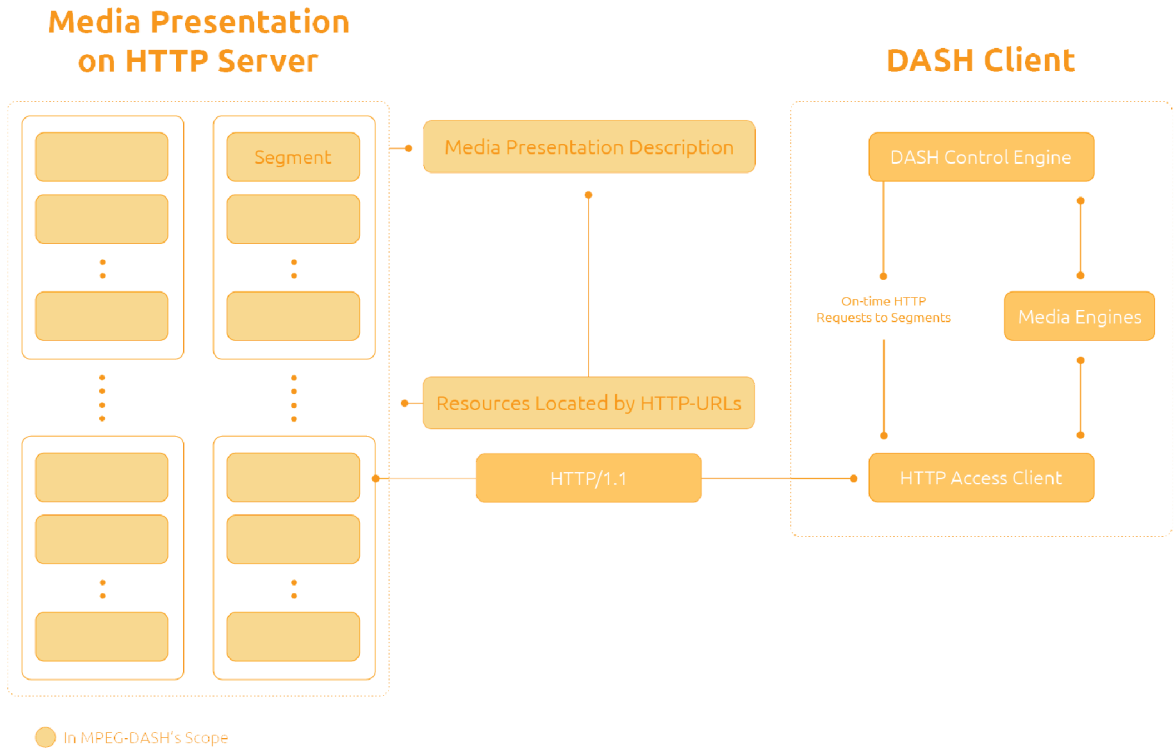

Figura 1 – Diagrama de representação do fluxo de funcionamento do protocolo DASH. Fonte: https://www.encoding.com/mpeg-dash/ [MPEG-DASH Workflow]

O fluxo de funcionamento do DASH é demonstrado na Figura 1, que apresenta a integração entre um cliente de vídeo compatível ao protocolo e um servidor que realiza a distribuição de mídia com itens capazes de alternância de qualidade, de modo que o manifesto DASH se torna o arquivo primário a ser buscado pelo cliente. Como é exemplificado na imagem, DASH tem uma premissa simplificada tanto para utilização em aplicações quanto para ser instaurado em serviços de transmissão, como é utilizado em grandes plataformas como Netflix e YouTube no formato HTML5, justificando sua pretensão de se tornar um padrão em meio às alternativas existentes.

Com a aquisição do manifesto, o cliente DASH identifica o conteúdo e os dados que compõem cada segmento podendo discernir qual a melhor representação inicial a ser requisitada através de requisições HTTP GET, dependendo da atual situação do sistema do usuário que será monitorado ao longo da reprodução do vídeo. Novos segmentos

são requisitados pelo cliente à medida que o conteúdo recuperado é adaptado para a visualização contínua do arquivo de mídia.

2.3 Apache Web Server

Desenvolvido e mantido pela Apache Software Foundation, o Apache HTTP Server, ou HTTPD, é um servidor web de código aberto disponibilizado em múltiplas plataformas compatível com HTTP/1.1 e licenciado sob a Apache License 2.0.

Suporta uma multitude de módulos que complementam as funcionalidades principais do servidor, entre elas:

- Suporte a linguagens de programação para desenvolvimento no servidor;
- Esquemas de autenticação;
- Balanceamento de carga entre servidores por *proxy* reverso;
- Módulos de segurança para camadas de comunicação e transporte;
- Deteção e prevenção de intrusão em aplicações Web;
- Arquivos de log configuráveis;
- Compressão de páginas Web por HTTP;
- *Virtual Hosts* que permitem Apache distinguir múltiplos endereços de diferentes páginas Web dentro de uma única instalação;
- Configuração do índice de diretório utilizado por cada endereço acessado.

Além dos módulos já existentes, Apache disponibiliza suporte para escrita de novos módulos pelo usuário possibilitando mais diversificação no uso da ferramenta.

Servidores Apache são configurados a partir de *MultiProcessing Modules* (MPMs) para atender a demandas específicas da infraestrutura em que se encontra. MPMs possibilitam a diversificação no modo como o servidor Apache atenderá a novas requisições, podendo ser executado como processos (*prefork*), em modo híbrido (*MPM worker*) ou modo de eventos-híbrido (*MPM event*).

2.4 Kubernetes

O sistema de orquestração de contêineres Kubernetes [K8s] é uma plataforma de código aberto para automatização de implantação, dimensionamento e gerenciamento de

serviços conteinerizados executados em um aglomerado de máquinas. O projeto é atualmente mantido pela *Cloud Native Computing Foundation* [CNCF].

Kubernetes baseia suas funcionalidades em um ambiente centrado em contêineres, integrando serviços de armazenamento, networking, segurança, entre outros, oferecendo portabilidade e facilidade de gerenciamento de serviços criados e mantidos através de múltiplos contêineres, conforme esquema apresentado na Figura 2.

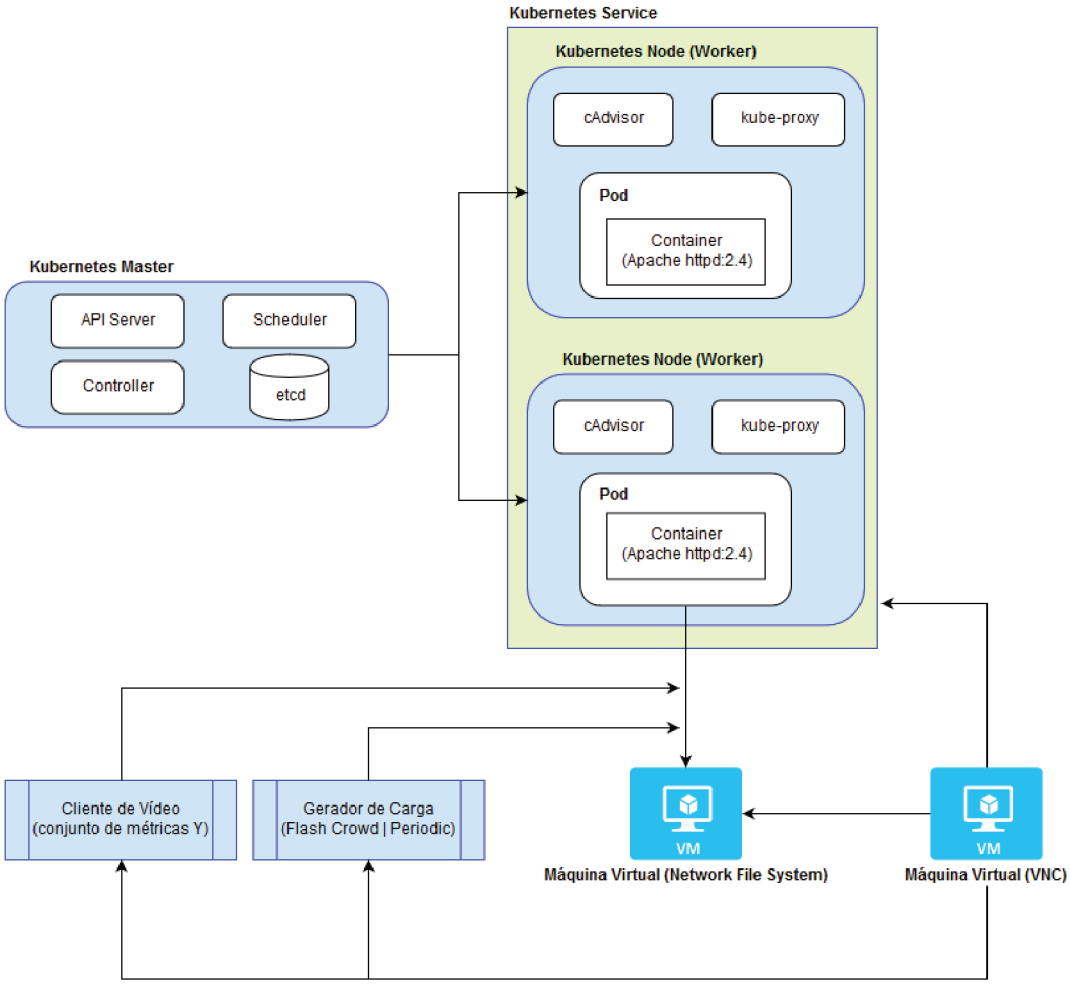

Figura 2 – Diagrama de representação da rede Kubernetes.

A plataforma oferece uma base para a infraestrutura que pode ser criada para a administração de contêineres, definida pelos componentes:

- *Master*: máquina encarregada de supervisionar e atribuir tarefas aos *Nodes*, mantendo um número de processos capazes de conservar o estado de um aglomerado de máquinas que incluem os contêineres a serem executados. Esta máquina também é executada sob um *Node* designado como *Master Node*;
- *Nodes*: máquinas que realizam tarefas requisitadas pela máquina *Master*, contendo

Pods para agrupamento de contêineres. Uma vez configurados e ativos, a interação direta com um *Node* é mínima em comparação com os outros componentes;

- *Pods*: contidos em um *Node*, são formados por um ou mais contêineres executados em um contexto no qual compartilham recursos de rede e armazenamento. *Pods* contêm um endereço IP único dentro do aglomerado e são capazes de distribuir portas para as aplicações executadas sem que hajam conflitos. Possibilitam a criação de volumes persistentes de dados, podendo ser de diretórios locais ou pela rede (e.g., NFS, armazenamento em nuvem, etc.)
- *Controllers*: controladores realizam tarefas diversas a partir dos Pods, como replicação de *Pods* no aglomerado e geração de *Pods* substitutos no caso de falha na execução de um Pod.
- *Services*: serviços são abstrações geradas através da exposição do conjunto de funcionalidades providas pelos contêineres de um agrupamento de Pods. Para um serviço pode ser atribuído:
  - o endereço IP do aglomerado de seu *Pod* restringindo-o apenas para acessos internos (*ClusterIP*);
  - uma porta estática e utilizar o endereço IP do *Node* podendo ser acessado externamente (*NodePort*);
  - balanceamento de carga, do tipo *round-robin*, se exposto através de um serviço de armazenamento em nuvem e recebendo um endereço IP externo (*LoadBalancer*).

2.4.1 Contêineres

O contêiner é a abstração do serviço executável, formado a partir de uma imagem autônoma do software contendo o necessário para operá-lo de forma isolada ao sistema: código, dependências, bibliotecas e configurações empacotados e disponibilizados em uma única imagem.

Considerados mais fáceis de usar e manter, contêineres podem ser executados em qualquer distribuição de sistema operacional com suporte a ferramentas de geração, aquisição e gerenciamento de contêineres (e.g., Docker).

A aquisição de imagens se torna uma tarefa simplificada, são leves e podem ser incorporadas rapidamente por conta da portabilidade oferecida. Em sua virtualização, são capazes de compartilhar o kernel do sistema operacional onde são executados e utilizam menos recursos computacionais.

Comparados a Máquinas Virtuais (VM - Virtual Machine), contêineres são mais eficientes à medida que replicam apenas as partes necessárias do sistema operacional,

utilizam sistemas de arquivos separados e executam como processos isolados, permitindo a ação simultânea de múltiplos contêineres em uma única máquina com menor utilização de recursos, diferentemente de VMs que necessitam virtualizar uma cópia do sistema operacional e os componentes de hardware que fazem capazes sua execução.

2.4.2 Imagens

Imagens são modelos contendo instruções de criação de contêineres separadas por camadas, cada camada sendo uma linha de instrução, onde é transmitida uma coleção específica de comandos que ditam o formato a ser tomado pelo contêiner. Podendo ser compostas por outras imagens, é possível usar uma imagem já existente e complementá-la com novos componentes. Assim, é viável adaptar e gerar imagens únicas para situações específicas de desenvolvimento.

A geração de imagens é feita por um *Dockerfile*, arquivo que conterá a cada linha uma instrução de processamento da imagem. Uma vez que a imagem é processada, nas alterações subsequentes da imagem apenas serão processadas as modificações incluídas, dessa forma diminui o tempo necessário para modificar uma imagem já existente e não causa grandes mudanças em seu tamanho, mantendo a portabilidade já oferecida.

2.4.3 Docker

Para realizar a importação de contêineres para o Kubernetes é possível utilizar a integração com Docker [Docker], plataforma de desenvolvimento e empacotamento de aplicações com suporte a criação e conteinerização de imagens. Sua interface permite compilar *Dockerfiles* criados pelo usuário, gerar contêineres locais, estabelecer conexão com o servidor da plataforma e enviá-los para um repositório pessoal de onde podem, futuramente, ser recuperados e aplicados tanto como um serviço *stand-alone*, como ser incorporados em novos contêineres.

O Kubernetes é capaz de utilizar a estrutura de aquisição de imagens a partir dos repositórios do *Docker Hub* [Docker Hub] durante a implantação de Pods, adquirindo os contêineres e integrando-os aos *Pods* da rede para a futura geração de novos serviços. Este passo facilita o processo de criação do serviço a ser exposto pelo Kubernetes, pois elimina qualquer trabalho extra necessário para complementar a aplicação desejada.

2.5 Aprendizado de Máquina

Campo de estudo na ciência da computação que utiliza dados estatísticos para aperfeiçoamento do funcionamento de uma tarefa computacional. Pela natureza dos algoritmos utilizados, ML tem fortes ligações com otimizações matemáticas em computação

que auxiliam com tarefas extensivas que poderiam se tornar difíceis ou prolongadas de se realizar com métodos mais comuns.

Conforme o conjunto de dados é moldado e distribuído ao algoritmo de ML, duas principais categorias determinam o tipo de aprendizado aplicado ao sistema: Aprendizado Supervisionado e Não Supervisionado.

No Aprendizado Supervisionado, ambos os conjuntos de entrada e saída, que neste cenário são identificados e reconhecidos pelo algoritmo, são previamente definidos. Tendo como objetivo analisar o mapeamento dos pares de dados de treino existentes, a função aplicada para apuração dos resultados é capaz de gerar métricas para novos conjuntos de dados introduzidos ao sistema, estes que serão reconhecidos e rotulados pelo algoritmo de acordo com o conjunto de treino processado e agrupá-las com as métricas já calculadas.

Em contrapartida, o conjunto de dados provido no Aprendizado Não Supervisionado não é atribuído com uma identificação ao que os dados inferem, originando a tarefa de determinar como os dados serão estruturados e processados pelo algoritmo. O algoritmo busca formas de agrupar os dados, derivando novos agrupamentos de acordo com relações de semelhança existentes entre dados dentro do conjunto.

Como base de cálculo para os métodos de treino, têm-se as seguintes funções como referência:

- Função de Hipótese: designa características como parâmetros para a função com finalidade de calcular estimativas que correspondam aos valores do conjunto de saída, formando um modelo de representação para as predições do treino.
- Função de Custo: utilizada para medir a acurácia da função de hipótese, a partir das próprias predições observadas. Tem como objetivo minimizar os parâmetros utilizados no treinamento do algoritmo, inferindo na diminuição da margem de erro das predições quando comparadas com os valores do conjunto de saída.

Os métodos de treino podem ser afetados tanto pelo tamanho do conjunto de dados, como pelo esquema de separação dos diferentes conjuntos a serem passados para o algoritmo. É necessário analisar quantidade de dados com a qual é trabalhada e considerar a possibilidade de criar uma curva de aprendizado para encolher o conjunto ou se é necessário acrescentar mais dados ao conjunto para que o processamento do algoritmo não seja prejudicado. A seleção de características úteis também se torna um passo importante, não serão todas as características coletadas que serão necessárias no aprendizado do algoritmo.

A escolha do algoritmo é outra parte do processo que deve ser considerada cuidadosamente. As características escolhidas, as predições a serem observadas, as categorias

de dados a serem formadas e o comportamento do aprendizado com relação aos tipos de exemplos providos ao algoritmo se tornam fatores de decisão para encontrar o melhor método de aprendizagem.

Durante as etapas de treinamento, os dados coletados podem ser separados entre conjuntos de treino, validação e teste:

- Conjunto de treino: agrupamento de exemplos aplicado no treinamento do algoritmo de ML.
- Conjunto de validação: amostra de dados imparcial utilizada para avaliar o modelo de treino adaptado para o algoritmo. O algoritmo não faz treinamento destes dados, apenas possibilita que ajustes de parâmetros sejam feitos no modelo.
- Conjunto de teste: conjunto de dados que proporciona uma avaliação definitiva do modelo de treino completado, utilizado para avaliar a performance de classificação das predições.

Após o modelo de treino ser formado e testado com as características escolhidas, é preciso diagnosticar os problemas existentes quanto ao ajuste do modelo. Caso o modelo escolhido não consiga criar uma representação adequada das características em relação aos valores de saída, ou seja, o resultado das predições não é capaz de demonstrar um modelo que faça uma separação satisfatória da coleção de dados tem-se o subajuste (*underfitting*). Por outro lado, se o modelo de predição demontra alta precisão quanto ao conjunto de dados testados, terá dificuldade de representar novos dados que possam ser passados ao algoritmo após o treinamento, causando sobreajuste (*overfitting*) do modelo. Estes problemas devem ser identificados e tratados de acordo, ajustando a regularização aplicada no modelo de predição e selecionando um número, maior ou menor dependendo da falha, de características que contribua com a melhoria do treinamento.

Neste trabalho, o conceito de ML é importante para reforçar a finalidade da plataforma em projetos futuros, uma vez que um dos objetivos do serviço de transmissão de vídeos é agrupar os valores coletados pelo cliente de vídeo para possibilitar a formação de conjuntos de dados de saída coerentes, à medida que os resultados gradualmente obtidos são analisados para confirmar que a plataforma se mostrará capaz de gerar novos conjuntos de dados em cenários diversificados, dependendo de quaisquer configurações que possam ser alteradas e que afetem de alguma forma os resultados adquiridos.

2.5.1 Regressão

Análises de regressão consistem em calcular predições partindo de um conjunto de valores de entrada, que formam as características do conjunto de treino, com relação a

um conjunto de valores de saída, este capaz de categorizar os tipos de resultado de treino esperados, formado por valores contínuos e quantitativos mas também, em diferentes casos, podendo representar um tipo de resultado discreto. Os dados do conjunto de entrada são aplicados sobre uma função contínua para realizar estimativas, uma aplicação comum sendo o cálculo da raiz do erro médio quadrático (RMSE - *Root Mean Squared Error*) para minimização dos parâmetros de treino.

Os métodos de cálculo de regressão partem de um fundamento formado por quatro elementos: variável dependente, variável independente, constante e erro. A variável dependente é composta por um valor do conjunto de dados de saída, ou seja, uma representação da predição a ser feita. A constante determina o índice que intercepta os valores dependentes da predição. A variável independente é um valor pertencente ao conjunto de entrada e é aplicada sobre uma inclinação, como uma linha de gradiente, para indicar o quanto a variável dependente pode mudar à medida que a variável dependente é alterada. Por fim, o erro demonstra o quão distante o algoritmo está de um resultado plausível. A capacidade de diminuição do erro é uma tarefa importante que pode ajudar a determinar se o método de predição aplicado sobre as coleções de dados obtidas é apropriado.

As estimativas retornadas como resultado expressam a relação entre valores independentes de entrada e os correspondentes valores de saída dependentes sendo possível compará-las ao conjunto de valores verdade já existentes, formando novas predições. Como os conjuntos de valores são bem definidos no Aprendizado Supervisionado, têm-se uma ideia melhor do tipo de valor que será retornado pelas predições.

2.5.2 Classificação

Treinamento por classificação envolve utilizar métodos de aprendizado para definir como as estimativas conseguem definir agrupamentos de dados do conjunto de treino e identificá-los apropriadamente. O conjunto de valores de saída é formado por valores discretos ou reais que, assim como em regressão, são classes ou categorias nas quais as predições alcançadas serão mapeadas pelo algoritmo classificador.

Na verificação do grau de aprendizado do algoritmo classificador, a quantidade de características classificadas corretamente demonstra a eficácia do método de predição para o conjunto de treino parametrizado no algoritmo. A partir das predições classificadas corretamente, é capaz de traçar fronteiras de decisão mais bem definidas.

O modelo classificatório usa quatro tipos de medidas para avaliação: verdadeiro positivo, falso positivo, verdadeiro negativo, falso negativo. A partir da quantidade de dados predita em cada medida, torna-se possível calcular a taxa de erro do modelo.

Utilizando as medidas de erro coletadas, pode-se avaliar a performance do algoritmo com a ajuda de diferentes métricas:

- Acurácia: métrica calculada a partir da divisão entre a proporção de predições corretas e o número total de dados utilizados na predição.
- Precisão: proporção de predições classificadas corretamente como verdadeiros positivos. Desconsidera qualquer exemplo negativo para realizar o cálculo.
- Recordação/Sensibilidade (*Recall*): proporção total de exemplos de treino classificados como verdadeiros positivos. Diferente da precisão, considera todos os exemplos positivos no cálculo (verdadeiro positivo, falso negativo).
- Especificidade: proporção de exemplos de treino classificados como negativos. Considera todos os exemplos negativos no cálculo (verdadeiro negativo, falso positivo).
- *F1 Score*: Média harmônica calculada a partir dos valores de precisão e recordação do modelo classificatório.

3 Desenvolvimento do Projeto

Nas etapas iniciais de teste, o trabalho consiste na coleta de métricas pré-definidas através dos atributos dos vídeos disponíveis em diferentes configurações para o diagnóstico da plataforma. A análise das métricas adquiridas deverá possibilitar a observação do comportamento apresentado pela plataforma durante a execução do serviço, envolvendo a performance dos vídeos configurados com o protocolo DASH sob o estresse causado pelo gerador de carga nos componentes do Kubernetes.

A princípio, a infraestrutura do Kubernetes é construída com a visão de como os servidores Apache são distribuídos entre cada servidor físico disponível. Com a implantação e exposição do serviço conteinerizado, nós selecionados na rede adquirem contêineres com a imagem de um servidor Apache HTTPD (*HyperText Transfer Protocol Daemon*) [Apache HTTPD], onde o diretório raiz de cada servidor contido em um contêiner é redirecionado para o diretório de uma VM específica configurada com um sistema de arquivos em rede (*NFS - Network File System*) designada com a tarefa de armazenar o conteúdo de mídia a ser recuperado pelos clientes de reprodução de vídeo.

O Kubernetes irá prover os meios de instanciamento e acesso entre os clientes e o serviço de distribuição de mídia, possibilitando a geração de réplicas dos componentes de rede que propiciam os ambientes de cada servidor. Com mais servidores em funcionamento, há a possibilidade da estrutura se beneficiar dos módulos de balanceamento de carga, ampliando as oportunidades de teste.

3.1 Infraestrutura Kubernetes

A infraestrutura é composta pelo conjunto de máquinas que formam a VM *Master* e seus respectivos nós. A VM *Master* (192.168.0.30) é o controlador da rede Kubernetes, onde as tarefas como instanciação de nós e implantação de contêineres em *Pods* dentro do aglomerado de componentes são executadas. No entanto, existe a política de não executar contêineres no componente *Master* por este se encarregar da administração dos componentes pertencentes a rede Kubernetes. Os nós da rede Kubernetes (*Nodes*) estão diretamente ligados à Master, recuperando imagens, criando contêineres e mantendo serviços ativos em seus *Pods* de acordo com a implantação definida para funcionar na estrutura. Para os propósitos deste trabalho, oito componentes formam os nós da rede que deverão trabalhar como pontos focais de transmissão dos dados de mídia para os clientes de vídeo.

Como a máquina *Master* provê uma inferface de controle para toda a rede Kubernetes, possui uma série de componentes destinados às tarefas de administração do

aglomerado. Inicialmente estes componentes podem ser executados em qualquer máquina interna da rede Kubernetes, mas, por questão de simplicidade e até mesmo melhor administração da rede criada, pode-se automatizar este passo com a execução de scripts numa mesma máquina, definindo-a assim como a Master.

- ***kube-apiserver***
  Componente que funciona como o *front-end* da infraestrutura Kubernetes, com a capacidade de expor a API Kubernetes quando executado.

- ***kube-scheduler***
  Pods gerados na rede Kubernetes pertencem a um nó e, dessa forma, este componente tem a tarefa de analisar *Pods* gerados e atribuí-los a algum nó levando em consideração fatores como a utilização coletiva de recursos, restrições existentes e carga de trabalho entre componentes da topologia.

- ***kube-controller***
  Este componente executa controladores, sendo estes laços de controle que observam o estado do aglomerado, utilizando essa informação para tentar mudar o estado atual da estrutura para outro estado desejado. Controladores são separados em:

  - *Node Controller*: Visualiza a situação dos nós, verificando quando algum nó para de funcionar.
  - *Replication Controller*: Mantém o número correto de *Pods* por nó na rede.
  - *Endpoints Controller*: Responsável por popular a API Endpoints da rede Kubernetes. Essa API é atualizada quando detecta alteração nos *Pods* de um serviço.
  - *Service Account, Token Controllers*: Responsável por criar contas padrões e fichas de acesso para novos *namespaces*.

- ***etcd***
  Reservatório chave-valor consistente e de alta disponibilidade para armazenamento e recuperação dos dados do aglomerado.

Cada *Pod* da rede tem acesso a um volume de armazenamento interno criado pela *Master* utilizado para acessar o conteúdo do serviço de transmissão de vídeos. Dois componentes de armazenamento são definidos para o volume: *Persistent Volume* (PV) e *Persistent Volume Claim* (PVC). O PV é um espaço de armazenamento de recursos utilizados por contêineres ou serviços, funciona como mais um recurso de volume de dados alocado no aglomerado administrado que contém um ciclo de vida separado daquele do Pod, enquanto que PVCs são usados para requisitar acesso a um PV, oferecendo acesso para consumir quaisquer recursos que o volume de dados referenciado pelo PV

proporciona. Com o PVC é possível assegurar configurações como o modo de acesso dos recursos do volume.

Para que os *Pods* sejam gerados e, consequentemente, a exposição dos serviços ocorra é executado um script de implantação através da Master. Com isso, será provocada uma mudança no estado atual da rede, inserindo novos recursos ou atualizando aqueles já existentes para atender a demanda necessária. O script de implantação definido na infraestrutura Kubernetes criada para este trabalho realiza três funções: gera contêineres executando imagens do servidor HTTPD, gera a instância de serviço para capacitar cada nó de possibilitar acesso ao conteúdo destinado aos clientes DASH e, por último, define o endereço de acesso aos *Endpoints* do serviço.

Com o *deployment* e a geração dos *Pods* da rede, têm-se um número de contêineres ativos onde cada contém um servidor Apache HTTPD montado para distribuir o conteúdo de vídeo DASH a ser transmitido para qualquer cliente que se conecte com o nó.

Os servidores HTTPD conteinerizados não possuem o conteúdo de vídeo em si. Ao invés disso, acessam um diretório compartilhado na máquina virtual do endereço 192.168.0.110 de onde todos os vídeos DASH serão recuperados em primeira instância a partir da implantação de um sistema de compartilhamento de arquivos por NFS entre os nós da rede. A ligação com a rede NFS entre cada novo servidor HTTPD e o diretório de armazenamento é feita através do PV existente que define o diretório do sistema de arquivos compartilhado no endereço 192.168.0.110 como a instância do volume de dados dos *Pods* da rede. Dessa forma, o diretório especificado por */home/cloud/videos* fica encarregado de compartilhar os manifestos e vídeos DASH com quaisquer servidores HTTPD provenientes de contêineres do Kubernetes. A utilização de NFS para distribuição do conteúdo nos trás a vantagem de que o PV guarda a configuração de endereço do diretório de rede que será visto entre quaisquer nós da rede Kubernetes, sejam eles já existentes ou recentemente criados, o processo já é automatizado na implantação de cada *Pod*, além de centralizar todo o conteúdo em um único espaço acessível.

Com a construção dos *Pods* e aquisição dos contêineres é possível criar um serviço, ou seja, uma abstração de sua composição que defina como suas funcionalidades poderão ser visualizadas e acessadas por agentes externos. Os serviços definidos dos *Pods* existentes também definem a política de acesso ao conteúdo contido nele, de forma que o tipo de serviço escolhido dita tal comportamento.

Na estrutura aqui definida teremos o serviço exposto de duas maneiras diferentes: com tipo NodePort, indicando que podem ser acessados através do endereço IP do nó em que o *Pod* atual reside, com uma porta de acesso a princípio aletória mas tem seu valor aqui definido (porta 31450), contida dentro de um intervalo padronizado pela estrutura que provê o tratamento necessário para evitar colisão de portas na instanciação de dois serviços. Assim, na distribuição do serviço por NodePort os manifestos DASH serão

acessados através do endereço IP do nó e sua respectiva porta. Exemplo:

```
http://NodeIP:Porta/playlist.xspf
```

A segunda forma de exposição do serviço é pela utilização de um balanceador de carga (*LoadBalancer*). Uma vez que o cliente de vídeo iniciar requisições para este serviço, definido no endereço *video.example.com:31080* e associado ao IP 192.168.0.120, os envios das requisições serão alternados entre os *Pods* identificados com a marcação **video-dash-loadbalancer=sim** e estes se encarregam de distribuir a carga direcionada para eles, devolvendo cada segmento de mídia para o cliente.

O trabalho de distribuição de carga é feito em modelo *round robin* realizando primeiramente alternância sequencial cíclica entre um número selecionado de nós delimitado por uma variável interna que indica uma porcentagem mínima a ser verificada, do primeiro até o último da sequência, a partir da avaliação de viabilidade de uso de cada nó feito pelo *kube-scheduler* que os atribui um tipo de nota para então selecionar os nós que se saíram melhor em sua avaliação. A variável de delimitação da verificação faz com que o *kube-scheduler* termine a verificação entre nós assim que a porcentagem de nós viáveis for alcançada, independente no número total de nós existentes, com o intuito de melhorar a performance desta tarefa à medidade que um aglomerado cresce. O valor padrão utilizado é de 50% podendo ser configurado para variar em 1 e 100, sendo que para aglomerados onde o número de nós existentes não ultrapassa o valor de verificação todos os nós serão avaliados. Para um nó escolhido por sua nota, o componente continuará a avaliação de viabilidade de uso a partir do último *Pod* selecionado utilizando o mesmo método *round robin*.

Neste cenário, é definido um número de réplicas a serem geradas a partir do *Pod* instanciado com serviço *loadBalancer* e cada réplica trabalhará dentro do *cluster* gerado para redirecionar as requisições recebidas.

3.2 Geração de vídeos com padrão DASH

O processo de geração dos vídeos no formato para utilização com o protocolo DASH se divide em duas etapas: divisão do vídeo original entre diferentes configurações e geração do manifesto DASH contendo os vídeos obtidos.

Para efetuar a divisão dos vídeos é utilizada a ferramenta FFMPEG [FFMPEG], capaz de realizar conversões e determinadas edições de arquivos de áudio e vídeo a partir de um conjunto de bibliotecas, algumas incluídas pela biblioteca de sistema libav, e funcionalidades habilitadas pelo usuário acopladas à ferramenta. FFMPEG recebe instruções a partir de linha de comando, tal que uma gama de parâmetros possibilita completo controle e personalização da tarefa realizada.

No trabalho realizado, o padrão de controle de configuração criado para a divisão dos vídeos no FFMPEG segue a estrutura da Tabela 1:

Taxa de quadros por segundo	Resolução	Taxa de bits (kbps)	Tamanho de buffer (kbps)
18	426x240	224	201
24	854x480	1200	1080
30	1280x720	3360	3024

Tabela 1 – Configuração estabelecida para os vídeos a serem utilizados.

As configurações apresentadas foram escolhidas após compararmos valores recomendados por diferentes serviços de transmissão de vídeos, como YouTube [YouTube - *Recommended encoding settings*] e Twitch [*Twitch Broadcasting Guidelines*] sendo alguns exemplos, chegando em uma aproximação que se encaixasse no ambiente desejado para a plataforma. Seria ideal obter uma resolução de pelo menos 720p para se ter uma qualidade mais alta de visualização com algo comparável a um serviço de transmissão próprio e, a partir desta resolução, decrementaríamos os conjuntos de valores de resolução, taxa de quadros e taxa de bits também se encaixando com as configurações recomendadas analisadas enquanto que o tamanho de buffer variasse para um valor de pelo menos metade da taxa de bits, isso porque o tamanho de buffer será o fator principal a ser considerado pelo FFMPEG para tentar manter a taxa de bits constante durante a codificação dos novos vídeos. A taxa de bits refere-se à quantidade de bits a serem processados em um período de tempo durante a execução do arquivo de vídeo, que pode impactar diretamente a qualidade de visualização caso não seja usado um valor mais compatível à resolução produzida, por isso se torna importante testar valores variados para o tamanho de buffer até que se encontre uma faixa aceitável. Neste tabalho, decidimos usar um tamanho de buffer igual a 90% do valor da taxa de bits por se mostrar uma configuração mais compatível às resoluções finais escolhidas.

Decrementar os valores de taxa de quadros foi uma escolha arbitrária para o projeto, uma vez que não é uma opção geralmente considerada para as plataformas utilizadas como referência sendo mais comum manterem a taxa de quadros constante de acordo com o vídeo disponibilizado no *upload*. Essa escolha foi feita como um método de demonstrar os níveis de qualidade a serem alcançados durante os experimentos uma vez que cada taxa de quadros consegue referenciar diretamente a resolução atrelada ao mesmo tempo que a divergência existente para essa propriedade pudesse se tornar mais uma métrica relevante para o conjunto de dados.

Em todos os casos prevaleceu o uso do codificador x264 [Libx264] pela escolha de vídeos no formato MP4 e ser uma biblioteca de codificação de vídeos amplamente utilizada entre os meios atuais de transmissão de vídeo, além de segmentos de quatro segundos contendo, para cada segundo de segmento, a quantidade de quadros referente

àquela configurada para o vídeo. Exemplificando, se o vídeo foi configurado com 18 quadros por segundo então cada segmento terá um total de 72 quadros com um *keyframe* contido a cada 18 quadros, ou seja, um *keyframe* por segundo de segmento. *Keyframe* se refere ao tipo de quadro de animação que define o início e fim de uma transição entre conjuntos de quadros num vídeo, carregando a maior quantidade de informação comparado aos outros quadros, geralmente indicando mudanças mais drásticas na imagem como mais movimentos de elementos inclusos no momento de um vídeo. É necessário que cada *keyframe* esteja alinhado com seu grupo de imagens corretamente para garantir que a transição de resoluções ocorra sem problemas.

Duas faixas de áudio são extraídas de cada vídeo, sendo considerada uma faixa de alta qualidade de 128 kbps e uma faixa de baixa qualidade de 32 kbps. O tamanho de cada faixa de áudio determina tanto a qualidade do áudio como sua compressão, de modo que com 128kbps temos uma qualidade de áudio comparada com uma taxa de bits mínima de arquivos MP3, não é uma qualidade de áudio comparável com os padrões atuais mas é o suficiente para que o áudio esteja em um nível aceitável ao mesmo tempo que sua compressão e envio ao cliente de vídeo se tornam mais fáceis. Faixas geradas com 32kbps perdem qualidade consideravelmente, a ponto de que se desfaz de vários efeitos adicionais de áudio, se tornando uma opção vantajosa apenas em situações críticas em que o cliente ainda tenta recuperar áudio mas necessita de uma faixa ainda mais leve para conseguir entregar o áudio durante a visualização do conteúdo. O *sample rate*, valor geralmente representado em *kiloHertz* (kHz) que indica a frequência da quantidade de amostras de áudio transportadas por segundo, é mantido em seu valor original que varia entre 44.1kHz e 48kHz para os vídeos utilizados no projeto visto que não se tornou uma propriedade que definiria mudanças importantes para os dados a serem observados. Com os valores já existentes do *sample rate* temos uma representação mais próxima da frequência de 20kHz percebida pelo ouvido humano, sendo que a frequência capturada é por volta de metade da taxa de amostragem, e uma propriedade com maior foco na percepção de áudio de um usuário seria menos interessante no contexto das métricas buscadas.

Neste formato, cada vídeo utilizado no serviço de transmissão estará sujeito a níveis de qualidade aceitáveis (alto, médio) e um nível de qualidade considerado mais crítico (baixo), indicando que clientes ou servidores estão atuando com poucos recursos durante a transmissão do conteúdo.

A extração das faixas de áudio é feita pelos comandos:

```
$ ffmpeg -i myVideo.mp4 -c:a aac -ac 2 -b:a 32k -vn
    myVideo_audio_32k.mp4
$ ffmpeg -i myVideo.mp4 -c:a aac -ac 2 -b:a 128k -vn
    myVideo_audio_128k.mp4
```

- -c:a: codec utilizado no processamento, especificando que o fluxo de áudio será trabalhado;
- -ac <2>: número de canais de áudio;
- -b:a <32k; 128k>: taxa de bits da faixa de áudio;
- -vn: vídeo é removido do processamento.

A divisão dos arquivos de vídeo é feita adaptando o seguinte comando:

```
$ ffmpeg -i myVideo.mp4 \
  -an -r <FRAMERATE> \
  -c:v libx264 -x264opts \
  'keyint=M:min-keyint=N:no-scenecut' \
  -b:v <BITRATE> \
  -maxrate <MAXRATE> \
  -bufsize <BUFSIZE> \
  -vf 'scale=X:Y' myVideo_new_format.mp4
```

- -an: Áudio é removido do processamento;
- -r <FRAMERATE>: Taxa de quadros por segundo alvo do novo vídeo;
- -c:v <codec>: Codec utilizado no processamento, especificando que o fluxo de vídeo será trabalhado;
- keyint: Tamanho do intervalo do grupo de imagens (GOP);
- min-keyint: Tamanho mínimo do intervalo do grupo de imagens. Junto com keyint, define o tamanho de um GOP com M quadros por segundo e N *keyframes* por segmento;
- no-scenecut: Elimina funcionalidade de detecção de corte de cena para separação de *keyframes*.
- -b:v <BITRATE>: Taxa de bits alvo do vídeo, o valor da opção indica uma aproximação média do intervalo de valores a ser assumido;
- -maxrate <MAXRATE>: Delimita o valor máximo que a taxa de bits consegue assumir.
- -bufsize <BUFSIZE>: Controla a taxa de verificação da taxa de bits do vídeo no momento de processamento, com intiuto de mantê-la o mais próximo possível do intervalo definido pelo valor de -b:v.

- scale=X:Y: Resolução do vídeo

Após o processamento do arquivo de vídeo original, o FFMPEG gera um novo arquivo contendo os segmentos descritos pelo comando informado. Cada segmento contém um número de quadros por duração de segmento, onde cada conjunto de quadros por segundo deve conter pelo menos um *keyframe* que possibilita a visualização do início do segmento no cliente. Neste projeto, utilizamos segmentos de quatro segundos que contém quatro *keyframes*. A quantidade de *keyframes* num segmento dependerá do tamanho do segmento e do tamanho do grupo de imagens (*GOP - Group of Pictures*), onde a diferença entre estes dois valores define o tamanho do intervalo entre *keyframes* dentro de um segmento.

Uma sequência de quadros contida num segmento de vídeo forma um grupo de imagens. GOPs são separados em três tipos de quadros: *intra coded picture (I-frame), predictive coded picture (P-frame) e bipredictive coded picture (B-frame). I-frames* ou *keyframes*, como denotados anteriormente, possuem uma imagem completa dentro de um quadro, tornando-se maior que os outros quadros inclusos no GOP. *P-frames* e *B-frames* formam o agrupamento de quadros encarregados de incluir mudanças adicionais, tanto quanto menores, na imagem completa trazida pelo *keyframe*, se diferindo no método de compressão dos dados que contém de forma que um *P-frame* utiliza informações do quadro imediatamente anterior para descompactar uma imagem enquanto que um *B-frame* pode usar tanto o quadro anterior quanto o próximo quadro da cadeia do GOP como referência. O tamanho do GOP pode ser definido a partir do cálculo entre tamanho de segmento * taxa de quadros onde o resultado obtido deverá ser um mútiplo do tamanho do GOP para garantir que o intervalo entre *keyframes* esteja alinhado com o início de cada segmento.

Arquivos de vídeos segmentados disponibilizados num MPD podem existir como um arquivo único contendo todos os segmentos ou como um conjunto de vários arquivos, separados por um segmento de vídeo cada. Tais segmentos formam o ponto principal de funcionamento do padrão DASH, pois definem onde as transições de vídeo podem ocorrer.

3.2.1 *Media Presentation Description* e transição entre resoluções

A criação do manifesto é feita com a ferramenta MP4Box [MP4Box]. Assim como o FFMPEG, também é executado por linha de comando, utilizando como parâmetros o tempo representativo do tamanho de cada segmento, as segmentações de vídeo e faixas de áudio processadas pelo FFMPEG, extraindo os metadados para gerar o arquivo de manifesto compatível com o padrão DASH.

A transmissão dos vídeos é realizada pela obtenção do arquivo de manifesto pelo cliente. O MPD conterá todas as informações necessárias para que arquivos de vídeo e áudio sejam recuperados e a transição entre segmentos de diferentes qualidades de

visualização ocorra, tal que a estrutura do MPD indicará quais segmentações de vídeos e faixas de áudio estão disponíveis, assim como a definição do relacionamento entre estes dois componentes, para cada representação sendo informada a um cliente. Tal estrutura é apresentada pela Figura 3.

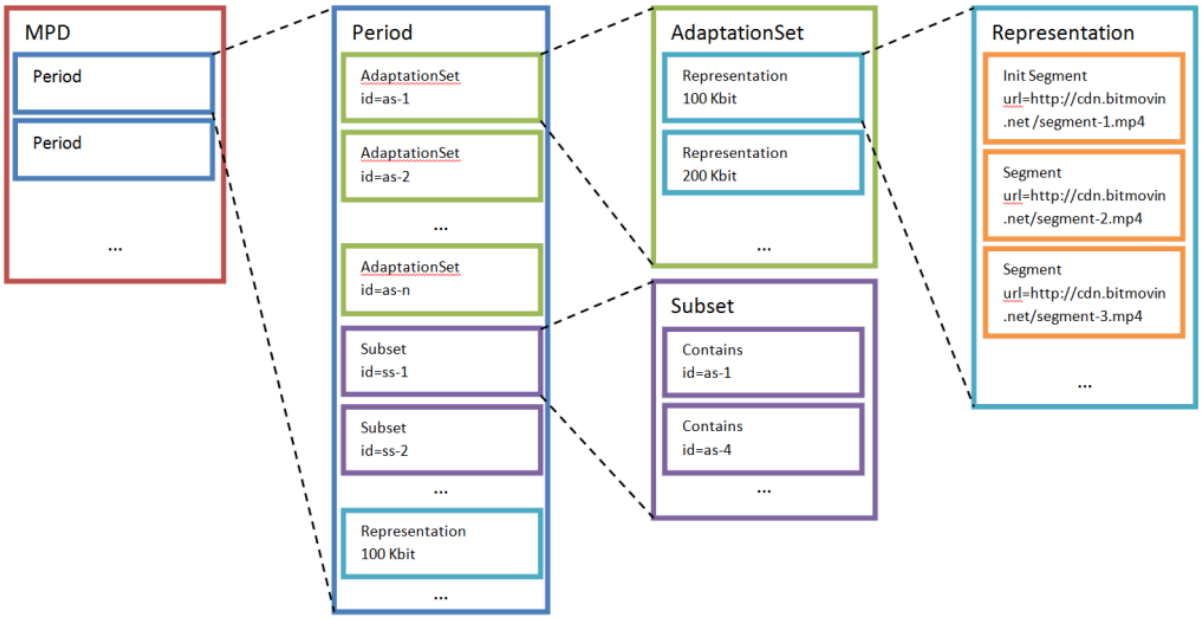

Figura 3 – Modelo padronizado de manifesto MPEG-DASH. Fonte: https://bitmovin.com/dynamic-adaptive-streaming-http-mpeg-dash/ [Modelo MPD]

A geração do manifesto com MP4Box é feita adaptando o seguinte comando:

```
$ MP4Box \
 -dash <SEGTIME> \
 -rap \
 -frag-rap \
 -profile <PROFTYPE> \
 -out myVideo_manifest.mpd myVideo_new_format.mp4
    myVideo_audio_128k.mp4 myVideo_audio_32k.mp4
```

- -dash <SEGTIME>: habilita segmentos em padrão DASH definidos nos arquivos de entrada para o manifesto.
- -rap: força segmentos a terem pontos de acesso aleatórios entre *keyframes*.
- -frag-rap: força os fragmentos dos segmentos a terem pontos de acesso aleatórios.

- -profile <PROFTYPE>: define o tipo de perfil do arquivo a ser transmitido, *live* gera um arquivo de mídia por segmento e *on demand* junta todos os segmentos em um único arquivo de mídia.
- -out <MANIFESTO>: define o arquivo de manifesto a ser gerado.

Para cada MPD, são informados ambos os arquivos de vídeo segmentados e arquivos de áudio a serem inclusos nos *Adaptation Sets*.

Os vídeos segmentados são requisitados pelo cliente a partir das representações encontradas no manifesto, onde cada par de representação indica um arquivo segmentado de vídeo e uma faixa de áudio. Os metadados são analisados e, de acordo com o protocolo de adaptação, o cliente analisa o estado da rede à medida que envia novas requisições verificando se recursos como largura de banda disponível é suficiente para recuperar e visualizar um segmento de vídeo de qualidade mais alta. O segmento considerado como a melhor opção de visualização será requisitado no servidor.

De acordo com a metodologia apresentada pelo protocolo DASH, a transição entre segmentos deve ser sutil ao usuário e eliminar a possibilidade de travamento do vídeo por insuficiência de recursos. A definição de três diferentes níveis de qualidade para o conteúdo utilizado neste trabalho proporciona tanto flexibilidade no funcionamento da reprodução de vídeos no padrão DASH, assim que mais níveis de qualidade serão úteis para manter um fluxo constante de requisição de segmentos diversos, como mais dados comparativos possíveis de se adicionar à coleção de métricas para aprendizado e a ocorrência de transição entre resoluções proporciona maior variação de dados.

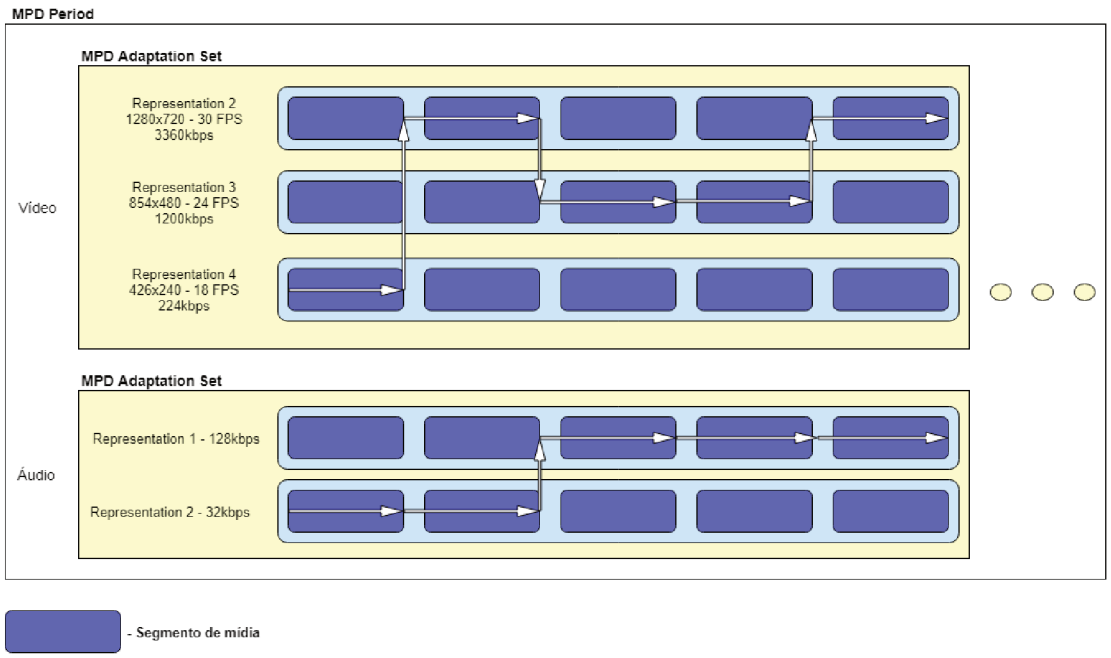

Figura 4 – Representação de transmissão adaptativa dinâmica com padrão do protocolo DASH.

A Figura 4 ilustra um exemplo de como a transição entre resoluções pode ocorrer de acordo com as configurações indicadas na Tabela 1. As transições entre resoluções de vídeo e faixas de áudio são feitas separadamente, visto que nos MPDs estão definidos em *Adaptation Sets* diferentes e não é determinado um vínculo entre estes dois componentes.

Realizados os passos de implementação do protocolo DASH e escolhida uma plataforma apta ao seu tipo de representação de mídia, os vídeos que receberem este tratamento serão elegíveis para efetuar a troca de resoluções a qualquer momento de atividade como exemplifica a Figura 5 mostrando três quadros de um mesmo vídeo que podem ser transmitidos em qualquer uma das configurações disponíveis.

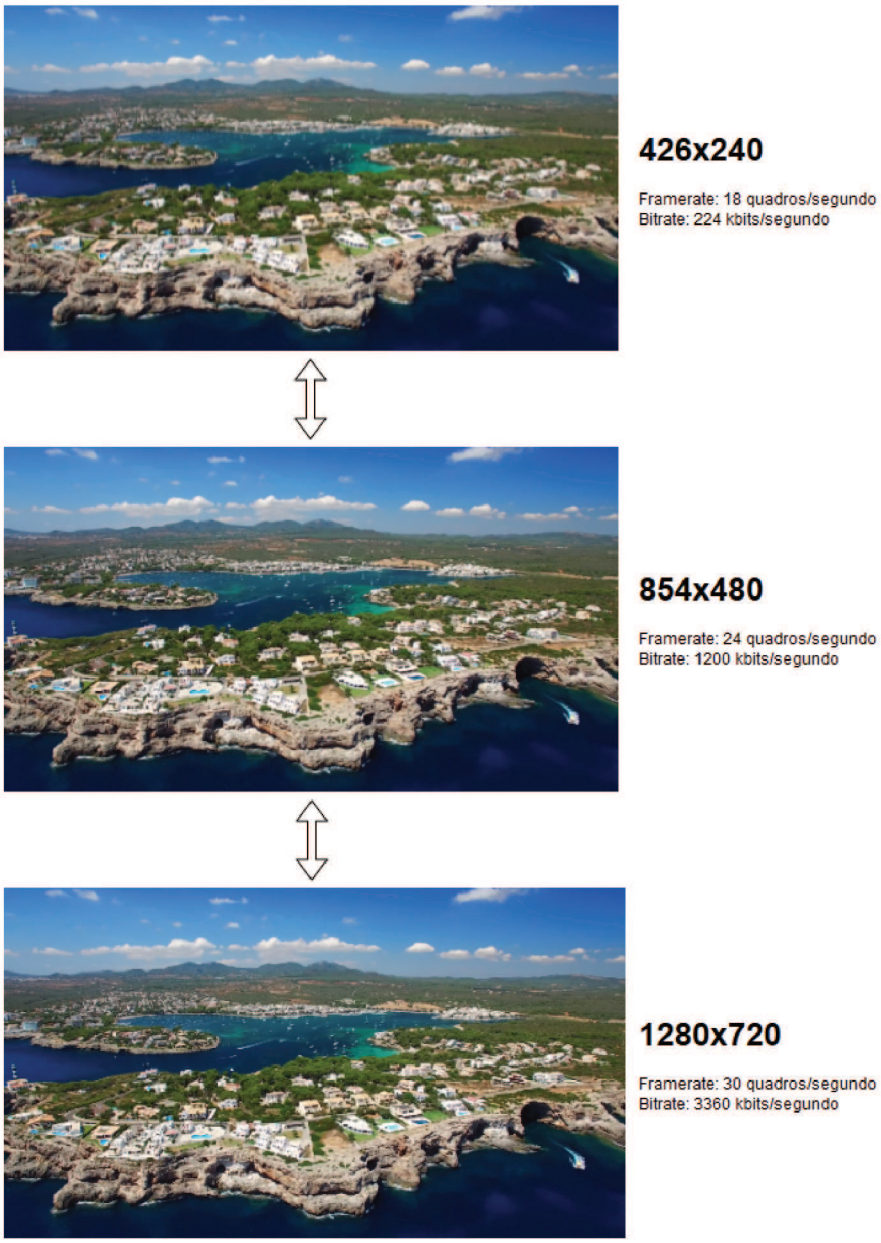

Figura 5 – Exemplificação do emprego de três configurações distintas a serem utilizadas pelo protocolo DASH.

3.3 Cliente de vídeo

Para a realização do projeto, o cliente de vídeo a ser chamado pelo serviço instanciado pelo Kubernetes teria que atender dois importantes requisitos: a capacidade de suportar o protocolo DASH e ser um projeto disponibilizado em código aberto para livre modificação por terceiros.

O cliente de vídeo escolhido para atuar na plataforma foi o VLC [VideoLAN Project - VLC Media Player], visto que as versões disponíveis no momento de escrita deste papel passaram a ter suporte ao padrão DASH (primeira versão oficial com suporte a partir de Fevereiro de 2018), auxiliando no escopo de obtenção de métricas por também ser distribuído em código aberto, adquirido no repositório do GitHub [Repositório VLC 3.0.3]. Dessa forma, já temos a capacidade de modificá-lo para capturar as métricas a serem escolhidas para análise.

O *fork* criado para o projeto [Github Fork VLC] teve o arquivo *stats.c* alterado, visto que a estrutura composta dos valores das propriedades de áudio e vídeo de um arquivo de mídia (input_stats_t) era aqui acessada, feito assim um trabalho simples de gravação dos valores encontrados para um arquivo de extensão *.log* na VM que executaria este cliente modificado, futuramente indicada por *client-dash*.

3.4 Conjuntos de Dados de Treinamento

Os conjuntos de dados de treino, denominados X e Y, são formados pelas métricas consideradas mais relevantes para o treinamento do algoritmo de regressão, formando os respectivos agrupamentos de valores dependentes e independentes. Cada conjunto têm suas métricas coletadas de formas diferentes: os dados do conjunto X são coletados por uma ferramenta de monitoramento de sistemas que fica responsável por inspecionar a condição dos componentes da estrutura Kubernetes criada para o projeto e abstrair os dados escolhidos e os dados do conjunto Y pelo cliente de vídeo modificado para capturar e registrar metadados dos vídeos em padrão DASH. Todos os dados coletados são registrados em um intervalo de um segundo de diferença.

Os dados disponíveis em cada conjunto serão necessários para verificar o comportamento da estrutura de transmissão de vídeos à medida que os geradores de carga forem acionados.

A partir do cliente de vídeo modificado são catalogadas quatro métricas, retornadas como valores numéricos, para o conjunto Y:

- *framesDisplayed*: quantidade de quadros de imagem exibidos.

- *playedAudioBuffers*: quantidade de quadros de áudio em buffer que foram reproduzidos.
- *decodedVideo*: quantidade de quadros de imagem decodificados e repassados para o dispositivo de saída de vídeo.
- *decodedAudio*: quantidade de quadros de áudio decodificados repassados para o dispositivo de saída de áudio.

O sistema de monitoramento Prometheus [*Prometheus Monitoring System*] oferece quatro tipos principais de métricas, diferenciadas pelas bibliotecas do cliente. *Counter* representa um tipo de métrica cumulativa podendo ser apenas incrementada ou resetada para cada vez que o serviço reinicia. *Gauge* representa um valor numérico, podendo indicar dados quanto o uso de componentes, como memória, que é incrementado ou decrementado arbitrariamente. *Histogram* tira amostras de valores relacionados, por exemplo, a tempo de requisições e tamanho de respostas gerando contadores que são armazenados de acordo com a configuração estabelecida, dessa forma conseguindo prover também uma totalização de todas as observações adquiridas. Os armazenamentos utilizados pelo tipo *Histogram* denominam-se baldes de observação (*observation buckets*). Finalmente, têm-se o tipo *Summary* que é semelhante ao *Histogram* diferindo no aspecto em que deriva das observações alcançadas um cálculo das quantias em relação a uma janela de tempo.

A partir da integração entre Kubernetes e Prometheus, é possível usufruir da ferramenta interna de monitoramento e coleta de informações cAdvisor, um *daemon* que é capaz de obter informações de uso de CPU, memória, rede e do sistema de arquivos dos contêineres gerados, de forma que este processo de monitoramento roda nos nós que os contêineres habitam.

4 Experimentos

4.1 Geradores de carga

Os geradores de carga têm a tarefa de alterar o comportamento da rede Kubernetes através do envio de quantidades alternadas de carga extra para cada nó onde o serviço de transmissão de vídeos opera. Novas previsões calculadas pelo algoritmo de regressão partem dos dados adquiridos dentro do período em que os servidores recebem a carga extra dos geradores, criando novas situações em que é possível comparar o status dos nós a cada momento de funcionamento do serviço.

São utilizados três tipos de geradores de carga na plataforma proposta: *constant*, *periodic* e *flashcrowd*.

- *Constant*: mantém um fluxo persistente de carga sem alterar a quantidade de clientes de vídeo ao longo da execução, tendo assim um tempo de vida igual ao tempo de execução do experimento e simulando um período mais longo de alto funcionamento do serviço, extenuando-o a ponto de que tenha um impacto considerável na performance observada através das variáveis obtidas do cliente de vídeo.
- *Periodic*: a geração de carga neste cenário ocorre periodicamente, dentro de intervalos de tempo de duração pré-determinada, onde a carga extra é continuamente enviada aos servidores enquanto o gerador está ativo e forma uma sinusóide dentro deste período. É definida uma taxa de geração de novos clientes que funciona de maneira proporcional a um valor central definido, entrando em ação à medida que os processos em execução são terminados de acordo com o tempo de vida parametrizado. Assim, a carga gerada dentro de um período deverá durar um determinado tempo antes que os processos atuais sejam eliminados da fila de execução e uma nova quantidade seja gerada continuando o ciclo.
- *Flash crowd*: o gerador *flash crowd* tem a função de espelhar uma situação em que rapidamente surge um grande aumento no trágefo de um serviço, levando à rapida degradação da performance dos servidores. Da mesma forma que o tráfego originado de uma *flash crowd* manifesta-se subitamente, ele se desfaz. Neste modelo teremos um número inicial definido de clientes que atingirá um pico, mantendo-se assim por determinado tempo, durante eventos relâmpago que ocorrem aleatoriamente dentro do período de execução.

A carga imposta pelos geradores é formada pela criação de novos processos do cliente de vídeo. Em ambos os modelos, os geradores de carga utilizam um processo

Poisson para definição da taxa de chegada de clientes por minuto de execução. O número definido de clientes em execução enviam requisições continuamente para segmentos dos vídeos armazenados e esperam que o servidor seja capaz de se manter responsivo, ou seja, a taxa de entrega de segmentos deve se manter o suficiente para que a reprodução dos vídeos seja mantida de acordo com o funcionamento do protocolo DASH, apenas alternando a qualidade de reprodução do vídeos conforme necessário.

Os geradores de carga são executados com uma série de parâmetros assim definidos:

- Carga Periódica: Amplitude (A) impondo variância sob quantidade de clientes durante período (P);
- Carga Flashcrowd: Fator de choque (S) com uma quantidade de eventos/hora (C), diminuindo de acordo com um valor de desaceleração (N).
- Parâmetros globais: Duração total do experimento e taxa média do número de clientes executados por minuto em relação a duração total da execução do gerador.

Os clientes de vídeo iniciados pelo gerador de carga são executados com os parâmetros: *–adaptive-logic=rate*, *–no-qt-privacy*, *–random*, *–repeat*, *-I*, *dummy* e *–zoom=0.15*. O parâmetro *–zoom* é usado para que a grande quantidade de clientes do gerador possam executar simultaneamente com a interface de vídeo habilitada, isso porque precisamos que tanto requisições para vídeo e áudio sejam feitas, sem que ocorram travamentos por falta de força computacional da VM *loadgen-dash*.

Os períodos ativos do gerador de carga deverão deixar sua marca nos experimentos da plataforma com faixas de tempo compreendendo tanto os períodos P na execução periódica quanto a quantidade de eventos/hora C da carga *flashcrowd* com alterações mais bruscas nos valores então observados através dos logs de execução do cliente VLC, sendo possível compreender que existe uma ampla carga externa equivalente a requisições do mesmo tipo daquelas deslocadas entre o cliente VLC modificado e os nós Kubernetes que também se mostra bem sucedida nos resultados esperados.

4.2 Vídeos disponíveis

Nos servidores Apache localizados nos nós da rede Kubernetes são disponibilizados quatorze diferentes vídeos com um total de quarenta e dois vídeos gerados que complementam o conteúdo em padrão DASH, sendo estes separados em três versões com valores determinados para taxa de quadros por segundo, tamanho de buffer e resolução. Essa divisão foi escolhida para que diferentes cenários pudessem ser abordados, tendo em vista que a configuração do arquivo de vídeo impactará o comportamento do cliente de vídeo ao tentar requisitar níveis de qualidade maiores.

Na Tabela 2, são listados os vídeos originais utilizados para padronização DASH.

Vídeo	Duração	Resolução	Quadros por segundo	Taxa de bits Total (kbps)	Taxa de bits de áudio (kbps)
Around The World	01:00:00	1280x720	30	17578	130
Big Buck Bunny	00:10:34	1920x1080	59.94	5487	125
Biosphere	01:08:13	1920x1080	60	2309	383
Einstein	01:29:37	1280x720	30	2435	125
HK_60fps	00:03:26	1376x776	59.83	21346	131
ImmigCenter	00:56:46	1280x720	30	12047	125
Jazz_60fps	00:02:38	1920x1080	59.94	5487	125
Midir_60fps	00:05:13	1280x720	60	3453	125
Snowboard_60fps	00:04:44	1280x720	59.94	3502	125
Spanish Isles	00:51:47	1920x1080	30	3358	125
Tesla Motors	00:50:16	1920x1080	30	2149	125
Transistor_60fps	00:05:22	1376x776	59.90	21433	132
Vancouver	00:57:53	1920x1080	30	2937	125
Viena	00:55:18	1920x1080	30	2115	125

Tabela 2 – Vídeos originais utilizados para a geração dos vídeos em padrão DASH.

Foi seguido o seguinte padrão para obtenção do conteúdo: buscamos utilizar vídeos com qualidade mínima de 720p (*standard HD*) e, a partir dos obtidos, gerar os novos vídeos degradando-os a cada passo da execução do FFMPEG. Dessa forma temos maior controle sobre a definição da resolução, taxa de quadros e tamanho do intervalo de quadros para composição do grupo de imagens alvo da sequência de vídeos, elementos que se mostraram mais importantes para garantir não só a configuração correta dos novos vídeos mas também para apresentar uma aproximação dos valores que seriam esperados durante os testes.

Com intuito de simular diferentes classes de visualização do conteúdo abrangendo tanto imagens de alta quanto baixíssima qualidade, assim como é visto em serviços de transmissão de vídeos mais atuais, a qualidade máxima a ser alcançada deverá demandar mais da interação iniciada pelo cliente de vídeo.

A nomenclatura dos vídeos gerados em padrão DASH segue o formato:
NomeOriginal_Resolucao_TaxaQuadros_TaxaBitsMaxima_TamanhoBuffer_dashinit.mp4

A relação total de vídeos padronizados para DASH é detalhada na Tabela 3:

Manifesto	Vídeos em padrão DASH
aroundTheWorld_30fps_buf90.mpd	aroundTheWorld_426x240_18_224k_buf90_dashinit.mp4
	aroundTheWorld_854x480_24_1200k_buf90_dashinit.mp4
	aroundTheWorld_1280x720_30_3360k_buf90_dashinit.mp4
bigBuckBunny_30fps_buf90.mpd	bigBuckBunny_ 426x240_18_224k_buf90_dashinit.mp4
	bigBuckBunny_854x480_24_1200k_buf90_dashinit.mp4
	bigBuckBunny_1280x720_30_3360k_buf90_dashinit.mp4
biosphere_30fps_buf90.mpd	biosphere_ 426x240_18_224k_buf90_dashinit.mp4
	biosphere_854x480_24_1200k_buf90_dashinit.mp4
	biosphere_1280x720_30_3360k_buf90_dashinit.mp4
einstein_30fps_buf90.mpd	einstein_426x240_18_224k_buf90_dashinit.mp4
	einstein_854x480_24_1200k_buf90_dashinit.mp4
	einstein_1280x720_30_3360k_buf90_dashinit.mp4
hk_30fps_buf90.mpd	hk_426x240_18_224k_buf90_dashinit.mp4
	hk_854x480_24_1200k_buf90_dashinit.mp4
	hk_1280x720_30_3360k_buf90_dashinit.mp4
immigCenter_30fps_buf90.mpd	immigCenter_ 426x240_18_224k_buf90_dashinit.mp4
	immigCenter_854x480_24_1200k_buf90_dashinit.mp4
	immigCenter_1280x720_30_3360k_buf90_dashinit.mp4
jazz_30fps_buf90.mpd	jazz_426x240_18_224k_buf90_dashinit.mp4
	jazz_854x480_24_1200k_buf90_dashinit.mp4
	jazz_1280x720_30_3360k_buf90_dashinit.mp4
midir_30fps_buf90.mpd	midir_426x240_18_224k_buf90_dashinit.mp4
	midir_854x480_24_1200k_buf90_dashinit.mp4
	midir_1280x720_30_3360k_buf90_dashinit.mp4
snowboard_30fps_buf90.mpd	snowboard_426x240_18_224k_buf90_dashinit.mp4
	snowboard_854x480_24_1200k_buf90_dashinit.mp4
	snowboard_1280x720_30_3360k_buf90_dashinit.mp4
spanishIsles_30fps_buf90.mpd	spanishIsles_426x240_18_224k_buf90_dashinit.mp4
	spanishIsles_854x480_24_1200k_buf90_dashinit.mp4
	spanishIsles_1280x720_30_3360k_buf90_dashinit.mp4
teslaMotors_30fps_buf90.mpd	teslaMotors_426x240_18_224k_buf90_dashinit.mp4
	teslaMotors_854x480_24_1200k_buf90_dashinit.mp4
	teslaMotors_1280x720_30_3360k_buf90_dashinit.mp4
transistor_30fps_buf90.mpd	transistor_426x240_18_224k_buf90_dashinit.mp4
	transistor_854x480_24_1200k_buf90_dashinit.mp4
	transistor_1280x720_30_3360k_buf90_dashinit.mp4
vancouver_30fps_buf90.mpd	vancouver_426x240_18_224k_buf90_dashinit.mp4
	vancouver_854x480_24_1200k_buf90_dashinit.mp4
	vancouver_1280x720_30_3360k_buf90_dashinit.mp4
viena_30fps_buf90.mpd	viena_426x240_18_224k_buf90_dashinit.mp4
	viena_854x480_24_1200k_buf90_dashinit.mp4
	viena_1280x720_30_3360k_buf90_dashinit.mp4

Tabela 3 – Relação de vídeos a 30 quadros/segundo em padrão DASH e seus manifestos.

O diretório raiz dos arquivos de vídeo está localizado em */home/cloud/videos* na VM de endereço 192.168.0.110, onde é feito compartilhamento por NFS deste diretório entre os servidores HTTPD dos nós que compõem o serviço instanciado na estrutura do Kubernetes. Os vídeos são armazenados no subdiretório */test4_buf90_minrate/4s_mpd* junto de seus respectivos manifestos.

No mesmo diretório raiz estão contidas as *playlists* identificadas pelos vídeos contidos nelas. Ao longo dos testes realizados para adequação da plataforma, foram geradas *playlists* compostas dos três formatos possíveis de apenas um dos vídeos, para cada vídeo, a fim de analisar se as configurações escolhidas para a codificação dos vídeos pelo FFMPEG foram bem sucedidas. A Tabela 4 ilustra este quadro:

Playlist	Arquivo de Vídeo	Tempo de Duração da *Playlist*
pl_viena_res.xspf	viena_426x240_18_224k_buf90_dashinit.mp4	02:45:54
	viena_854x480_24_1200k_buf90_dashinit.mp4	
	viena_1280x720_30_3360k_buf90_dashinit.mp4	

Tabela 4 – Playlist de teste do vídeo *Viena*.

Utilizando as *playlists* geradas, é possível testar não só a transição entre resoluções de um só vídeo mas também observar o comportamento do cliente ao transicionar entre diferentes vídeos, neste caso sendo equivalente à aquisição de diferentes manifestos DASH executados por uma playlist, criando mais um cenário útil para o treinamento do algoritmo de aprendizado de máquina.

A Tabela 5 expressa a relação de manifestos contidos na *playlist* principal a ser repassada aos clientes de vídeo:

Playlist	Manifesto	Tempo de Duração da *Playlist* (hh:mm:ss)
pl_30fps-1-v2.xspf	aroundTheWorld_30fps_buf90.mpd	08:31:15
	bigBuckBunny_30fps_buf90.mpd	
	biosphere_30fps_buf90.mpd	
	einstein_30fps_buf90.mpd	
	hk_30fps_buf90.mpd	
	immigCenter_30fps_buf90.mpd	
	jazz_30fps_buf90.mpd	
	midir_30fps_buf90.mpd	
	snowboard_30fps_buf90.mpd	
	spanishIsles_30fps_buf90.mpd	
	teslaMotors_30fps_buf90.mpd	
	transistor_30fps_buf90.mpd	
	vancouver_30fps_buf90.mpd	
	viena_30fps_buf90.mpd	

Tabela 5 – Playlists disponibilizadas pela máquina *nfs-dash*.

4.3 Configurações das máquinas

A estrutura funciona sobre um conjunto de máquinas físicas que separam-se entre os servidores principais que mantém a plataforma organizada e os nós da rede Kubernetes para a sustentação do serviço.

Os nós da rede Kubernetes são instanciados em computadores Raspberry Pi 2 Model B, com 1GB RAM, quatro núcleos ARM, 8GB de armazenamento em disco (memory card) e 100Mbits/s de rede.

Os servidores possuem a seguinte configuração:

- Servidores DELL EMC PowerEdge R740 (14ª Geração)
- Processador Intel Xeon Silver 4114 2.2G, 10C/20T, 9.6GT/s 2UPI, 14M Cache, Turbo, HT (85W) DDR4-2400
- Sistema configurado com dois processadores
- Chassis para até 8 x 3.5"SAS/SATA discos rígidos para configuração de 2 CPUs
- 4x pentes de memória de 16GB RDIMM, 2666MT/s, Dual Rank
- Bezel de segurança
- Configuração de Riser 2, 3 x8, 1 x16 slots
- Configuração de performance otimizada nas memórias
- Configuração de performance na BIOS
- iDRAC9, Enterprise
- 2x discos de 2TB 7.2K RPM SATA 6Gbps 512n 3.5in Hot-plug Hard Drive
- Controladora PERC H330+ RAID, adaptador, Low Profile
- DVD ROM, SATA, interno
- Fontes redundantes Hot-plug (1+1) de 750W de potência
- 2x cabos de força C13, BR14136 (padrão brasileiro), 250V, 10A, 2m de comprimento
- Placa de rede Intel Ethernet i350 QP 1Gb Network Daughter Card
- Sem sistema operacional
- OpenManage DVD Kit, PowerEdge R740
- Configuração dos discos em RAID 1
- Configuração UEFI BIOS Boot Mode

4.4 Versionamento dos componentes

As versões dos componentes utilizados no projeto são definidas a seguir. A denominação "Master" se refere à máquina onde o Kubernetes é executado (192.168.0.30).

- Arquitetura do OS (Master): Linux 4.9.80
- Sistema Operacional (Master):
  - Distribuição: Ubuntu
  - Versão: 17.10
  - Codename: artful
- Dispositivos Raspberry Pi:
  - Distribuição: Raspbian GNU/Linux 9 (stretch)
  - Versão: 9
- Sistema Operacional - Máquinas do experimento (*client-dash, nfs-dash, loadgen-dash*):
  - Distribuição: Ubuntu
  - Versão: 16.04.5 LTS
  - Codename: xenial
- Kubernetes (Master):
  - Cliente: 1.10.2
  - Servidor: 1.10.5
- Docker (Master):
  - Versão cliente/servidor: 1.13.1
  - Versão API: 1.26
  - Arquitetura: linux/amd64
- Kubernetes (Raspberry):
  - Cliente: 1.10.2
  - Servidor: 1.10.5
- Docker (Raspberry):
  - Versão cliente/servidor: 1.13.1

  - Versão API: 1.26
  - Arquitetura: linux/amd64

- Kubeadm:
  - Master: 1.10.2
  - Raspberry: 1.10.1

- Prometheus: 2.3
- Apache Server: 2.4.27
- Apache Server HTTPD (Imagem do repositório DockerHub): 2.4
- VLC: 3.0.3 Vetinari (revision 3.0.2-225-gc9e3360dd4)
- FFMPEG: 4.0, compilado com gcc 5.4.0:
  - libavutil: 56. 14.100
  - libavcodec: 58.18.100
  - libavformat: 58.12.100
  - libavdevice: 58.3.100
  - libavfilter: 7.16.100
  - libswscale: 5.1.100
  - libswresample: 3.1.100
  - libporstproc: 55.1.100
- MP4Box: 0.5.2 (revision 0.5.2-426-gc5ad4e4+dfsg5-3)

4.5 Metodologia aplicada

A metodologia consiste na inicialização da plataforma Kubernetes utilizando uma configuração de implantação, como a quantidade de *Pods* que vão expôr o serviço e o tipo de serviço, assim como a execução dos componentes da plataforma para a captura das métricas de serviço e geração de novos logs de execução. A partir destes, é feita a análise do funcionamento do serviço durante o tempo de experimentação. No processo de inicialização da captura de métricas, ocorrem mudanças nos parâmetros de carga informados ao gerador de carga para possibilitar que novos cenários sejam contemplados a cada execução. A Figura 6 apresenta o fluxograma da realização dos experimentos.

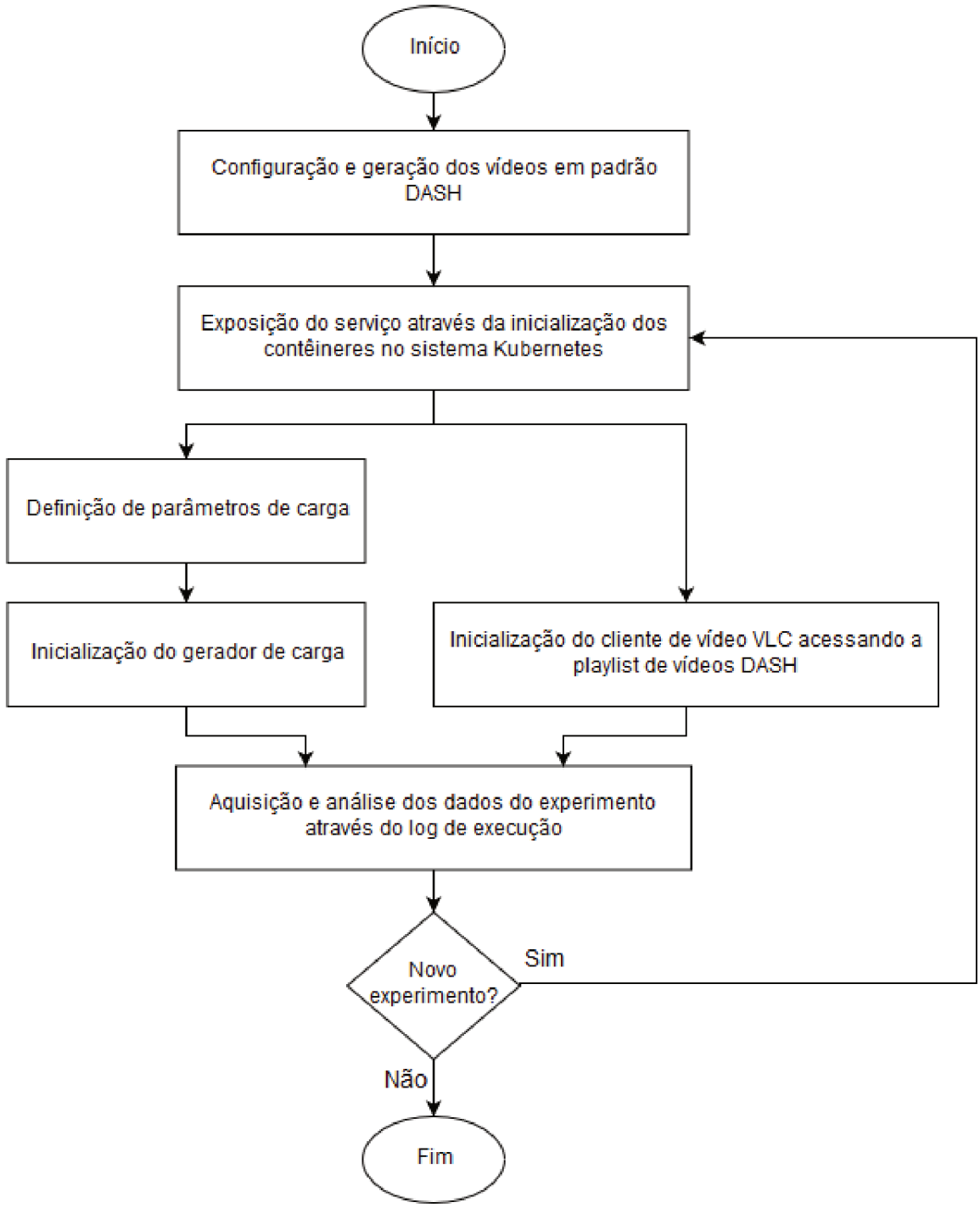

Figura 6 – Diagrama da metodologia aplicada para os experimentos do trabalho.

4.6 Resultados dos experimentos

Os testes para coletas de métricas do conjunto Y foram realizados utilizando os recursos disponíveis nas máquinas *client-dash* (192.168.0.112) e *loadgen-dash* (192.168.0.106) em dois cenários que efetivamente demonstrassem os efeitos do protocolo DASH: primeiro foi executado apenas o cliente de vídeo modificado sem que o acesso ao serviço do Kubernetes fosse comprometido; em seguida foram feitas execuções simultâneas do cliente de

vídeo e do gerador de carga para simular situações de estresse de diferentes magnitudes que causassem mudança no comportamento visualizado a partir dos logs do cliente de vídeo, de acordo com os parâmetros supridos ao gerador de carga.

Na Figura 7 é visualizado o formato que os vídeos em padrão DASH terão ao longo dos experimentos, separados em três níveis de qualidade utilizados na adaptação dinâmica. Na figura vê-se três execuções no total, sendo cada "degrau" um vídeo por completo para cada agrupamento de resolução, taxa de frames por segundo e taxa de bits por segundo associados durante a separação de mídia feita pelo FFMPEG. A taxa de quadros expressa nos resultados (eixo Y) é obtida através da diferença entre dois valores da métrica *framesDisplayed* a cada intervalo de um segundo durante a duração completa do experimento (eixo X). Dessa forma, o valor retornado é a aproximação da taxa de quadros alvo durante um momento na execução dos vídeos.

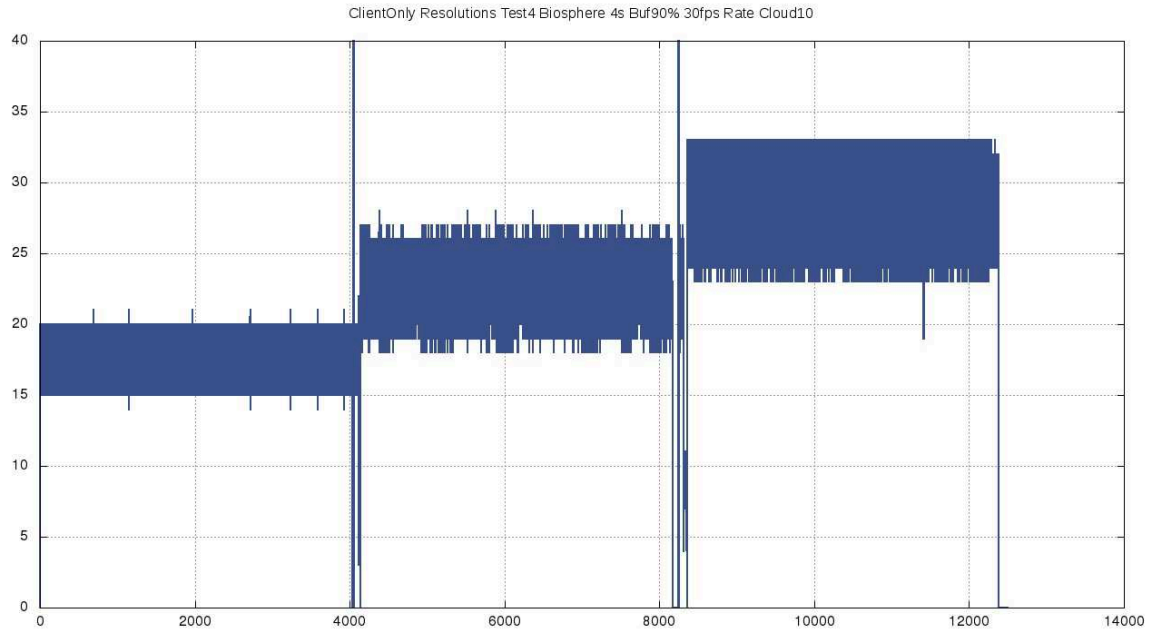

Figura 7 – Teste de execução por resolução disponível do vídeo Biosphere.

A execução contemplada na Figura 7 foi realizada com cada vídeo armazenado para garantir que as configurações definidas na execução do FFMPEG foram feitas corretamente e, uma vez validados, foram iniciados os experimentos de execução com o cliente sobre a carga do gerador.

Os experimentos foram efetuados por períodos de doze horas consecutivas com a *playlist* pl_30fps-1-v2.xspf. Cada instância do cliente de vídeo criada pelo gerador de carga requisita um manifesto aleatório contido na mesma *playlist* passada como parâmetro para o processo.

Na Figura 8 têm-se uma execução completa do experimento contendo todos os

vídeos selecionados após os testes. O gerador de carga foi executado com os parâmetros de amplitude de clientes e taxa média do número de clientes executados iguais a 10 e 25, respectivamente, e o cliente acessou o serviço DASH através do endereço associado aos *Pods* do balanceador de carga que neste experimento compreendia três réplicas do serviço. No resultado obtido é possível visualizar os vários momentos em que a carga gerada teve influência na execução do conteúdo pelo cliente tal que quedas do valor de taxa de quadros ocorrem periodicamente ao longo de cada vídeo.

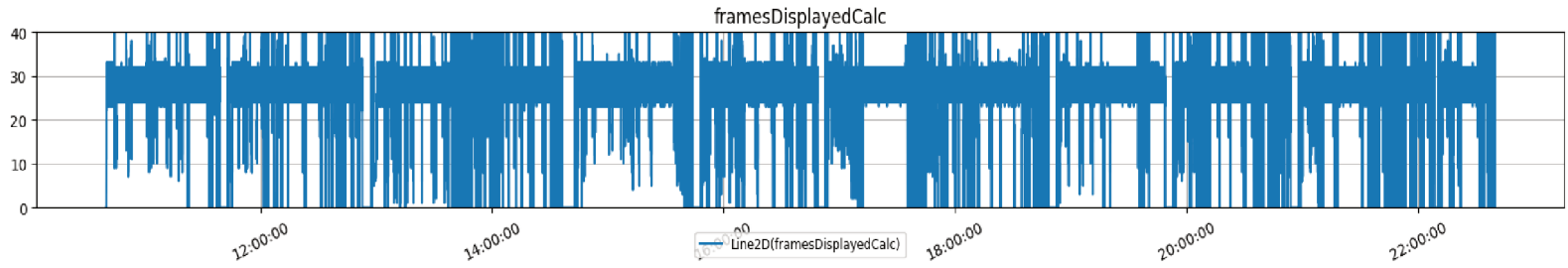

Figura 8 – Execução da *playlist* pl_30fps-1-v2.xspf por período de doze horas com três réplicas de nós do serviço DASH.

É notado um padrão para a taxa de quadros ao longo de todo o experimento onde cada vídeo teve a maioria dos fragmentos recuperados na faixa de 30 quadros por segundo e a carga aplicada causa uma variação bastante visível nos dados observados que, por sua vez, chegou a zerar a métrica por vários momentos.

O próximo experimento, apresentado na Figura 9, foi feito após duas novas réplicas serem acrescentadas à rede. Com cinco réplicas ativas do serviço, o cliente foi executado com a mesma carga do experimento anterior sendo aplicada e na separação de requisições entre mais réplicas se forma a tentativa de aliviar o trabalho feito por cada nó onde o serviço reside. Com o novo número de réplicas o cliente de vídeo ainda reportou vários momentos de queda de quadros, mostrando que a carga ainda foi suficiente para manter as interfaces dos serviços trabalhando para manter a entrega de fragmentos regularizada ao longo do experimento, mas em sua totalidade se mostrou mais eficiente que o experimento da Figura 8.

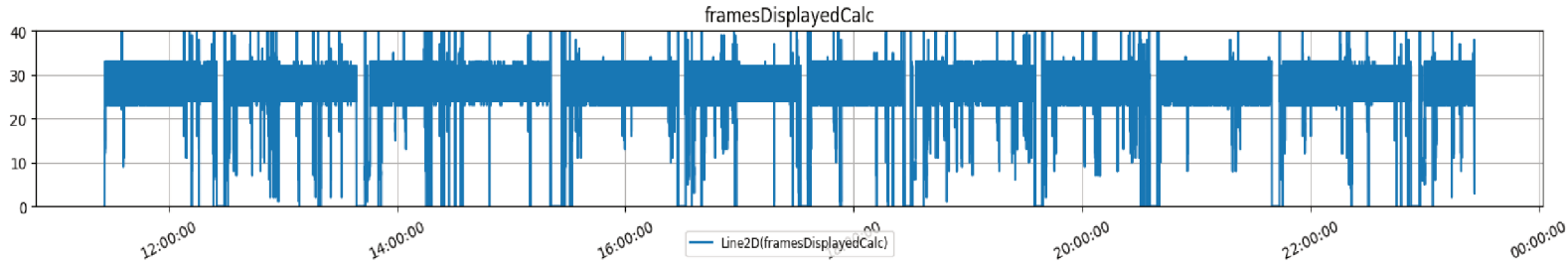

Figura 9 – Execução da *playlist* pl_30fps-1-v2.xspf por período de doze horas com cinco réplicas de nós do serviço DASH.

Com este experimento, demonstramos que a distribuição do acesso ao serviço para um número maior de *Pods* faz com que seja possível atender à crescente oscilação de requisições e ajuda a aliviar esta carga imposta nos componentes da rede. No entanto, essa breve solução pode ser desfeita de acordo com a quantidade de clientes externos que iniciam acesso ao serviço, uma vez que num cenário real o número de acessos poderá alterar consideravelmente num curto espaço de tempo e o número de réplicas atual gerará um desempenho igual ou menos satisfatório do que visto no experimento da Figura 9.

5 Conclusão

Neste capítulo serão destacados os resultados encontrados após executar o serviço de transmissão de vídeos na plataforma construída e o que pôde ser abstraído das métricas coletadas, além dos desafios enfrentados ao longo do estudo do conteúdo e utilização das ferramentas necessárias e, por fim, sugestões de como o projeto poderá ser ampliado no futuro.

5.1 Pesquisa realizada

O objetivo da pesquisa é observar a existência de variação dos parâmetros coletados através dos vídeos, uma vez que o padrão DASH possibilida a transição entre diferentes níveis de qualidade de áudio e vídeo, solidificando a usabilidade da adaptação dinâmica numa situação de crescente demanda e suprir os dados encontrados a um conjunto destinado à aprendizagem deste contexto. Buscamos impôr fluxos de carga durante a execução do serviço para que as transições automáticas de qualidade de fato ocorressem.

A abordagem utilizada, com a parametrização dos conjuntos de treino através do serviço de visualização de vídeos, atribui ao trabalho um contexto mais moderno levando em consideração os serviços ofertados nos dias de hoje pelas diversas plataformas web do mesmo gênero, com uma visão mais aproximada da extenuação imposta a servidores de grande porte que precisam processar tráfegos de dados substanciais diariamente.

Um fator importante é o porte das máquinas onde os nós da rede Kubernetes são configurados, uma série de computadores Raspberry Pi. Com isso, simulamos um conjunto de servidores funcionando com capacidade computacional menor que o esperado durante a exposição do serviço criado enquanto que um crescente volume de clientes são executados assim que, naturalmente, os componentes onde os servidores se baseiam facilitam a tarefa de atingir níveis de carga maiores. Através do aumento da demanda pelo conteúdo servido, é analisado o comportamento adotado pelos clientes de vídeo à medida que o nivelamento da qualidade de transmissão ocorre.

A plataforma deverá então suportar o funcionamento previsto em diferentes cenários onde demandas provenientes tanto de baixa carga quanto de cargas elevadas deverão ser suportadas e atendidas pela plataforma para possibilitar que conjuntos mais diversificados de dados possam ser capturados e providos ao algoritmo de ML na tentativa de aperfeiçoar sua aprendizagem.

5.2 Resultados obtidos

Com as evidências apresentadas na Seção 4.6, demonstramos que foi possível utilizar a capacidade de adaptação do protocolo DASH dentro do serviço situado na estrutura do Kubernetes e realizar envios de carga simultâneos dentro de períodos ditados no comando de execução do gerador de carga, atingindo um volume de dados maior daquele esperado pelas interfaces de rede usadas e coagindo as características principais do sistema a entrarem em ação. Em suma, temos uma plataforma funcional com componentes que ativamente distribuem o volume de dados recebido ao passo que permite um cliente conectado visualizar um conteúdo de mídia que se autoajuste à condição manifestada pela rede onde os fragmentos de mídia requisitados trafegam, uma vez que o cliente de vídeo suporte o protocolo DASH.

Tendo ambos cliente de vídeo e o sistema Prometheus armazenando métricas a partir do trabalho desempenhado pelo serviço, os conjuntos de dados esperados para serem repassados ao algoritmo de ML já começam a ser formados, acomodando informações das propriedades de áudio e vídeo para cada segundo da atividade empenhada.

5.3 Desafios e obstáculos enfrentados

O projeto apresentou vários conceitos novos para serem estudados e, consigo, seus obstáculos. A pesquisa para implementar um serviço com vídeos no padrão DASH mostrou-se, em tempos, obscura para encontrar informações tanto comuns quanto mais específicas do processo de geração dos vídeos, sendo comum uma falta de clareza quanto a vários comandos, por exemplo, estudados para serem usados nos testes realizados no início do trabalho.

Com isso, foi necessária uma fase marcada por mais tentativa e erro até que os conceitos por trás de certos parâmetros fossem melhor compreendidos. O entendimento dos parâmetros de separação de quadros na segmentação dos vídeos ganhou grande foco nesta fase por ser o que definiria o funcionamento correto do serviço, que necessitou a assimilação dos conceitos de tipos de quadros de imagens e formação de GOPs durante a geração dos novos vídeos.

Os obstáculos encontrados durante o aprendizado do DASH nos impulsionou a criar um tutorial para o até então denominado DASH_NECOS_UFU [Wiki - Tutorial Dash_Necos_UFU] destacando os passos tomados para gerar o ambiente de testes e para obter e utilizar as ferramentas escolhidas na geração dos arquivos necessários, abordando como os parâmetros dos comandos mostrados foram aplicados durante o processo para esclarecer os resultados alcançados.

A fase de construção da rede Kubernetes e utilização de contêineres também

apresentou-se intricada. A documentação apresentada pela plataforma de conteinerização é vasta e bem composta, mas não impediu que barreiras se formassem considerando que foi realizada a construção de uma rede Kubernetes inteiramente nova e falhas poderiam ocorrer como parte do processo.

Durante o estágio inicial de definição dos contêineres da rede surgiu um problema: para cada vídeo que passou pela padronização do protocolo DASH foram gerados três vídeos no total, com isso aumentando o espaço necessário para armazenar o conteúdo existente. Como um dos pontos principais das plataformas de conteinerização é a portabilidade dos serviços implantados, isso se tornou um problema pois os contêineres não seriam mais tão pequenos quanto era esperado. O problema foi contornado ao ser considerada a criação de uma máquina específica para a tarefa de armazenamento do conteúdo de vídeo que teria um servidor acessível pelos nós da rede Kubernetes através do NFS na rede onde se localizam.

À medida que testes foram realizados, mudanças nos vídeos utilizados foram necessárias para adequação da carga a ser utilizada na geração das métricas dos resultados finais. Os resultados dos testes feitos com os primeiros vídeos não foram satisfatórios e alguns dos vídeos demonstravam quedas bruscas da taxa de quadros, por isso os conceitos de geração dos vídeos pelo FFPMEG foram revisitados e selecionados novos valores para os parâmetros de entrada na separação de vídeos conforme novos vídeos foram selecionados para aumentar a carga visível dos testes. A partir deste ponto, foram necessários testes prolongados que garantissem a legitimidade das propriedades geradas para os mesmos vídeos.

5.4 Continuação do projeto

Até então foi construída uma plataforma capaz de hospedar e dar acesso a um serviço multimídia capaz de providenciar vídeos a um cliente com suporte a mídia em formato MPD e, a partir de seu funcionamento, coletamos métricas de funcionamento do serviço em cenários de exaustão dos recursos computacionais dos componentes de uma rede com balanceamento de carga.

O próximo passo é a utilização da coleta de dados na formação de um conjunto de exemplos apropriado para ser alimentado ao algoritmo de aprendizado de máquina. Com o cliente VLC provendo as métricas do conjunto Y e o sistema de monitoramento Prometheus coletando os valores pertinentes ao conjunto X, temos as ferramentas necesárias para montar um *testbed* apto para o quesito de processamento da predição de dados.

Referências

Bodík, P., Griffith, R., Sutton, C. A., Fox, A., Jordan, M. I., Patterson, D. A. (2009). "*Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters*", *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing (HotCloud 2009)*, vol. 9, pp. 12–12, 2009. Citado na página 11.

Bitmovin Inc. - *MPEG-DASH (Dynamic Adaptive Streaming over HTTP)*. Disponível em: <https://bitmovin.com/dynamic-adaptive-streaming-http-mpeg-dash/>. Acesso em: 17 jun. 2018. Citado na página 31.

Bitmovin - *Optimal Adaptive Streaming Formats MPEG-DASH & HLS Segment Length*. Disponível em: <https://bitmovin.com/mpeg-dash-hls-segment-length/>. Acesso em: 26 ago. 2018. Nenhuma citação no texto.

Chiariotti, F. (2015). *Reinforcement learning algorithms for DASH video streaming, University of Padova, Department of Information Engineering (DEI), Master's thesis in Telecommunications Engineering*. Citado na página 9.

Cloud Native Computing Foundation. Disponível em: <https://www.cncf.io/>. Acesso em: 11 jun. 2018. Citado na página 16.

DASHing YouTube: An analysis of using DASH in YouTube video service.. Disponível em: <https://ieeexplore.ieee.org/document/6761273>. Acesso em: 07 nov. 2018. Nenhuma citação no texto.

Lederer, S., Müller, C., Timmerer, C. (2012). "*Dynamic Adaptive Streaming over HTTP Dataset*", *Proceedings of the 3rd Multimedia Systems Conference*. Disponível em: <http://www-itec.uni-klu.ac.at/bib/files/p89-lederer.pdf>. Acesso em: Nenhuma citação no texto.

Docker. Disponível em: <https://www.docker.com/>. Acesso em: 13 mai. 2018. Citado na página 18.

Docker Hub. Disponível em: <https://hub.docker.com/>. Acesso em: 13 mai. 2018. Citado na página 18.

Docker Hub - Apache HTTPD Server Project. Disponível em: <https://hub.docker.com/_/HTTPD/>. Acesso em: 13 mai. 2018. Citado na página 23.

Encoding.com - Encoding Intelligence™. Disponível em: <https://www.encoding.com/mpeg-dash/>. Acesso em: 17 jun. 2018. Citado na página 14.

FFMPEG - *A complete, cross-platform solution to record, convert and stream audio and video.*. Disponível em: <https://www.ffmpeg.org/>. Acesso em: 14 nov. 2017. Citado na página 26.

Giotis, K., Argyropoulos, C., Androulidakis, G., Kalogeras, D., Maglaris, V. (2014). "*Combining OpenFlow and sFlow for an effective and scalable Anomaly Detection and Mitigation mechanism on SDN Environments, Computer Networks*", Volume 62, 7 April 2014, Pages 122-136. Citado na página 12.

GitHub - Repositório VLC versão 3.0.3. Disponível em: <https://github.com/videolan/vlc-3.0>. Acesso em: 13 ago. 2018. Citado na página 34.

Guedes, D., Vieira, L. F. M., Vieira, M. M., Rodrigues, H. e Nunes, R. V. (2012). Redes Definidas por Software: uma abordagem sistêmica para o desenvolvimento de pesquisas em Redes de Computadores, XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2012. Nenhuma citação no texto.

Greenberg, A., Hamilton, J., Maltz, D. A., Patel, P. (2008). "*The Cost of a Cloud: Research Problems in Data Center Networks*", ACM SIGCOMM Computer Communication Review, Microsoft Research, Redmond, WA, USA. Citado na página 8.

Greenberg, A., Lahiri, P., Maltz, D. A., Patel, P., Sengupta, S. (2008). "*Towards a Next Generation Data Center Architecture: Scalability and Commoditization*", Microsoft Research, Redmond, WA, USA. Citado na página 8.

Khan, A., Sun, L. and Ifeachor, E. (2009). "*Content Clustering-based Video Quality Prediction for MPEG4 Video Streaming over Wireless Networks*", Communications, 2009. ICC '09. IEEE International Conference. Citado na página 12.

Linux From Scratch - *List of dependencies for VLC compilation.*. Disponível em: <http://www.linuxfromscratch.org/blfs/view/8.3/multimedia/vlc.html>. Acesso em: 24 jul. 2018. Nenhuma citação no texto.

Kubernetes: *Production-Grade Container Orchestration.*. Disponível em: <https://kubernetes.io/>. Acesso em: 07 jan. 2018. Citado na página 15.

MP4Box - *GPAC multimedia packager.*. Disponível em: <https://gpac.wp.imt.fr/mp4box/>. Acesso em: 15 nov. 2017. Citado na página 30.

MPEG - *Moving Picture Expert Group.*. Disponível em: <https://mpeg.chiariglione.org/>. Acesso em: 07 jan. 2018. Citado na página 12.

ISO/IEC 23009-1:2014. *Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats.*

Disponível em: <https://www.iso.org/standard/65274.html>. Acesso em: 12 ago. 2018. Citado na página 8.

Pasquini, R. e Staddler, R. (2017). "*Learning End-to-end Application QoS from OpenFlow Switch Statistics*", Network Softwarization (NetSoft), 2017 IEEE Conference. Citado na página 11.

Prometheus - Monitoring system & time series database. Disponível em: <https://prometheus.io/>. Acesso em: 21 jun. 2018. Citado na página 35.

Qazi, Z. A., Lee, J., Jin, T., Bellala, G., Arndt, M., Noubir, G. (2013). "*Application-Awareness in SDN*", Proceeding, SIGCOMM '13 Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, Pages 487-488. Citado na página 11.

Repositório VLC - Fork de alteração para log de métricas. Disponível em: <https://github.com/JulianoRFMatos/vlc>. Disponibilizado em novembro de 2018. Citado na página 34.

Stallings, W. (2013). "*Software-Defined Networks and OpenFlow - The Internet Protocol Journal*", Volume 16, No. 1. Nenhuma citação no texto.

Tutorial de geração de vídeos em padrão DASH aplicado. Disponível em: <https://github.com/mcalasans/dash_necos_ufu>. Disponibilizado em agosto de 2018. Citado na página 49.

Twitch Broadcasting Guidelines. Disponível em: <https://stream.twitch.tv/encoding/>. Acesso em: 02 mai. 2018. Citado na página 27.

Verdi, F. L., Rothenberg, C. E., Pasquini, R. e Magalhães, M. F. (2010). Novas Arquiteturas de Data Center para Cloud Computing, XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos - SBRC 2010. Citado na página 8.

Videolan - x264 library. Disponível em: <https://www.videolan.org/developers/x264.html>. Acesso em: 11 jan. 2018. Citado na página 27.

VLC Media Player. Disponível em: <https://www.videolan.org/vlc/index.html>. Acesso em: 11 jan. 2018. Citado na página 34.

Why YouTube & Netflix use MPEG-DASH in HTML5, 2015. Disponível em: <https://bitmovin.com/status-mpeg-dash-today-youtube-netflix-use-html5-beyond/>. Acesso em: 12 set. 2018. Nenhuma citação no texto.

YouTube Help - Recommended upload encoding settings. Disponível em: <https://support.google.com/youtube/answer/1722171?hl=en>. Acesso em: 27 abr. 2018. Citado na página 27.