Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

Arquitetura Distribuída para Transcodificação de Vídeo com Alto Desempenho

Autor:

João Bernardo Vianna S. de M. Oliveira

Orientador:

Prof. Claudio Luis de Amorim, Ph.D.

Orientador:

Prof. Aloysio de Castro Pinto Pedroza, D.Sc.

Examinador:

Prof. Heraldo Luís Silveira de Almeida, D.Sc.

Examinador:

Prof. Sergio Barbosa Villas-Boas, Ph.D.

DEL

Novembro de 2013

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, Bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

AGRADECIMENTO

Agradeço aos meus pais pelo carinho e educação que me proporcionaram e aos meus irmãos e irmãs pela paciência e atenção.

Agradeço aos meus orientadores e também aos amigos do Laboratório de Computação Paralela e Sistemas Móveis que possibilitaram este trabalho e contribuíram para que fosse realizado.

Agradeço também a todos os amigos que, pela companhia, entusiasmo e alegria, fizeram com que esses anos de estudo fossem mais agradáveis.

RESUMO

Este trabalho propõe e avalia um sistema eficiente de transcodificação distribuída de vídeo. Diferente de soluções conhecidas que exigem a modificação do codificador ou restringem o desempenho da solução, apresentamos uma técnica otimizada de busca em vídeo, que permite a criação de segmentos independentes que podem ser transcodificados em paralelo.

Palavras-Chave: vídeo, transcodificação, processamento distribuído, escalabilidade, balanceamento de carga.

ABSTRACT

This work proposes and evaluates an efficient distributed video transcoding. Unlike known solutions that require modification of the encoder or restrict the performance of the solution, we present a optimized technique for video search, allowing the creation of independent segments that can be transcoded in parallel.

Keywords: video, transcoding, distributed processing, scalability, load balancing.

SIGLAS

CPU – *Central processing unit*.

GOP – *Group of Pictures*.

HD – *High Definition*.

HTTP – *Hyper Text Transfer Protocol*.

MPEG – Motion Picture Experts Group.

NFS – *Network File System*.

RGB – *Red Green Blue*.

Sumário

Sumário **vii**

Lista de Figuras **x**

Lista de Tabelas **xii**

1 Introdução **1**

1.1 Tema 1

1.2 Delimitação 1

1.3 Justificativa 2

1.4 Objetivos 3

1.5 Metodologia 3

1.6 Organização 5

2 Vídeo Digital **6**

2.1 Compressão 6

2.2 Formas de Compressão 7

2.2.1 Codificação *Intra-Frame* 8

2.2.2 Codificação Temporal 10

2.3 Formatos 13

2.3.1 *Codecs* 13

2.3.2 Contêineres 15

2.4 Serviços Web 16

2.5 Transcodificação de Vídeo 18

3 Transcodificação de Vídeo Distribuída **20**

3.1 Processo 21

3.1.1 Distribuição de tarefas 22
3.2 Segmentação de Vídeo por Busca 23
3.2.1 Busca Otimizada 24
3.3 Adequação do uso de recursos de CPU – Processos simultâneos . . . 25
3.4 Balanceamento de Carga 26
3.5 Transcodificação do Áudio 27

4 Avaliação Experimental e Resultados 29
4.1 Detalhes dos Vídeos Gerados 30
4.2 Experimento Preliminar – Vídeo Curto 32
4.2.1 Avaliação da Transcodificação Convencional 33
4.2.2 Distribuição de Segmentos pelo Cluster 34
4.2.3 Threads e Processos 35
4.2.4 Balanceamento de Carga 37
4.2.5 Aceleração da Técnica e Tempos de execução 41
4.2.6 Transcodificação da faixa de Áudio 43
4.3 Vídeo Longo – BBC 43
4.3.1 Balanceamento de Carga 44
4.3.2 Aceleração da Técnica e Tempos de execução 47
4.4 Video Médio – Sintel 49
4.4.1 Balanceamento de Carga 50
4.4.2 Aceleração da Técnica e Tempos de execução 52
4.5 Resultados e Trabalhos Relacionados 54

5 Conclusão e Trabalhos Futuros 56
5.1 Conclusão 56
5.2 Trabalhos Futuros 57
5.2.1 Análise de GOP e Variação dinâmica do tamanho dos Segmentos 58
5.2.2 Cluster heterogêneo 59
5.2.3 Sistemas de Arquivos Distribuído e Balanceamento de Carga com Roubo de tarefa 59
5.2.4 Análise de Qualidade/Taxa de Bits 60

5.2.5 Transcodificação de Áudio 61

Referências Bibliográficas **62**

A Comandos do FFmpeg **64**

A.1 Análise de Metadados 65

A.2 Extração do Áudio 65

A.3 Segmentação e Codificação do Segmento 66

A.4 Concatenação dos Segmentos do Vídeo e Áudio 68

Lista de Figuras

2.1 Exemplo de GOP aberto com quadros I, P e B. 12

3.1 Etapas da transcodificação distribuída com um servidor mestre e 4 servidores de transcodificação 23
3.2 Segmentação do arquivo de vídeo. 24

4.1 Tempo da Transcodificação em 1 Nó com 8 Cores – CCD. 33
4.2 Tempo da Transcodificação Distribuída – 4 Segmentos em 4 Nós com 16 Threads por Segmento – CCD. 34
4.3 Tempo da Transcodificação Distribuída vs. No. Segmentos – Cluster com 4 Nós – CCD. 36
4.4 Tempo da Transcodificação Distribuída – 32 Segmentos em 4 Nós com 8 threads por segmento – CCD. 36
4.5 Escalonamento de tarefas para 2 nós de transcodificação com 2 controladores de pedidos e um total de 12 segmentos de vídeo (tarefas a serem executadas). 38
4.6 Variação na quantidade de segmentos de vídeo com transcodificação simultânea de 32 segmentos em 4 máquinas – CCD. 39
4.7 Aceleração real da técnica de balanceamento (com reta de aceleração linear teórico para comparação) – CCD. 41
4.8 Tempos de Transcodificação com Balanceamento de Carga – CCD. 42
4.9 Variação na quantidade de segmentos de vídeo com transcodificação simultânea de 32 segmentos em 4 máquinas (32 a 384 segmentos) – BBC. 45
4.10 Variação na quantidade de segmentos de vídeo com transcodificação simultânea de 32 segmentos em 4 máquinas (512 a 1024 segmentos) – BBC. 46
4.11 Aceleração real da técnica de balanceamento (com reta de aceleração linear teórico para comparação) - BBC. 48

4.12 Tempos de Transcodificação com Balanceamento de Carga comparado ao modelo puramente multithread – BBC. 48
4.13 Variação na quantidade de segmentos de vídeo com transcodificação simultânea de 32 segmentos em 4 máquinas (32 a 160 segmentos) – Sintel. 50
4.14 Variação na quantidade de segmentos de vídeo com transcodificação simultânea de 32 segmentos em 4 máquinas (192 a 320 segmentos) – Sintel. 51
4.15 Aceleração real da técnica de balanceamento (com reta de aceleração linear teórico para comparação) – Sintel. 52
4.16 Tempos de Transcodificação com Balanceamento de Carga comparado ao modelo puramente multithread – Sintel. 53

Lista de Tabelas

4.1 Ambiente Experimental - Nó Computacional 29
4.2 Ambiente Experimental - Detalhes do vídeo gerado 30
4.3 Ambiente Experimental - CCD 32
4.4 Componentes do Tempo Total de Transcodificação da Figura 4.2 34
4.5 Componentes do Tempo Total de Transcodificação da Figura 4.4 37
4.6 Ambiente Experimental - BBC 44
4.7 Ambiente Experimental – Sintel 49

A.1 Descrição do comando de análise de metadados. 65
A.2 Descrição do comando de extração de áudio. 66
A.3 Descrição do comando de segmentação. 67
A.4 Descrição do comando de concatenação. 68

Capítulo 1

Introdução

1.1 Tema

O Tema do estudo é o desenvolvimento de técnicas para transcodificação paralela e distribuída de arquivos de vídeo entre dois formatos com uma ou mais taxas de bits (*bitrate*) diferentes. Sabendo que processamento de vídeo é notável pelo seu alto consumo de recursos computacionais, pretende-se reduzir o tempo total de conversão através de novas técnicas que aumentem a eficiência desse processamento em um computador e expandido-o para diversos computadores interconectados.

1.2 Delimitação

Para diminuir o tempo de codificação, busca-se uma solução custo-efetiva em software, comparável com soluções em hardware, que distribua o processamento por vários núcleos de um processador e/ou vários computadores de um cluster.

Este projeto visa beneficiar as recentes aplicações multimídia na Internet considerando principalmente o modelo de transmissão adaptativa de vídeo.

Para este modo adaptativo de transmissão, um servidor mantém várias versões do mesmo vídeo, codificados com diferentes taxas de reprodução e níveis de qualidade. Um dispositivo na rede pode solicitar a versão que se adequar melhor à transmissão de acordo com as condições encontradas na sua conexão com o servidor.

Uma nítida desvantagem do modo adaptativo de transmissão é o custo de codificação do vídeo em várias taxas. A operação de codificação do vídeo com qualidade requer muito tempo de processamento, e este tempo aumenta de acordo com o aumento na qualidade exigida.

Desta forma, é necessário investigar meios de tratar a conversão de um vídeo, em particular, novas formas de distribuir eficazmente a carga de transcodificação utilizando múltiplos processadores.

1.3 Justificativa

O tempo gasto no processo de transcodificação para diversas taxas de bits em sistemas *web* de vídeo sob demanda como YouTube e Netflix, limitam a disponibilidade de vídeos para seus usuários. Do momento em que o sistema recebe um vídeo novo até o momento em que ele estará pronto para ser acessado, pode ser gasto mais tempo que algumas vezes a duração do mesmo e isto tem pouco espaço para melhorias no modelo atual de transcodificação.

Convencionalmente, as melhorias que são executadas consistem apenas em distribuir para uma rede de servidores as tarefas de produção de diversas taxas de vídeo onde cada servidor irá produzir uma delas. O paralelismo dessa técnica está

diretamente limitado com o número de taxas produzidas que chegam a variar de 3 a 15 no YouTube e no Netflix, por exemplo.

Por consequência, o único espaço para crescimento está no uso de servidores mais robustos, sendo que essa escalabilidade vertical está limitada no processamento de múltiplas threads de codificação, como será visto no capítulo 3. O modelo aqui proposto permite divisão do trabalho para um grande número de servidores, dependendo do vídeo a ser codificado e das máquinas utilizadas. Isto tudo ainda é munido de um melhor uso de recursos de hardware numa mesma máquina, com uso de multiprocessamento.

1.4 Objetivos

O objetivo geral é propor e desenvolver um sistema capaz de otimizar o uso de recursos de hardware, como processador e memória, além tirar proveito da possibilidade de se alocar uma quantidade grande de máquinas trabalhando em conjunto a fim de reduzir o tempo total de transcodificação de vídeo. Dessa forma, pretende-se garantir um serviço melhor de video sob demanda.

1.5 Metodologia

Este trabalho utiliza um sistema de segmentação de vídeo com base em busca de quadros-chave sem executar cortes no arquivo a ser processado e, com isso, possibilitar a distribuição de trabalho mais efetivamente e com pouco pré-processamento. Este trabalho inicial em cima do vídeo se limita apenas à aquisição de metadados e escolha dos pontos de corte onde cada processo vai trabalhar,

definidos de acordo com o número de máquinas disponíveis para execução da tarefa.

O processo de segmentação estará limitado a certos tipos de *codecs* de vídeo que permitam que esse sistema de busca seja feito rapidamente e com precisão, sendo esse um dos pontos de estudo para que a execução seja confiável [1].

Após a implementação do processo descrito para executar a segmentação, é necessário executar a distribuição de carga entre os diversos nós de codificação disponíveis, sendo que cada servidor poderá processar simultaneamente mais de um segmento de vídeo para aproveitar melhor seus recursos.

Além de segmentar o vídeo, o processo ainda necessita de uma etapa final de junção das partes criadas, onde isso é feito por meio de concatenação de arquivos e empacotamento com um contêiner, neste caso, MP4.

Os recursos de hardware ainda serão otimizados com análise de métricas de utilização dos mesmos e tentativas de ocupar ao máximo o poder de processamento da máquina. Isto deve ser feito com um balanceamento mais efetivo de distribuição de processos, tendo por base o uso de segmentos menores que os da etapa anterior e filas de espera para ajudar a igualar o trabalho entre as diversas máquinas. Isso é feito sem que algum servidor seja obrigado a processar muito mais dados que outros [2].

Todo o processo foi, por fim, testado com a conversão de diversos vídeos com os *codecs* mais comuns que estejam disponíveis no mercado e com diferentes resoluções e taxas de bits de entrada. A saída do sistema sempre consiste em um ou mais vídeos com o *codec* H.264 (Advanced Video Coding – AVC – MPEG-4 Part 10)[3] e com o áudio codificado em AAC (Advanced Audio Coding – MPEG-4 Part 3)[4] por garantirem alta compatibilidade com os *players* do mercado e, no caso do H.264, produzir vídeos com melhor qualidade que outros *codecs* disponíveis utilizando a mesma taxa de bits [5].

A produção de diversas taxas de vídeo será, neste caso, feita simultaneamente numa mesma máquina para cada segmento pois isso garante que para cada trecho somente será executada uma etapa de decodificação do vídeo antes de ser codificado.

1.6 Organização

No capítulo 2 serão apresentados os detalhes da codificação do vídeo digital, tendo por foco os aspectos da compressão. O processo de transcodificação, também mostrado nesse capítulo, será utilizado como base para criar o modelo distribuído, aproveitando-se ao máximo das técnicas e procedimentos disponíveis no mercado.

O capítulo 3 explica a abordagem distribuída dada à transcodificação de vídeo assim como o modelo de balanceamento de carga aplicado para garantir melhor utilização de recursos computacionais disponíveis. Também é visto nesse capítulo o tratamento dado ao fluxo de áudio para fazer com que a transcodificação possa ser aplicada de forma concreta em arquivos de vídeo.

Os resultados experimentais são apresentados no capítulo 4. Nele serão vistos na prática os efeitos decorrentes da divisão de trabalho entre diversas máquinas simultaneamente. Este capítulo irá fazer um comparativo entre o modelo sequencial vigente e o modelo aqui proposto que executa a transcodificação com alto nível de paralelismo.

Por fim, no capítulo 5 serão apresentados a conclusão e alguns possíveis trabalhos futuros para ampliar o conhecimento adquirido por este projeto e também possibilitar melhorias na técnica.

Capítulo 2

Vídeo Digital

A codificação de vídeo vem sendo um campo muito estudado e de grande valor para a indústria das últimas décadas. Isto se dá ao fato de que se tornou possível armazenar e transmitir digitalmente conteúdo de vídeo, e banda de rede e disco sempre foram recursos bastante limitados.

Para suprir a necessidade de poupar recursos, o vídeo digital precisa ser armazenado com algum tipo de compressão.

Este capítulo expõe os detalhes da codificação do vídeo digital, suas formas de armazenamento e serviços web que o utilizam.

2.1 Compressão

Um vídeo digital é definido por uma sequência de quadros ou frames de imagens que são exibidos de forma contínua numa determinada taxa. Cada quadro tem altura e largura definidos (resolução) e é formado por pixels que, por sua vez, guardam a informação pontual de cor.

Um fluxo de vídeo possivelmente está acompanhado de um fluxo de áudio, sendo estes empacotados conjuntamente para serem exibidos sincronizadamente.

Este empacotamento pode conter um ou mais fluxos de áudio e vídeo, mas, por simplicidade, vamos tratar apenas do caso onde há no máximo um de cada.

Todavia, para armazenar e transmitir este tipo de mídia, é necessário algum tipo de compressão de forma a viabilizar o uso. Como este projeto se limita a técnicas de compressão de vídeo, iremos desconsiderar a compressão do áudio inicialmente.

Para entender a necessidade da compressão do vídeo, podemos tomar o exemplo de um vídeo com resolução *Full HD* (1920×1080 pixels) com 8 bits para cada uma das 3 cores primárias RGB, sendo exibido a 30 quadros por segundo.

$$1920 \times 1080 \times 8 \times 3 \times 30 \approx 1.39 Gbps$$

Estando descomprimido, este fluxo exige uma banda de transmissão de 1,39 gigabits por segundo, sendo este valor maior do que a capacidade de muitos *switches* ou roteadores de rede. Um *switch* comercial para consumo doméstico permite taxas de transferência de 1Gbps ou 100Mbps, enquanto roteadores sem fio desta mesma classe utilizam no máximo 300Mbps ou 150Mbps. Portanto, definitivamente não é viável transmitir este tipo de mídia sem fazer um processamento prévio.

2.2 Formas de Compressão

Para comprimir um fluxo de vídeo, leva-se em consideração as capacidades da visão humana: as cores que ela enxerga, as diferenças de luminosidade que é capaz de distinguir, entre outros fatores.

Uma característica comum da visão humana é o fato dela se adaptar a diferentes níveis de estímulo, sendo sensível apenas às variações temporais e espaciais diretas

destes níveis de estímulo [6]. Assim, a codificação de imagens em computador pode levar em consideração a razão entre a luminância de um artefato com o estímulo gerado pelo plano de fundo, chamado contraste [7].

Além dos níveis de estímulo, a visão também responde a certas frequências do espectro eletromagnético que, ao serem combinadas, representam as diferentes informações cromáticas que são captadas, chegando a milhões de cores que podem ser individualmente enxergadas.

A percepção das cores, entretanto, é mais complexa que apenas a frequência da luz, dependendo de fatores das células cone que compõem a retina do observador e das conexões nervosas que levam a informação ao cérebro. Desta forma, a visão permite que degradação na qualidade de uma imagem possa passar despercebida, possibilitando que a compressão digital feita apresente perdas e ainda seja inteligível [8].

A codificação pode ser feita em cada quadro do vídeo (Intra-Frame) ou levando em consideração a diferença entre quadros consecutivos (Inter-Frame).

2.2.1 Codificação *Intra-Frame*

Cada quadro do vídeo é tratado e codificado separadamente, contendo toda a informação necessária para ser decodificado e exibido. Quadros de vídeo armazenados desta forma são independentes dos quadros anteriores e futuros, podendo posteriormente ser utilizados como base para codificação Inter-Frame.

Esquema de Cores

Em vez de armazenar 8 bits para cada uma das 3 cores primárias RGB, resultando em 24 bits por pixel, algumas codificações, como o H.264, armazenam informações de luminância e de crominância do espaço de cor YUV[8, Cap 2]. Isto é vantajoso pois o olho humano é mais sensível às variações de intensidade

luminosa que à variação cromática de mesma intensidade, sendo assim, possível reduzir a quantidade de bits para essa parte do espectro.

O espaço YUV (também referido por YCbCr) consiste em na dimensão de luminância Y e duas dimensões de crominância U e V. Historicamente, este sistema surgiu para manter a compatibilidade da transmissão de TV analógica que antes era monocromática (apenas contendo informação de luminância) para o esquema de TV à cores.

No vídeo digital, o armazenamento de informação no espaço YUV se dá com uma técnica de subamostragem de cor, onde normalmente todos os pixels tem sua informação de luminosidade, mas alguns só tem parte da informação de cor, como por exemplo o formato YUV422, onde são utilizados 4 bits para luminância de cada pixel e os canais U e V são amostrados a cada 2 pixels. Ainda é comum o format YUV420p, onde a cada 4 pixels são utilizados 6 bytes (48 bits) de informação, resultando em metade das resoluções horizontal e vertical para crominância.

Desta forma, utilizando um esquema de cores adequado, é possível reduzir consideravelmente a informação a ser armazenada sem que exista grande diferença visual para o observador, já que, diferentemente do espaço RGB onde as três cores são igualmente importantes, no YUV, existem distinções perceptuais.

Compressão Espacial

Mesmo utilizando o espaço de cores YUV para codificação de vídeo, ainda seriam necessários cerca de 710 megabits por segundo para a transmissão do mesmo vídeo *Full HD* anterior com 30 quadros por segundo. Isto continua sendo impraticável para transmissão web ou armazenamento em mídias físicas. Desta forma, inicialmente é possível aplicar técnicas de compressão espacial para reduzir mais o tamanho ocupado por cada frame de vídeo.

A compressão de dados vem à custa de poder computacional extra tanto para codificar como decodificar o vídeo, sendo este um fator possivelmente limitante para gerar ou reproduzir vídeo em alguns dispositivos.

Por mais que a compressão ou compactação de dados seja, por definição, um processo sem perdas, já sabemos que o olho humano tolera com grande facilidade pequenas diferenças ou imperfeições nas cores, sendo isso excelente para utilizar aproximações nas codificações de imagens.

Muitas das técnicas de compactação de dados binários utilizam da existência de padrões de repetição nas sequências de bytes, como por exemplo no caso de imagens, existe uma alta correlação horizontal e vertical entre os pixels[9, Cap 14], o que resulta em uma compressão modesta (em torno de 50%) com uso de algoritmos universais de compactação como RLE[I] e LZW[II].

Entretanto, é possível conseguir taxas de compressão muito mais altas ao explorar as características da percepção humana em relação a redundâncias, podendo ser espaciais devido a áreas como luminância ou crominância similares ou temporais, quando quadros consecutivos apresentam a mesma redundância espacial [10]. Este último caso será abordado no próximo tópico.

2.2.2 Codificação Temporal

A compressão temporal é possível pela existência de similaridades entre áreas de um vídeo entre quadros consecutivos.

Tipos de Quadros

Os codificadores temporais trazem uma distinção entre os tipos de quadro de um vídeo de acordo com a forma com que ele se relaciona com outros quadros. Basicamente, existem 3 tipos de quadros: **I**, **P** e **B**. Em codificadores modernos, podem ainda existir variações desses quadros para auxiliar o chaveamento entre fluxos de vídeo de taxas diferentes.

[I] Run-length encoding.

[II] Lempel–Ziv–Welch algorithm.

Um quadro do tipo **I**[III] traz a informação completa, apenas comprimida espacialmente. Ele não depende de nenhum outro para ser exibido e pode ser a base para outros quadros.

Os quadros dos tipos **P**[IV] e **B**[V] exibem compressão temporal e necessitam de outro(s) quadro(s) para serem decodificados e exibidos. Enquanto quadros **P** somente podem depender de quadros anteriores, quadros do tipo **B** são bidirecionais e dependem também de quadros seguintes para melhorar a codificação e reduzir a quantidade de bytes necessária para armazenar informação.

Ambos os tipos de quadros **P** e **B** dividem a imagem em pequenos blocos e utilizam a técnica de compensação de movimento para predizer a posição destes blocos em relação ao(s) quadro(s) tomados por referência. Desta forma, estes quadros normalmente são armazenados como vetores de movimento (Motion Vectors – MV) e coeficientes de compensação[9, Cap 40]. Para evitar propagação de erro, alguns codificadores não permitem que quadros **B** sejam utilizados como referência para outros.

Além de exibir blocos com predição de movimento, quadros **P** e **B** também podem conter blocos com codificação intra-quadro: a mesma dos quadros **I**. Isto acontece quando é necessário codificar um pedaço de imagem que não apareceu nos quadros utilizados por referência.

O espaçamento dos quadros **I** define os pontos onde o vídeo pode começar a ser exibido, portanto não podem ser muito grandes para aplicações web como videoconferência, pois o usuário teria que esperar o próximo quadro **I** para iniciar a exibição.

[III] **I** – *Intra-Frame coded picture* – Figura com codificação intra-quadro.

[IV] **P** – *Predicted picture* – Figura com predição.

[V] **B** – *Bi-directional predicted picture* – Figura com predição bidirecional.

GOP – Group of Pictures

O GOP (Grupo de Figuras) é uma unidade utilizada em codificação de vídeo temporal que define o espaçamento de quadros **I**, sendo iniciado por um destes quadros e seguido por um ou mais quadros **P** e **B**.

No evento de quadros pertencentes a um GOP que façam referência a um quadro de outro, este GOP é denominado aberto. No caso de GOP fechado, todas as referências são feitas a quadros do mesmo GOP. A figura 2.1 apresenta quadros de um fluxo de vídeo teórico onde os primeiros 6 quadros fazem parte de um GOP aberto pois há referências a um quadro I posterior pelos quadros 5 e 6.

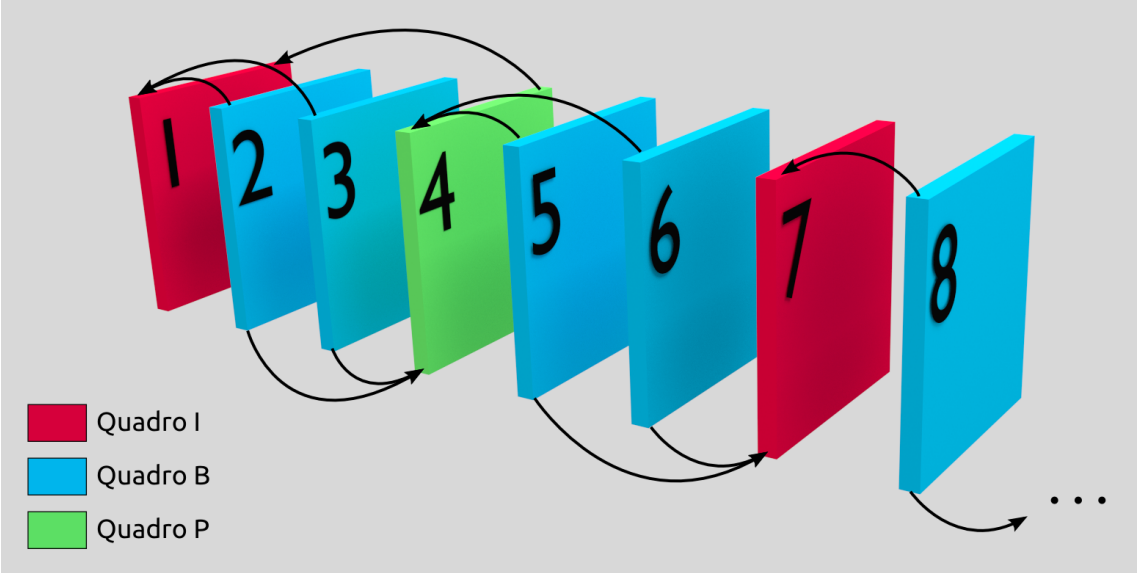

Figura 2.1: Exemplo de GOP aberto com quadros I, P e B.

Reordenação: Para a transmissão de um vídeo, o uso de quadros B pode não parecer muito interessante à primeira vista, pois este obrigaria o envio de um quadro que não pode ser decodificado até que o quadro I ou P ao qual ele está referenciando chegue. Entretanto, é comum os codificadores reordenarem os quadros no momento de emitir o vídeo para que os quadros B já possam ser decodificados no momento em que forem recebidos. Ainda no exemplo da figura 2.1, o quadro 4 seria o segundo armazenado, enquanto os quadros 2 e 3 seriam respectivamente o terceiro e o quarto. Essa mesma inversão acontecerá com os quadros 5, 6 e 7; mesmo este último sendo pertencente a outro GOP.

Key-Frames: Alguns *codecs* fazem distinção entre *Key-Frame* e Quadro **I**, onde um *Key-Frame* é um Quadro **I** que inicia um GOP. Ou seja, é possível existirem quadros **I** que permitam referências a quadros anteriores a ele por quadros posteriores. Apesar disso, neste trabalho, todas codificações geradas obrigam os quadros **I** a serem também *key-frames* e que cada GOP tenha apenas um quadro **I**.

Isto é interessante para que os dois termos sejam intercambiáveis e, mesmo que existam quadros **I** que não sejam *key-frames*, estes podem ser considerados na estrutura do GOP como quadros **P** contendo apenas compressão intra-quadro e nenhuma referência a outro quadro.

A estrutura do GOP e a sua organização serão de grande importância para o desenvolvimento da técnica de distribuição de processos de vídeo, descrita no capítulo 3.

2.3 Formatos

Sobre formatos de vídeo, aqui serão abordados os *codecs* — Software para codificar e decodificar vídeo; os contêineres — Padrões de armazenamento que podem abrigar um ou mais fluxos de vídeo ou áudio; e os formatos para transmissão de vídeo na Internet.

2.3.1 *Codecs*

No mercado, existe uma série de *codecs* de vídeo disponíveis para diversos nichos. No campo de vídeo sob demanda, utiliza-se exclusivamente *codecs* com compressão temporal para reduzir quantidade de armazenamento e banda de rede utilizada, já que estes tendem várias vezes mais eficientes que outros que apenas executem compressão espacial. Podemos citar alguns destes *codecs*:

H.264/AVC Quadros I, P e B e compressão com ou sem perdas.

VP8 Quadros I e P e compressão com perdas.

MJPEG Somente quadros I e compressão com perdas.

Este projeto tem em vista o uso dos codecs mais modernos do mercado, que são o H.264 e VP8, pois ambos permitem a utilização de busca de quadros de forma indexada durante a decodificação (necessário para a implementação deste projeto, como será descrito em 3.2.1), além de serem os mais difundidos e utilizados no mercado atual.

Anteriores ao H.264 estão o H.263 e H.262/MPEG-2, utilizados ainda em aplicações *Flash* de Videoconferência e TV Digital/DVD, respectivamente. No entanto, estes estão sendo substituídos pelos padrões mais modernos. O mesmo já aconteceu com os padrões VP6 e VP7 em relação ao VP8.

O codec MJPEG foi aqui ainda citado por ser um comumente utilizado para evitar efeitos da compressão temporal, contendo apenas quadros I. Contudo, a arquitetura descrita neste documento irá permitir descartar o uso de MJPEG para transcodificação distribuída, como ainda é feito em alguns casos[2].

Estado da arte

Os codecs a seguir vão definir a próxima geração de vídeo digital, trazendo diversas melhorias, relacionadas principalmente a melhores taxas de compressão. No momento desta pesquisa, estes codecs ainda não estavam disponíveis para o público geral, o que dificultaria a avaliação da arquitetura proposta neste documento.

Entretanto, nada impede que estes sejam utilizados num futuro próximo, com as vantagens do processamento distribuído que será apresentado e discutida nos próximos capítulos.

H.265/HEVC Quadros I, P e B e compressão com ou sem perdas. Suporte a resoluções até 8K (7680×4320).

VP9 Quadros I e P e compressão com perdas.

Ambos formatos permitem reduzir em até 50% a taxa de bits de um fluxo de vídeo, quando comparados ao seus predecessores, com qualidade similar.

2.3.2 Contêineres

Para exibir um videoclipe, somente os dados do vídeo codificado podem não ser suficientes. Isto acontece pois, muitas vezes, o vídeo está associado a uma trilha de áudio, legendas e outros metadados.

Um contêiner serve para sincronizar os diferentes fluxos que juntos compõem a obra que será assistida pelos usuários, funcionando como um receptáculo para a coexistência destas informações.

Para vídeo, existem diversos formatos que podem ser utilizados para armazenar vídeo, sendo alguns específicos para poucos *codecs* e outros mais abrangentes.

Entre os contêineres mais usuais, podemos citar:

AVI, MP4, FLV, MKV, WEBM

Para esse projeto, será aplicado um contêiner MP4 ao vídeo gerado, por este ser um dos mais populares em sistemas de vídeo sob demanda para uso com H.264, sendo reconhecido pela maioria dos dispositivos de reprodução de vídeo.

2.4 Serviços Web

A técnica desenvolvida neste projeto tem como alvo primário a aplicação de vídeo sob demanda. Serviços web de vídeo sob demanda necessitam de técnicas efetivas de transmissão de dados para suprir os requisitos dos clientes para que o conteúdo seja exibido com qualidade visual e qualidade de serviço. Alguns desses serviços mais conhecidos são YouTube[VI], Netflix[VII] e Vimeo[VIII].

Este tipo de aplicação, nos últimos anos, vem utilizando o protocolo HTTP por sua simplicidade ao tratar o conteúdo como arquivos. Entretanto, outras tecnologias e protocolos também são utilizados e podem influenciar como o vídeo deve ser codificado.

HTTP

Transmitir um arquivo de vídeo por HTTP, ou download progressivo, é um procedimento bastante simples e eficaz, mas não permite que o usuário escolha qual parte do vídeo assistir. Para tal funcionalidade, vários servidores Web implementam a funcionalidade de *byte-range* e/ou *media-range*, disponíveis com o protocolo HTTP 1.1[11]. Esta técnica é conhecida como *pseudo-streaming*.

Além do envio convencional descrito, existem outras técnicas baseadas em HTTP para transmitir vídeo sendo elas muitas vezes utilizadas para transmissões ao vivo, mas que também podem trazer vantagens para vídeo sob demanda, como: HLS[IX], Smooth Streaming[X], DASH[XI]. Esses modelos de transmissão facilitam o uso de múltiplas taxas de vídeo, para se adaptar à capacidade da rede do usuário e do servidor.

[VI] `http://www.youtube.com`

[VII] `http://www.netflix.com`

[VIII] `http://www.vimeo.com`

[IX] Apple HTTP Live Streaming.

[X] Microsoft Smooth Streaming.

[XI] MPEG Dynamic Adaptive Streaming over HTTP.

No caso do YouTube, o seu *player* de vídeo favorece o uso de DASH para vídeo sob demanda quando o navegador web possuir Adobe Flash ou implementar o protocolo nativamente. Este fato faz com que o uso de banda de rede seja reduzido pois somente as partes do vídeo sendo assistidas são descarregadas do servidor, não permitindo que todo o conteúdo seja armazenado no cliente. No caso do Netflix o seu *player* utiliza Smooth Streaming para também evitar desperdício de banda de rede.

Essas técnicas muitas vezes favorecem o uso de H.263 e H.264 com contêineres FLV e MP4 para a transmissão, mas ainda existe espaço para o uso de VP8 com WEBM.

RTMP

Para vídeo sob demanda, o uso de RTMP[XII] vem sendo reduzido pelas facilidades oferecidas pelo HTTP, no entanto, para vídeo ao vivo, ainda há grande uso do protocolo para gerar o envio do fluxo de vídeo que, após transcodificado pode ser redistribuído em um série de formatos, incluindo o próprio RTMP.

Assim como o HTTP, o RTMP funciona em cima de TCP, mas por utilizar portas diferentes, este pode acabar sendo bloqueado dentro de *firewalls* de algumas corporações. Para contornar essa limitação, alguns optam por encapsular o fluxo RTMP em uma transmissão HTTP e assim reduzir as chances de bloqueio.

O fato do RTMP ser concebido como um protocolo fechado e com uma especificação limitada em alguns aspectos faz com que sua adesão seja restringida em diversos ambientes, como navegadores de internet que não utilizem o Adobe Flash.

Entre os sistemas web citados, YouTube e Vimeo utilizam RTMP para certas formas de conteúdo em tempo real. O primeiro o utiliza para publicação de conteúdo ao vivo por parte dos usuários, onde este conteúdo será transcodificado e reenvi-

[XII] Real Time Message Protocol - Adobe - `http://www.adobe.com/devnet/rtmp.html`

ado por HLS. No caso do Vimeo, seu serviço de edição de conteúdo em tempo real[XIII] utiliza do protocolo para permitir que a aplicação responda com agilidade.

2.5 Transcodificação de Vídeo

A transcodificação de vídeo é composta por uma fase de decodificação do vídeo inicial, seguida pela codificação no formato escolhido. Estas duas etapas podem ocorrer de forma paralela, ou seja, enquanto uma parte do programa busca dados armazenados e decodifica, outra parte se encarrega de gerar o vídeo final.

A etapa de decodificação consiste em eliminar qualquer compressão que tenha sido aplicada ao vídeo, de forma que agora este seja representado por planos de cor (RGB) ou de crominância/luminância (YUV), podendo ter mais outras informações, como um plano extra contendo o canal *alpha* para transparência. À medida que os quadros vão sendo decodificados, estes serão enviados para um *buffer* ou outra estrutura de dados em memória que funcione como uma fila FIFO[XIV], para assim serem tratados.

Antes de acontecer a codificação de vídeo, o vídeo decodificado pode passar por um processo de filtragem onde diversas características suas podem ser alteradas, como resolução, taxa de quadros por segundo, tonalidade de cores etc.

Ao passo que os quadros de vídeo vão sendo decodificados e filtrados, eles vão sendo enviados para a etapa de codificação, onde o software aplica técnicas de compressão condizentes com o *codec* especificado e empacota o vídeo em um contêiner, podendo também multiplexar pacotes de áudio, legendas e metadados durante o progresso.

[XIII] `http://www.vimeo.com/enhancer`

[XIV] First in, First out - Elemetos são retirados da estrutura na ordem em que são inseridos.

É interessante notar que o áudio e outros possíveis fluxos também podem ser codificados em diversos formatos, mas o empenho deste trabalho está em mudar e aprimorar a mais complexa e custosa dessas conversões, que é a de vídeo.

Capítulo 3

Transcodificação de Vídeo Distribuída

No capítulo anterior, foi apresentado como o vídeo digital é formado e como a codificação temporal afeta a decodificação e isso é um fator a ser transcendido para possibilitar a paralelização da técnica de transcodificar fluxos de vídeo.

Neste momento será descrito o processo de segmentação e distribuição do arquivo a ser transcodificado e, também, o modo como a carga de trabalho pode ser balanceada para execução eficiente em diversas máquinas interconectadas. O procedimento formulado depende da obtenção de informação sobre os quadros completos de vídeo, onde não são permitidas referências a outros anteriores, para dar início ao processo de decodificação e, posteriormente, inciar a codificação somente da informação necessária para uma determinada parte do vídeo, aqui denominada de segmento.

Com diversos segmentos de um vídeo, é possível então distribuir tarefas por diversos processadores, podendo estes estarem em mais de uma máquina para serem executados.

3.1 Processo

A codificação de um vídeo com compressão temporal (como MPEG2, H.264 e VP6, entre outros) utiliza diferentes tipos de quadros. Como visto, na especificação do H.264, por exemplo, existem os quadros I, P e B, dos quais o primeiro é um quadro completo e os outros dois utilizam referências de quadros próximos para formar uma imagem completa.

Em um segmento contendo apenas GOP fechados, todos os quadros podem ser decodificados independentemente. Ao contrário, se o primeiro GOP de um segmento for um GOP aberto, o GOP anterior é necessário para decodificar quadros do tipo B no segmento. Por esta razão, um vídeo H.264 não pode ter um segmento transcodificado (uma decodificação, seguida de codificação em outra taxa ou formato) de forma independente se não possuir todas as referências necessária aos quadros B contidos nele.

As soluções de transcodificação distribuída existentes modificam o codificador para garantir que as referências de quadros P e B estejam contidas em grupos de quadros bem definidos ou realizam a decodificação completa do vídeo - para retirar a compressão temporal - e só após segmentam o vídeo para distribuir a codificação.

No sistema de transcodificação aqui proposto, um arquivo de vídeo pode ser segmentado em quantas divisões forem necessárias e estas são distribuídas entre nós remotos para a transcodificação. Mas a segmentação não divide o vídeo em arquivos menores, apenas cria os parâmetros necessários para gerar os segmentos. Todos os nós remotos possuem o vídeo original completo, assim o processo de transcodificação tem acesso às referências necessárias ao decodificar cada segmento. Assim, evitamos a perda da informação ao dividir o vídeo em segmentos.

No processo de transcodificação, de cada segmento indicado é gerado um novo arquivo. No final, no nó mestre, estes arquivos serão integrados, junto com o áu-

dio, para gerar o novo vídeo. A segmentação e a correspondente integração dos segmentos devem ser simples e rápidas para que não aumentem desnecessariamente o tempo de transcodificação do vídeo.

A faixa de áudio não pode ser segmentada. O áudio por conter dados entre quadros consecutivos de vídeo, apresentam artefatos quando segmentado, ocasionando perda de informação. Assim, a transcodificação do áudio se dá sequencialmente e é independente da codificação do vídeo.

Observação Para executar a transcodificação de fluxos de áudio e vídeo, será utilizado o programa FFmpeg controlado por um software próprio, caracterizado pelas informações deste capítulo. O uso desse software, como será melhor explicado no capítulo 4, se deve a facilidade de implementação das idiossincrasias dessa técnica. Para o leitor interessado, o anexo A traz os comandos utilizados nesse programa para executar as etapas da transcodificação distribuída.

3.1.1 Distribuição de tarefas

A segmentação é iniciada no nó mestre, com a análise dos metadados do vídeo. Nesta etapa é selecionado os respectivos pontos de segmentação e duração do segmento, de acordo com a duração, taxa de bits e quantidade de quadros do vídeo.

Após receber os parâmetros da segmentação, cada nó de transcodificação inicia o processamento dos respectivos segmentos (veja Figura 3.1). O nó mestre possui uma área do sistema de arquivo compartilhada com os nós de transcodificação, onde pode ser encontrado o vídeo original.

Os nós computacionais realizam a transcodificação de todos os segmentos de forma embaraçosamente paralela, gerando arquivos no formato H.264 sem um contêiner. Ao final, o nó mestre junta todos os segmentos gerados durante a transcodificação distribuída em um só arquivo, encontrados na área de dados comparti-

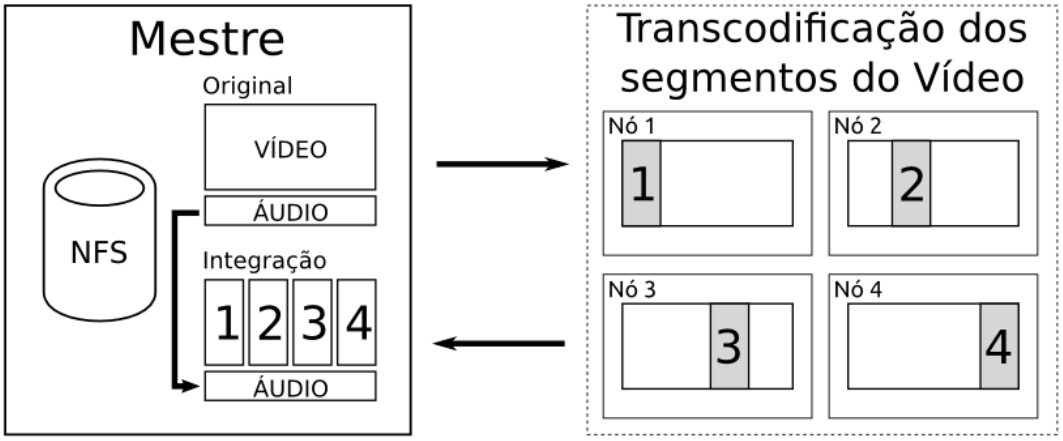

Figura 3.1: Etapas da transcodificação distribuída com um servidor mestre e 4 servidores de transcodificação

lhada, para então criar o contêiner MP4 [12] do vídeo, sincronizando com o áudio codificado.

Para gerar arquivos H.264 sem contêiner, é necessário armazenar os quadros utilizando o *Byte stream format* definido no Anexo B da especificação [3]. Desta forma, o processo de juntar os segmentos se assemelha a uma concatenação de arquivos de vídeo, não exigindo processamento extra além da inserção do contêiner MP4 na etapa final do processo.

3.2 Segmentação de Vídeo por Busca

A nossa solução para segmentar o vídeo sem perder as referências entre os quadros, requer que os nós de transcodificação acessem o arquivo do vídeo original e caminhem por ele até alcançar o ponto de início da conversão. Convencionalmente, uma ação de busca num arquivo de vídeo faria a decodificação quadro a quadro até o ponto especificado. Desta forma, cada um dos N processos realiza a decodificação de todo o vídeo anterior ao seu trecho de trabalho. O que equivale ao processo N decodificar todo o vídeo, anulando o ganho de desempenho na distribuição da transcodificação.

Com isto, a quantidade de vídeo decodificado é dada pela soma de P.A. com razão $1/N$ sendo equivalente a $(N+1)/2$ vezes o tamanho do vídeo quando idealmente seria apenas uma vez.

3.2.1 Busca Otimizada

A nossa técnica de busca foi aprimorada para reduzir o percentual de decodificação extra necessário na definição do segmento. A idéia é fazer uma busca indexada pelo quadro I e procurar qual está mais próximo do início do segmento e a partir daí, decodificar os quadros até o fim do segmento. A Figura 3.2 ilustra a escolha dos segmentos e o posicionamento do quadro I que será usado na decodificação do respectivo segmento. O tempo de busca do quadro I é praticamente desprezível comparado com a decodificação quadro-a-quadro na busca convencional.

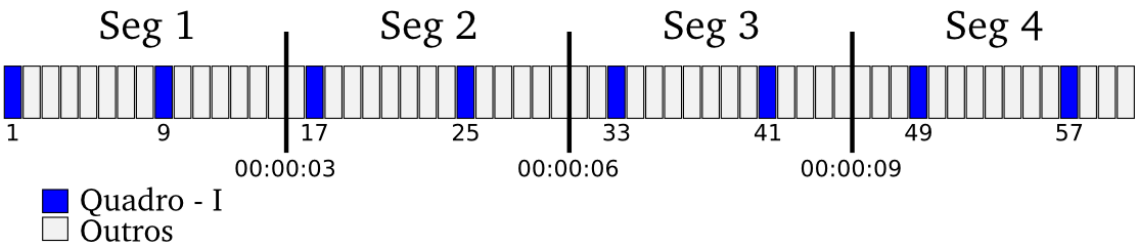

Figura 3.2: Segmentação do arquivo de vídeo.

A Figura 3.2 mostra um vídeo com taxa de 5 quadros por segundo e um total de 60 quadros. Para 4 servidores de transcodificação, o sistema seleciona os tempos 0, 3, 6 e 9 segundos como os devidos limites e atribui 15 quadros a cada segmento. Através da Figura podemos descrever a criação do segmento 2. O segmento foi delimitado pelos tempos 3 e 6 segundos. Na transcodificação é feita uma busca do quadro I mais próximo e anterior ao segmento. Encontrado o quadro 9, este é usado como referência para a decodificação do segmento 2.

O uso de busca otimizada irá reduzir a quantidade de transcodificação total realizada em relação à busca simples, mas esta quantidade ainda será maior ou

igual à quantidade do processo convencional, que é de apenas uma vez. No caso ótimo, onde todos os cortes são feitos exatamente num quadro I do vídeo, não haverá decodificação extra, mas no pior caso, onde o corte é feito logo antes de um quadro I, haverá a decodificação extra do GOP anterior. Desta forma, o pior caso exige decodificar tantos GOPs extras quantos forem os segmentos do vídeo. Probabilisticamente, a segmentação do vídeo cairá no caso intermediário a esses dois, onde existirá na média, decodificação extra de apenas metade do GOP anterior.

Incidentalmente, por mais que haja necessidade de decodificação extra, este processo ocorrerá em paralelo nos vários processos que executam simultaneamente. Isto fará com que o tempo extra gasto considerado seja apenas o do nó mais lento.

3.3 Adequação do uso de recursos de CPU – Processos simultâneos

Como visto na seção 3.1.1, cada nó de transcodificação deve receber um ou mais segmentos de vídeo para processar simultaneamente. Isso se dá ao fato que o processamento de apenas um segmento pode não ser suficiente para esgotar os recursos de CPU da máquina, pois o processo de transcodificação necessita de grande volume de entrada e saída de dados, produto e resultado da decodificação e codificação, respectivamente. Desta forma, o uso de múltiplos processos por máquina irá acarretar numa melhora significativa no uso de CPU e, consequentemente, no tempo de codificação.

O fato da segmentação do vídeo desenvolvida ser a nível de quadros reduz a dependência de dados existente. No caso da transcodificação puramente multithread que os codificadores oferecem, a divisão de trabalho se dá nos blocos do

quadro, havendo maior dependência entre eles e não permitindo o uso efetivo de todo poder de processamento disponível.

Cada processo de transcodificação iniciado pode ter uma certa quantidade de threads simultâneas para executar o passo de codificação, além de outras threads de propósito geral. Para o processo de transcodificação distribuída, vamos deixar o FFmpeg trabalhar com as threads de propósito geral da forma que ele escolha melhor e limite as threads de codificação a fim de encontrar um valor que seja adequado à máquina de testes.

Desta forma, ao variar a quantidade de processos simultâneos que executarão em cada máquina e também o número de threads de codificação de cada processo, podemos encontrar um balanço entre essas duas estruturas de paralelismo que melhore o desempenho da máquina.

3.4 Balanceamento de Carga

A quantidade de processos simultâneos definida na seção anterior vai definir uma quantidade máxima de segmentos que o vídeo poderia ser subdividido e esta quantidade provavelmente não vai ser a ideal pois o processamento de segmentos de mesmo tamanho não garante um tempo de processamento igual, uma vez que a complexidade de codificação está mais inerente ao conteúdo do vídeo que ao seu comprimento.

Sabendo que alguns segmentos podem demorar mais tempo para serem finalizados que outros, é possível criar um sistema de balanceamento de carga que subdivida o vídeo em um número de segmentos maior que a capacidade de processamento simultânea do cluster e tente manter a carga de trabalho sempre constante.

Portanto, é possível melhor balancear a carga de vídeo no final do processo, onde a discrepância tende a ser cada vez menor.

O mecanismo de balanceamento utilizado para a técnica descrita neste projeto inicia em cada máquina de transcodificação um certo número de threads para controlar o pedido de segmentos. Inicialmente, cada uma dessas threads vai pedir um segmento para a fila de execução existente no servidor mestre e aguardar a transcodificação terminar. Ao fim da transcodificação de cada segmento assistido, a thread que estava em seu controle pede ao servidor mestre um novo segmento e assim o processo se repete paralelamente até que se esgotem os segmentos de vídeo.

3.5 Transcodificação do Áudio

A pesar do foco deste trabalho ser apenas a codificação do fluxo de vídeo, é importante apresentar como a parte do áudio foi tratada e os possíveis impactos para a técnica.

Por causa de limitação de algumas ferramentas e *codecs*, a segmentação de um arquivo de áudio pode gerar diversas imperfeições e ruídos no processo de junção de pedaços codificados. Desta forma e pelo fato deste processo ser inerentemente mais simples e possivelmente mais rápido de executar que a transcodificação de vídeo, fizemos com que ele fosse executado sequencialmente numa máquina à parte.

Absolutamente falando, é possível que, se utilizarmos muitos nós de transcodificação para vídeo, que o tempo da transcodificação de áudio passe a ser o limitante da técnica.

Por conseguinte, é necessário estudar novas técnicas de transcodificação de áudio para verificar a possibilidade de dividir o processo de forma a este não atrapalhar o tempo total de transcodificação.

Para a execução, utilizamos o *codec* de áudio AAC por ser o mais comumente utilizado em conjunto com o H.264 e pelo alto grau de compactação possibilitado.

Capítulo 4

Avaliação Experimental e Resultados

Para avaliar o sistema de transcodificação distribuído proposto realizamos experimentos em um cluster composto de 5 nós: um nó mestre, responsável pela segmentação, escalonamento de tarefas e a integração final dos segmentos transcodificados; e mais 4 nós, responsáveis pela transcodificação dos segmentos.

Servidores	4 (Transcodificação) + 1 (Mestre)
Processadores	2 x Intel Xeon E5620 2.4GHz 4-Core HT
Memória	16GB ECC DDR3
Rede	1 Gigabit/s
Disco	HD 1TB Sata III 3Gbps 7200rpm
Sistema Operacional	Ubuntu 13.04 64-bit
Sistema de Distribuição	Python 2.7
Transcoder	FFmpeg 1.2
Bibliotecas	libx264, libvpx e libfdk_aac
Armazenamento	Partição NFS no Mestre

Tabela 4.1: Ambiente Experimental - Nó Computacional

A Tabela 4.1 apresenta as principais características de *Hardware* e *Software* do ambiente experimental, sendo o uso de 4 nós de transcodificação e 1 servidor mestre, todos rodando o sistema operacional Ubuntu mais recente.

Cada nó do cluster possui dois processadores Intel com 4 núcleos, habilitados para a execução de duas threads por núcleo, oferecendo um total de 8 processadores físicos e a execução de 16 threads simultâneas.

Nos experimentos utilizamos o software FFmpeg [13] com libx264 para implementar a análise dos metadados, transcodificação e a integração dos segmentos transcodificados. O FFmpeg permite, através da opção “seek" [14], que possamos facilmente fazer a busca de um quadro I em um ponto desejado de um arquivo de vídeo (como descrito na seção 3.2.1). Os codecs H.264, VP8 e AAC utilizados no FFmpeg são provindos das bibliotecas libx264, libvpx e libfdk_aac, respectivamente.

Toda a lógica de processamento de metadados, distribuição de segmentos e balanceamento de carga foi desenvolvida utilizando Python.

4.1 Detalhes dos Vídeos Gerados

A Tabela 4.2 mostra detalhes dos vídeos gerados em cada um dos experimentos. O processo de transcodificação utilizado foi o mesmo em todos os testes para garantir pouca variação na qualidade de imagem, também mantendo a mesma resolução e taxa de quadros do arquivo de entrada.

Codecs	H.264 e AAC
Resolução	Igual à do vídeo utilizado incialmente
Taxa de Quadros	Igual à do vídeo utilizado incialmente
Controle de taxa de bits	CRF 23
GOP	Variável: Até 250 quadros + *Scenecut*
Quadros P e B	Sim
Quadros de Referência	Até 16
Profile H.264	High
Preset libx264	Medium
Compressão	CABAC

Tabela 4.2: Ambiente Experimental - Detalhes do vídeo gerado

A métrica CRF (Constant Rate Factor) é determinada na biblioteca libx264 como a quantidade de perda visual permitida no vídeo. Esse valor varia de 0 a 51 e afeta a qualidade da imagem de forma logarítimica, sendo 0 o caso onde não há

perdas e 51 o pior caso que o codificador gera. Para valores abaixo de 19, o olho humano tem muita dificuldade de perceber perdas na qualidade e o valor padrão aceito por obter um bom *trade-off* entre qualidade e taxa de bits é 23[15], sendo este utilizado nos testes.

A estrutura de GOP utilizada nos vídeos gerados escolhida é variável de até 250 quadros (padrão libx264) com a estratégia *Scenecut* habilitada, que permite criar um quadro **I** quando for notada uma mudança brusca na cena do vídeo.

É importante notar que o máximo GOP possível pode ser limitado também pelo tamanho do segmento de vídeo gerado. Dentro de cada GOP, são utilizados quadros **I**, **P** e **B** com possibilidade de até 16 quadros de referência para compressão temporal.

O *profile* do H.264 utilizado é o *High*, que codifica o vídeo de forma mais eficiente e produz melhores taxas de bits que outros *profiles*, como *Main* e *Baseline*. Em comparação, o *profile High* faz uso mais intensivo de recursos de CPU que os outros. Para esse *profile*, o método de compressão utilizado foi o CABAC.

A biblioteca libx264 tem uma série de *presets* que definem as estratégias de processamento do vídeo que influenciam diretamente velocidade de execução. O *preset* utitlizado foi o padrão *Medium* que tem um bom comprometimento entre tempo de execução e taxa de bits. Como foi definido o uso de CRF, é possível deixar a codificação mais rápida (*presets Fast*, *Very Fast* ou *Ultra Fast*) ou mais devagar (*presets Slow* e *Very Slow*) para aumentar ou reduzir a taxa de bits, respectivamente.

4.2 Experimento Preliminar – Vídeo Curto

Para testar o funcionamente e a capacidade do sistema distribuído, foi utilizado um vídeo de curta duração, mas que fosse comprido o suficiente para possibilitar a divisão por quatro máquinas. A partir dos testes preliminares, foi possível observar um certo desbalanceamento de carga que nos permitiu modificar o projeto para melhorar o desempenho do sistema.

O vídeo escolhido [16] está inicialmente codificado em H.264 com a taxa de 1200 kbps e resolução de 1280×720 pixels. Para o áudio foi usado a codificação AAC [4], mantendo a taxa original de 64 kbps. A duração do vídeo é de 4 minutos e 25 segundos (265 segundos). Nas duas instâncias o áudio e o vídeo foram sincronizados no formato MP4 [17].

Este vídeo tem por título "*CCD: The heart of a digital camera*[I]" e será referenciado no texto pela sigla CCD.

Formato	MP4
Codecs	H.264 e AAC
Resolução	1280×720 px
Bitrate de Vídeo	1200 kbps
Bitrate de Áudio	148 kbps
Duração	4min 25s (265s)
Taxa de Quadros	29.97 fps
GOP médio	55.7 frames (1.86s)

Tabela 4.3: Ambiente Experimental - CCD

[I] "CCD: O coração da câmera digital" – Tradução independente

4.2.1 Avaliação da Transcodificação Convencional

Antes de colocar em prática o teste distribuído, foi analisado o comportamento da transcodificação convencional em uma do nós de transcodificação servir de parâmetro para os testes subsequentes. Neste experimento, variamos a quantidade de *threads* de codificação utilizadas para gerar o vídeo final. Foram tomados a média e o desvio padrão de 30 execuções consecutivas de cada teste.

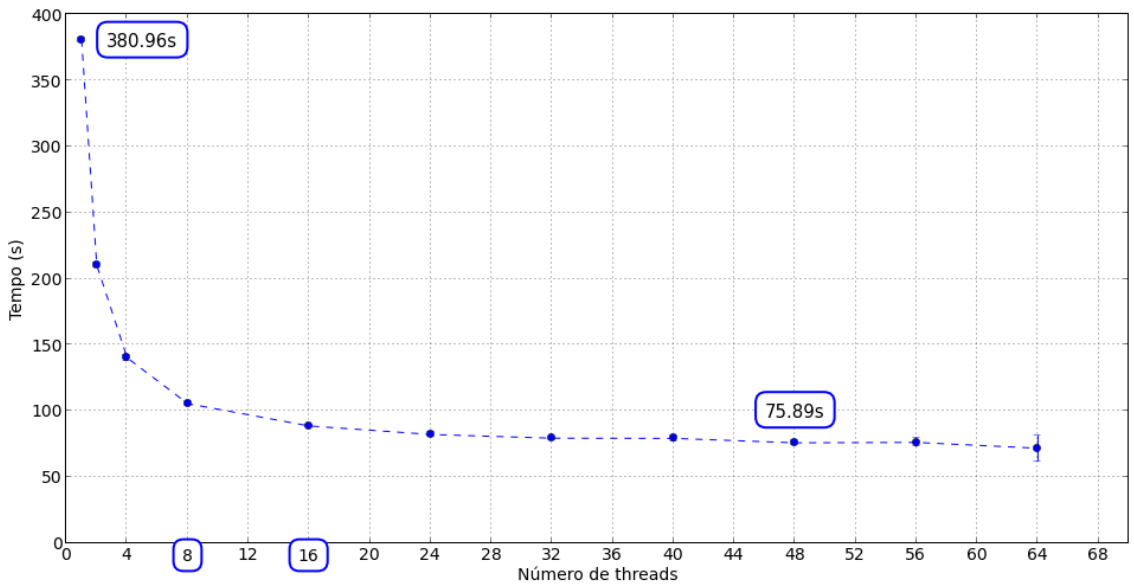

Figura 4.1: Tempo da Transcodificação em 1 Nó com 8 Cores – CCD.

O desempenho da transcodificação, no modo *multithreading*, é mostrado no gráfico da Figura 4.1. Nesta máquina de 8 *cores* com *Hyper-threading*, observamos um desempenho 5 vezes melhor com 48 threads. As quantidades maiores de threads apresentaram resultados próximos, mas com variação mais alta do tempo de execução.

Todavia, o uso de mais 16 threads de codificação oferece pouco ganho de desempenho e, de acordo com a biblioteca libx264, é aconselhado que a quantidade de threads de transcodificação não extrapole este valor por questões de estabilidade. Desta forma, todos os testes realizados serão limitados em 16 threads por

processo, sem passar de 64 threads por máquina, devido ao resultado apresentar aumento na variação estatística.

4.2.2 Distribuição de Segmentos pelo Cluster

Após a investigação do comportamento *multithread*, o vídeo foi segmentado em partes de mesma duração e estas foram enviadas simultaneamente a cada um dos nós do cluster. Este teste é importante para mostrar que a técnica de segmentação de vídeo obteve êxito e nos leva a pensar em melhorias.

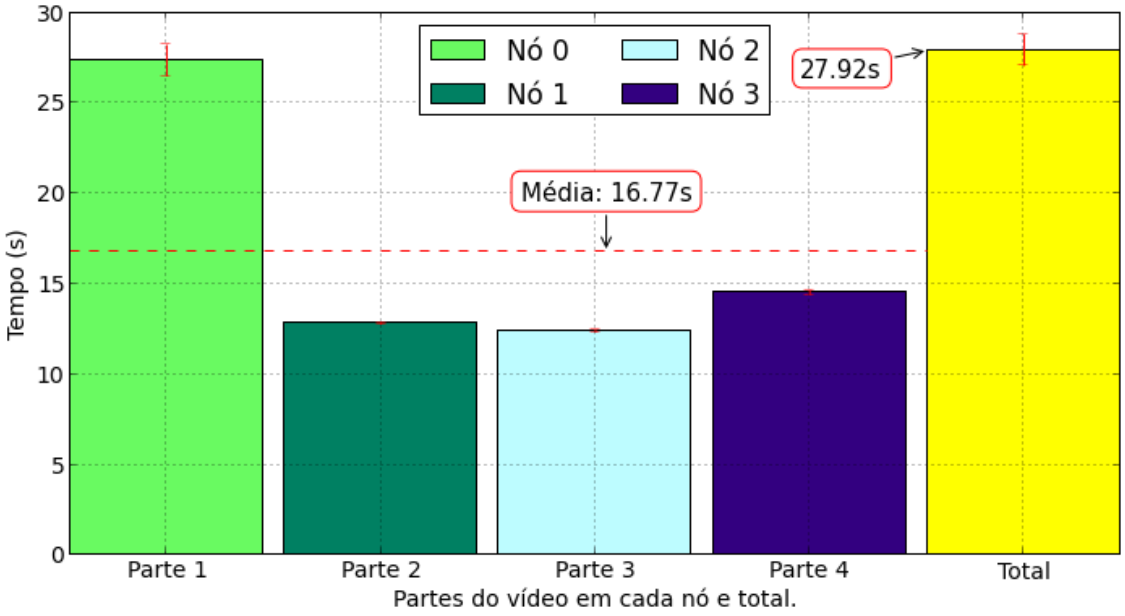

Figura 4.2: Tempo da Transcodificação Distribuída – 4 Segmentos em 4 Nós com 16 Threads por Segmento – CCD.

Análise	Transcodificação	Integração	Total
0,19s	27,36s	0,37s	27,92s

Tabela 4.4: Componentes do Tempo Total de Transcodificação da Figura 4.2

O gráfico de barras na Figura 4.2 apresenta o resultado da execução do sistema de transcodificação distribuída no ambiente experimental. Um processo FFmpeg, executando 16 threads, foi enviado a cada nó para a transcodificação de um seg-

mento de 1/4 do vídeo. No referido gráfico encontramos o tempo de transcodificação de cada segmento e o tempo total, composto pelos tempos decorridos na análise/segmentação, transcodificação e a integração do áudio e segmentos de vídeo no formato MP4. A tabela 4.4 apresenta o tempo decorrido em cada componente da barra Total.

O resultado apresentado demonstra a eficiência do sistema proposto. O tempo de conversão do vídeo foi 27,92 segundos, que representa um ganho de desempenho de 2,7 vezes se comparado com melhor resultado do modo *multithreading*. Podemos observar no gráfico a ocorrência de desbalanceamento de carga no escalonamento dos processos de transcodificação. A explicação é a variação da complexidade na compressão de diferentes trechos do vídeo. Devido ao desbalanceamento de carga, a média do tempo de transcodificação de cada segmento foi 16,77 segundos. Este resultado sugere que podemos aumentar o desempenho do sistema se melhoramos o escalonamento das tarefas.

4.2.3 Threads e Processos

Tendo em vista o desbalanceamento de carga ocorrido no experimento anterior, procuramos diminuir a ociosidade dos nós aumentando o número de segmentos de vídeo para transcodificação. Neste experimento exploramos o uso dos recursos do nó de transcodificação variando a quantidade de threads e processos. Executamos de 8 a 64 threads por nó e foram escalonados de 1 a 8 processos de transcodificação simultaneamente.

Os resultados são apresentados no gráfico da Figura 4.3. O menor tempo obtido - 13,5 segundos-, foi na transcodificação usando 8 segmentos por nó e 8 threads por processo. Observamos no gráfico que o aumento da quantidade de segmentos contribui mais para melhorar o desempenho da transcodificação distribuída do que o aumento do número de threads em cada nó. A razão deve-se ao aumento do número de segmentos e em consequência a dispersão dos trechos de codificação

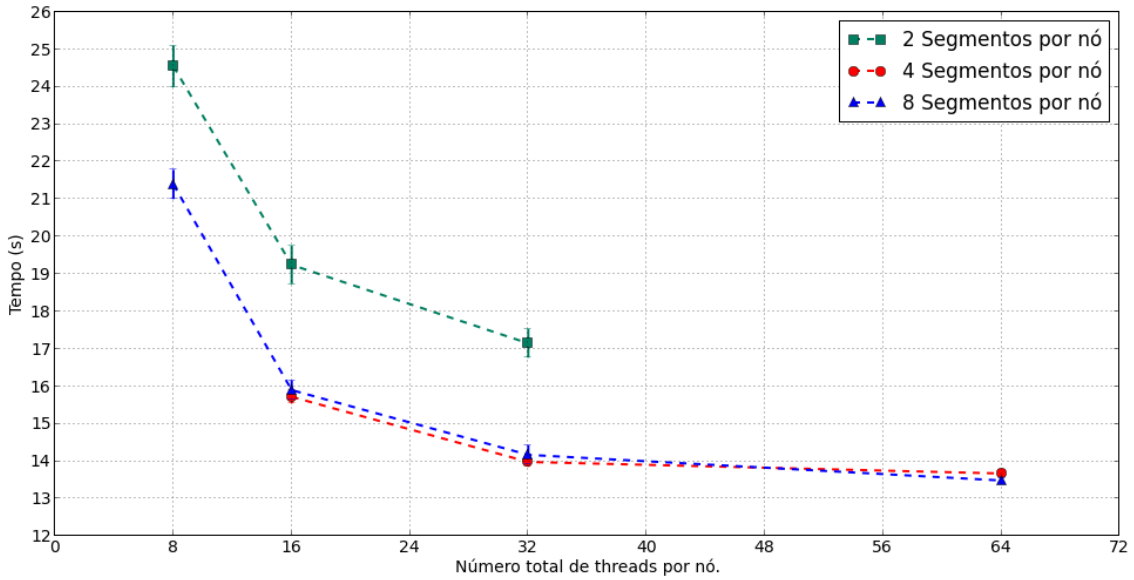

Figura 4.3: Tempo da Transcodificação Distribuída vs. No. Segmentos – Cluster com 4 Nós – CCD.

mais complexos entre os nós, diminuindo o desbalanceamento da carga, como podemos observar no gráfico da Figura 4.4.

O gráfico mostra os tempos de transcodificação de cada segmento para a melhor composição de segmentos e threads. Observamos a melhora no desbalanceamento da carga entre os nós, o que consequentemente contribuiu para o aumento do desempenho do sistema de transcodificação distribuído.

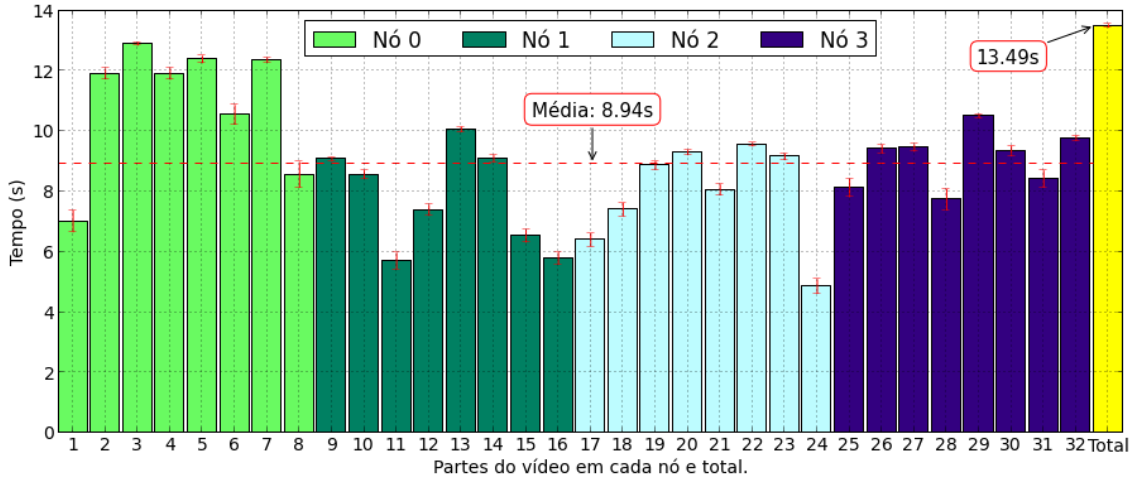

Figura 4.4: Tempo da Transcodificação Distribuída – 32 Segmentos em 4 Nós com 8 threads por segmento – CCD.

Análise	Transcodificação	Integração	Total
0,19s	12,93s	0,37s	13,49s

Tabela 4.5: Componentes do Tempo Total de Transcodificação da Figura 4.4

Em comparação com o melhor caso no modo multithreading, onde havia 48 threads, o processamento é feito 5,6 vezes mais rápido com uso de apenas 4 vezes mais recursos de hardware. Entretanto, a aceleração que a técnica proporciona não pode ser gerada em comparação com o caso original pois a técnica puramente multithread não é suficiente para utilizar todos os recursos de hardware que a máquina utilizada possui.

Por causa do desbalanceamento de carga ainda evidente na discrepância dos tempos de execução dos segmentos, acreditamos que é possível alcançar um tempo de processamento paralelo mais próximo da média dos segmentos aplicando-se um escalonamento eficiente de segmentos menores de vídeo. Feito isso, podemos variar a quantidade de máquinas dentro da técnica para verificar a aceleração real no melhor caso.

4.2.4 Balanceamento de Carga

Ainda utilizando o mesmo vídeo, novos testes foram executados para analisar a eficácia de um sistema balanceador de carga. Este experimento varia o número total de segmentos de vídeo, mas sempre transcodificando 32 simultaneamente no *cluster*, ou seja, 8 por máquina. Fica assumido para este e experimentos seguintes que esta distribuição é a mais efetiva para o ambiente com estas especificações, de acordo com os dados expostos na figura 4.3.

Neste modo de funcionamento, no mestre é criada uma fila com o total de segmentos a serem processados e, em cada um dos nós, são iniciados 8 controladores de tarefas para fazer pedidos de segmentos. Isso faz com que o escalonamento de processos do sistema operacional embaralhe os pedidos de vídeo, sendo possível qualquer ordem de execução acesso para essas primeiras tarefas. Ao terminar de

processar um segmento, cada controlador faz um novo pedido ao servidor mestre até que a fila de tarefas se esvazie.

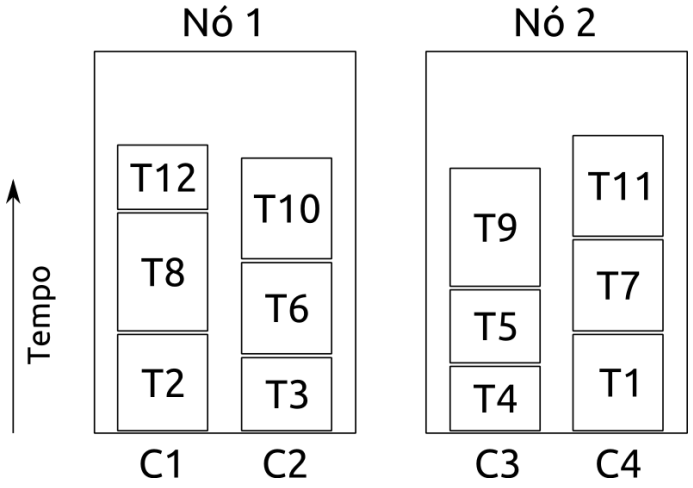

Figura 4.5: Escalonamento de tarefas para 2 nós de transcodificação com 2 controladores de pedidos e um total de 12 segmentos de vídeo (tarefas a serem executadas).

A Figura 4.5 exemplifica como, em dois nós de transcodificação, ocorrem os pedidos de tarefas para a fila de execução. Inicialmente, cada um dos controladores C1 a C4 pediu um segmento para executar e foram atendidos em ordem aleatória com as tarefas T1 a T4. O primeiro a terminar a execução foi C3, pedindo então o próximo segmento. Assim, até que os 12 segmentos de vídeo desse exemplo fossem executados, os controladores continuaram fazendo pedidos na ordem que foram terminando suas tarefas.

Com esse modelo de processamento implementado, podemos ver os resultados após variar do número total de segmentos para esse vídeo (tarefas), sendo visto na Figura 4.6. Esta figura apresenta a variação de 32 a 192 tarefas com um passo de 32. A escolha destes números se dá pelo total de controladores de tarefas ativo no sistema, escolhendo sempre um múltiplo deste número para que cada controlador execute a conversão do mesmo número de segmentos, aproximadamente.

O primeiro gráfico da figura ilustra o fato de que embaralhar os segmentos de vídeo tem como consequência positiva o desacoplamento de trechos do vídeo mais complexos que antes podiam ser executados numa mesma máquina, mas

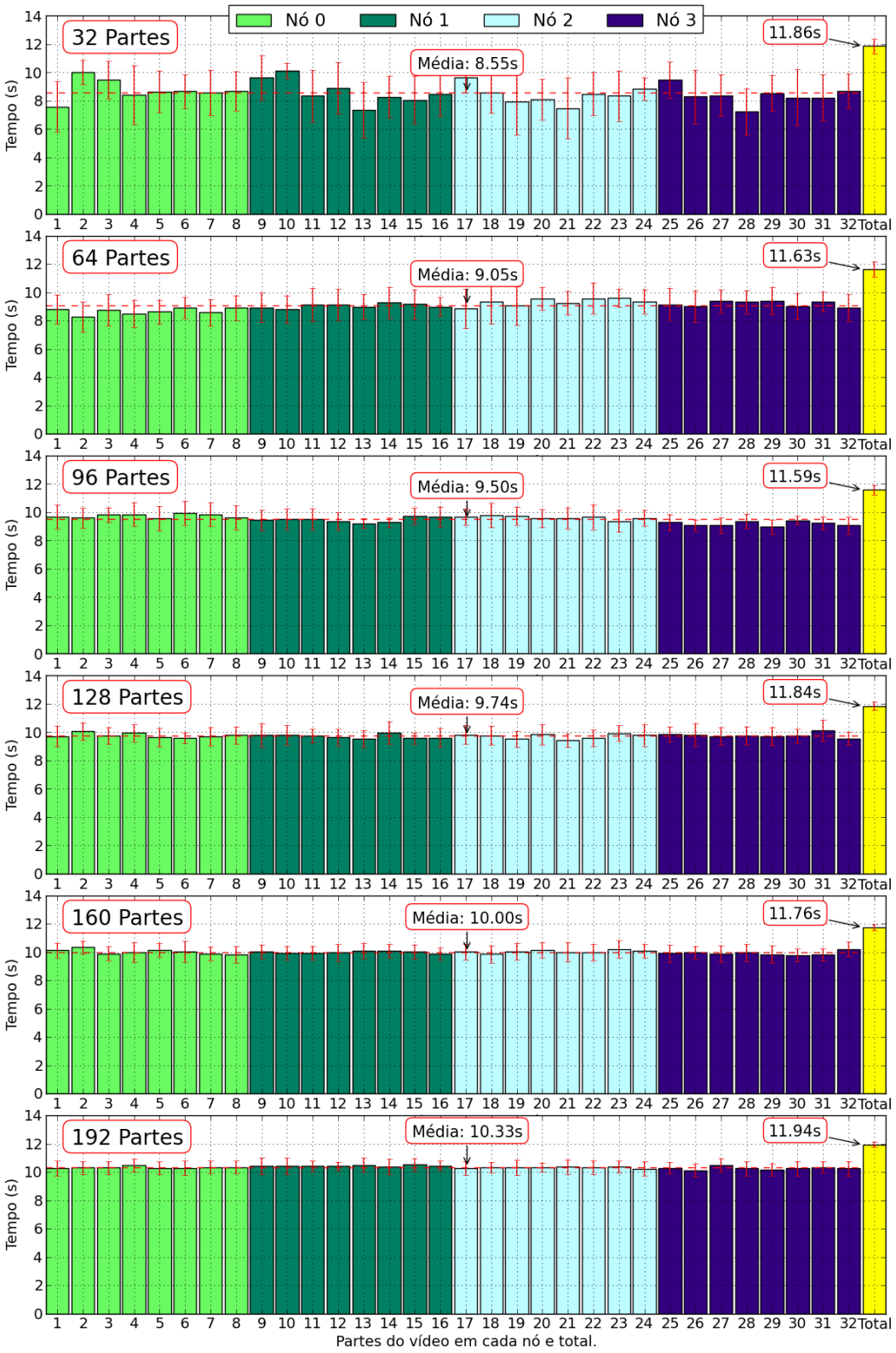

Figura 4.6: Variação na quantidade de segmentos de vídeo com transcodificação simultânea de 32 segmentos em 4 máquinas – CCD.

agora estão espalhados, criando uma situação mais balanceada que a anterior, mas ainda com grande variação estatística. Note que, mesmo com a melhora de 1,6 segundos, a média dos segmentos ainda é consistente com o caso anterior, sendo menor em apenas 0,39s por manter todas as máquinas um pouco mais ociosas ao invés de apenas um delas estar carregada ao fim do processo.

Os gráficos subsequentes da figura 4.6 apresentam um balanceamento de carga cada vez mais evidente. Cada uma das barras mostra o somatório dos tempos de transcodificação dos segmentos sob controle de cada uma das *threads* de pedido.

Os tempos dos segmentos se aproximando cada vez mais da média e o desvio padrão sendo reduzido com o aumento da quantidade de segmentos, entretanto, não são suficientes para reduzir o tempo de transcodificação pois este aumento gera um aumento na quantidade de decodificação de vídeo extra necessária para iniciar cada codificação. Isso é refletido no aumento da média das transcodificações.

Entre os 6 gráficos da figura 4.6, o caso com 96 segmentos teve o menor tempo total de execução e apresentou um balanceamento de carga com desvio padrão suficientemente baixo para poder ser escolhido como melhor opção de transcodificação para esse arquivo de vídeo.

Com duração de 265 segundos e GOP médio de 1.86 segundos, são necessários cerca de 143 segmentos para que aconteça mais de uma decodificação por GOP, deixando o processo cada vez mais ineficiente. O cálculo de decodificação extra de 50% o tamanho do GOP médio só é válido para segmentos maiores que esta unidade.

4.2.5 Aceleração da Técnica e Tempos de execução

A aceleração é a métrica para avaliar o ganho de desempenho com a variação da quantidade de paralelismo. Neste caso, a unidade de paralelismo é o nó de transcodificação, onde cada nó é capaz de transcodificar 8 segmentos simultaneamente. Para esse experimento, repetimos o teste assumido como melhor no caso anterior (96 segmentos) e variamos a quantidade de máquinas: de uma a quatro.

O gráfico 4.7 mostra em azul a reta de aceleração ideal e, em vermelho, o ganho de desempenho com a variação do número de máquinas.

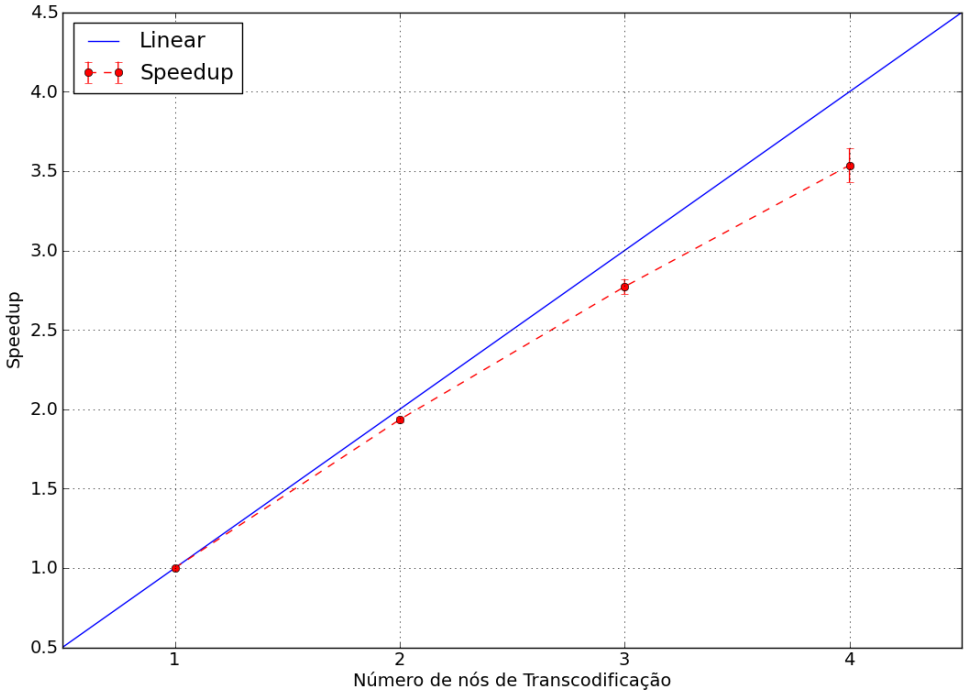

Figura 4.7: Aceleração real da técnica de balanceamento (com reta de aceleração linear teórico para comparação) – CCD.

Por fim, a Figura 4.8 apresenta a comparação entre o tempo original do caso puramente multithread com 48 threads com os tempos obtidos com o sistema de balanceamento de carga. A figura mostra a evolução ocorrida com o aumento do número de nós de transcodificação.

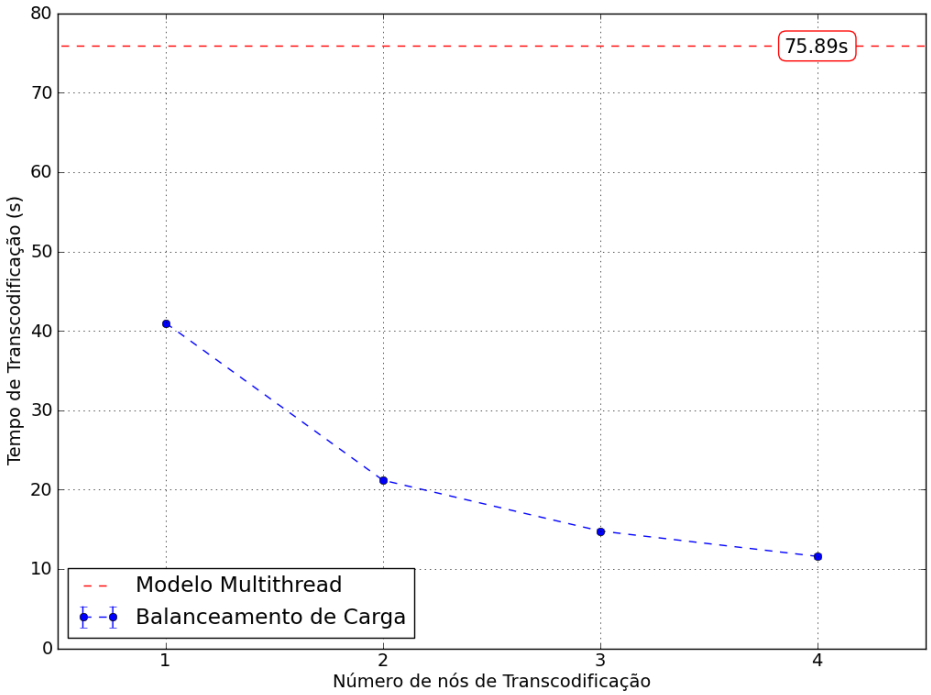

Figura 4.8: Tempos de Transcodificação com Balanceamento de Carga – CCD.

Em relação ao modelo puramente multithread onde o tempo de execução foi de 75,9s, o ganho de desempenho para o caso com 4 máquinas foi de 6,5 vezes. Entretanto, é importante ressaltar que não houve uma aceleração superlinear, pois o caso original subutilizava os recursos de CPU da máquina. Sendo assim, a comparação só é justa quando é colocada uma máquina com a mesma quantidade de processos paralelos executando, o que resultou na aceleração de 3,6 vezes visto acima.

A informação da Figura 4.8 é interessante para mostrar como é possível deixar o modelo de transcodificação de vídeo mais eficiente sem aumentar a capacidade computacional. Com apenas uma máquina, houve uma redução de 54% do tempo de transcodificação no modelo balanceado em relação ao tempo original.

4.2.6 Transcodificação da faixa de Áudio

Para fins de comparação, para este arquivo de vídeo, a faixa de áudio foi transcodificada numa máquina separada das que executavam a transcodificação dos segmentos de vídeo. Esta transcodificação ocorre de forma paralela e converte o áudio AAC de 148kbps para AAC de 128kbps, mantendo-se as outras características.

O tempo de execução do processo foi de 9,0 $\pm$ 0,1 segundos, tendo sido repetido 30 vezes.

A transcodificação da faixa de áudio se dá de forma sequencial e, como é possível perceber, termina cerca de 2,1s antes da transcodificação de vídeo obtida no melhor caso visto dos resultados da Figura 4.8. Entretanto, os experimentos foram realizados com no máximo 4 máquinas. De acordo com a aceleração vista na Figura 4.7, o uso de 6 ou mesmo 8 máquinas ainda deve apresentar um resultado não muito distante da reta de aceleração ideal, podendo fazer com que a transcodificação de vídeo termine antes que o áudio esteja pronto e faça com que este passo se torne o novo gargalo do sistema.

4.3 Vídeo Longo – BBC

Para este experimento, foi escolhido um vídeo [18] de maior duração para poder analisar efeitos de tempo de decodificação quando o segmento se aproxima do tamanho do GOP. Este vídeo está inicialmente codificado em H.264 com a taxa de 2300 kbps e resolução de 1280$\times$720 pixels. Outros detalhes estão na Tabela 4.6.

O vídeo tem por título *"Natural World – A Highland Haven*[II]*"*, produzido pela BBC e será referenciado no texto por esse acrônimo. Esse vídeo foi seleciona pela sua grande variação na complexidade visual oferecida, contendo uma mistura de cenas que podem ser muito simples ou muito complicadas para o processo de codificação.

Formato	MP4
Codecs	H.264 e AAC
Resolução	1280×720 px
Bitrate de Vídeo	2300 kbps
Bitrate de Áudio	160 kbps
Duração	58min 47s
Taxa de Quadros	29.97 fps
GOP médio	49.3 frames (1.64s)

Tabela 4.6: Ambiente Experimental - BBC

4.3.1 Balanceamento de Carga

De acordo com os resultados vistos na seção 4.2.4 sobre balanceamento de carga do vídeo "CCD", vemos que o uso da técnica é mais efetivo que a simples divisão do vídeo em N segmentos que serão executados simultaneamente e com sequência definida. No caso inicial do modelo de balanceamento de carga onde o mesmo número de tarefas é gerado quando comparado ao modelo anterior, os resultados serão melhores no caso médio, pelo embaralhamento da ordem de execução nos nós de transcodificação. Portanto, para este vídeo só serão demonstrados os resultados de testes com balanceamento de carga.

Visto que, para essa configuração de servidores, a transcodificação de 8 processos simultaneamente com 8 threads por processo em cada uma dos nós apresenta os melhores resultados, isto foi repetido nestes testes.

[II] "O Mundo Natural – Um refúgio nas alturas" – Tradução independente

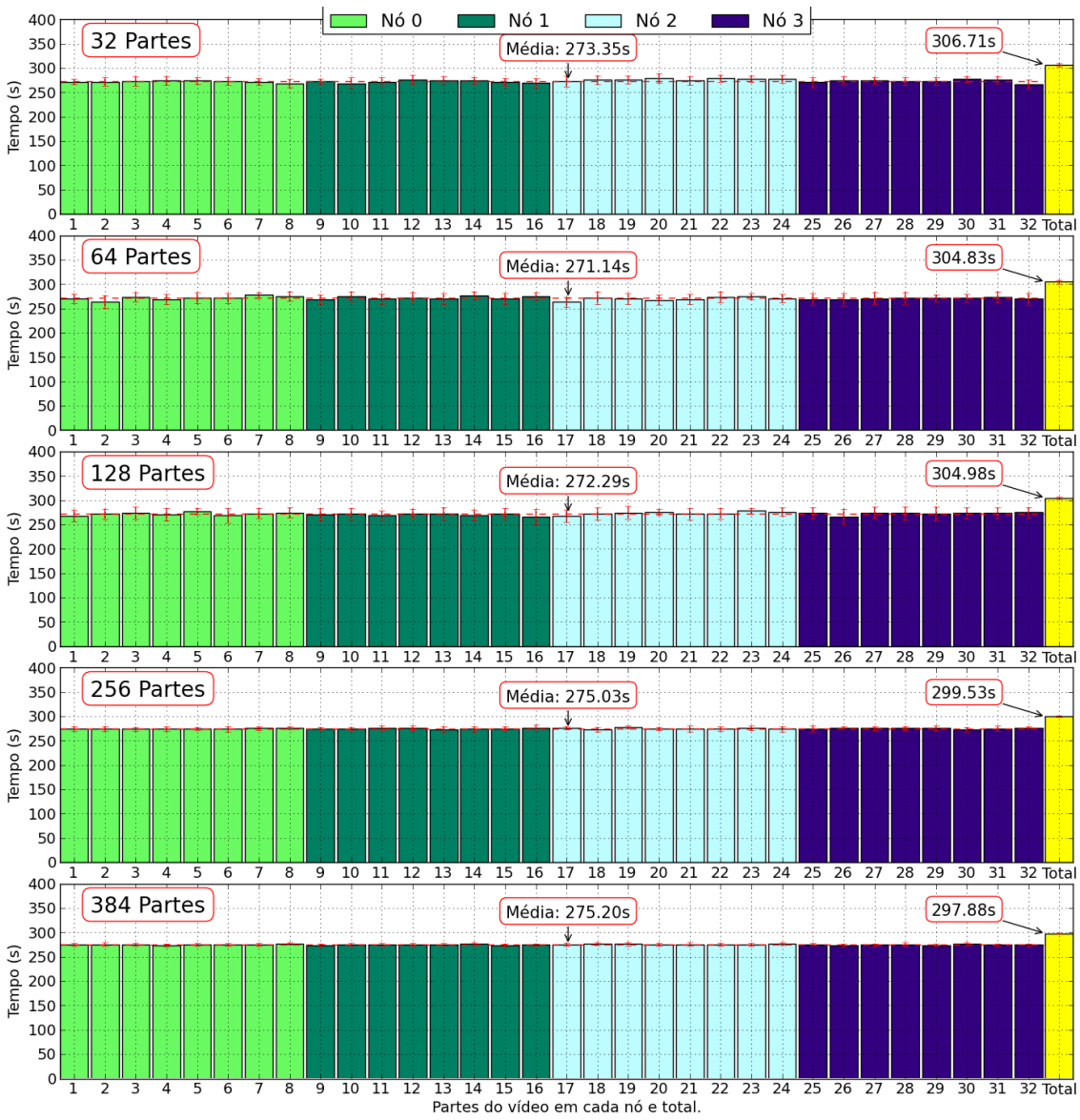

Figura 4.9: Variação na quantidade de segmentos de vídeo com transcodificação simultânea de 32 segmentos em 4 máquinas (32 a 384 segmentos) – BBC.

As Figuras 4.9 e 4.10 apresentam a variação do número de segmentos de vídeo de 32 a 1024 segmentos, com passo de 128 segmentos sem considerar os gráficos de 32 e 64 segmentos. O motivo de dividir este vídeo em uma quantidade muito maior de segmentos que o vídeo anterior se dá pela sua duração de uma hora, que é cerca de 12 vezes maior que a do outro vídeo.

Novamente, de acordo com o resultado dessas figuras, podemos perceber que o aumento do número de segmentos de vídeo melhora o balanceamento de carga, mas também aumenta a quantidade de decodificação extra necessária a ser execu-

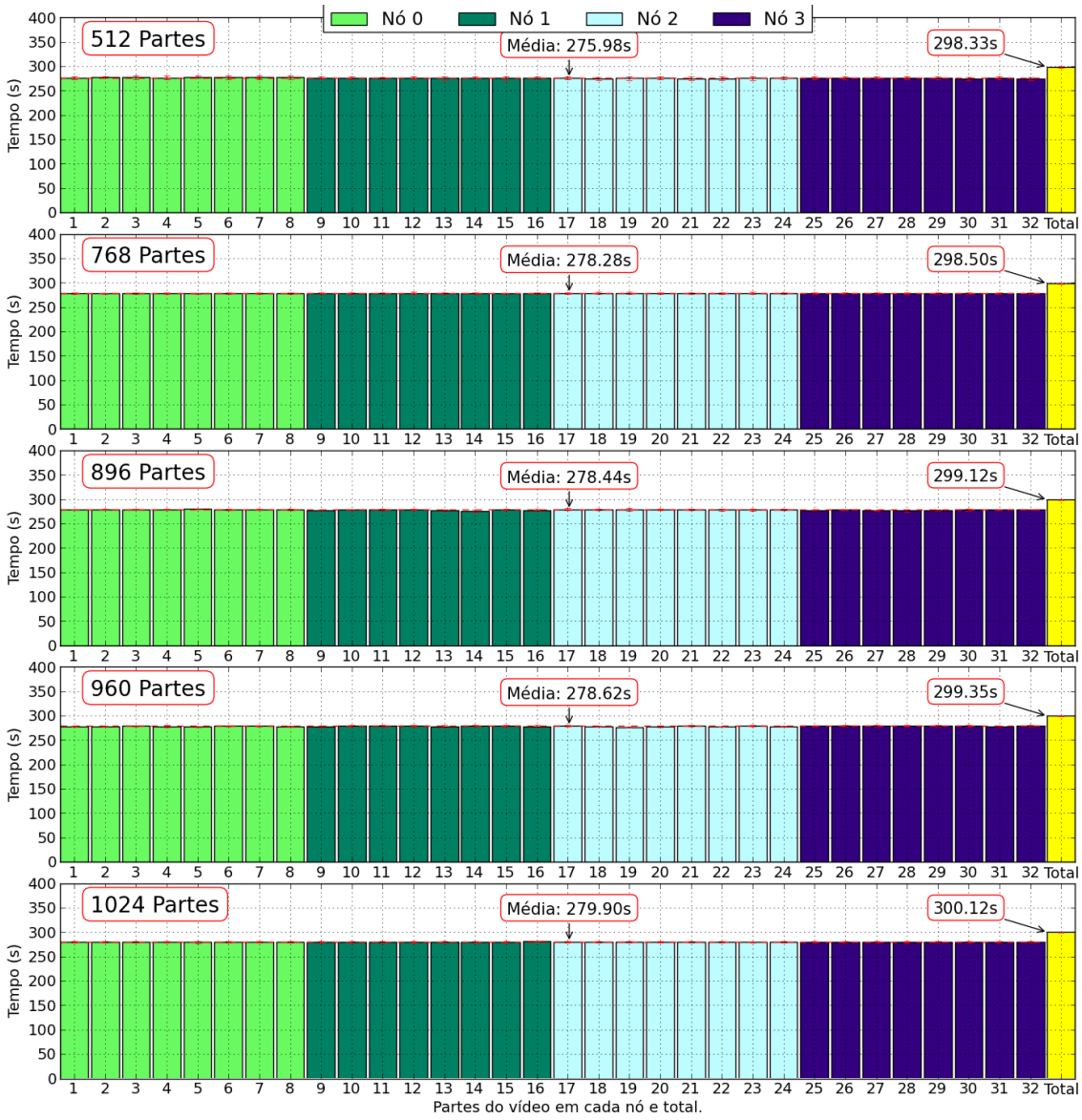

Figura 4.10: Variação na quantidade de segmentos de vídeo com transcodificação simultânea de 32 segmentos em 4 máquinas (512 a 1024 segmentos) – BBC.

tada. Isto se reflete no aumento da média do tempo de execução dos segmentos.

Uma característica marcante do fato desse vídeo ser mais comprido que o anterior é que, proporcionalmente, os balanceamento de carga fica aparente com uma quantidade não tão grande de segmentos, podendo ser visto desde o caso com 256 segmentos, onde cada valor do gráfico dista da média em menos de 4%. Assim, o fato de aumentar o número de segmentos pouco influencia no controle do desbalanceamento (chegando a cerca de 1% com 1024 segmentos), mas tem um impacto negativo mais visível na decodificação extra necessária.

Para este vídeo, o melhor resultado de transcodificação se dá com o vídeo segmentado em 384 partes, tendo um tempo total de 297,9 segundos ou aproximadamente 5 minutos. Com esta divisão, cada segmento tem cerca de 9 segundos de duração, que é aproximadamente 5 vezes maior que o tamanho do GOP médio do vídeo.

Com essas informações, podemos conjeturar que as dimensões do *cluster* de transcodificação possam definir a quantidade de segmentos necessários para se obter o nível de balanceamento de carga, dado que a quantidade de tarefas seja grande o suficiente para que o tempo de transcodificação de cada uma delas seja aproximadamente o mesmo. Isto mostra que não só o GOP do vídeo é suficiente para definir a quantidade de segmentos, mas também as características do ambiente, como o número de nós e quantidade de processadores que controlam a capacidade de transcodificação simultânea de segmentos.

4.3.2 Aceleração da Técnica e Tempos de execução

Novamente podemos traçar a curva de aceleração do processo de transcodificação, levando em consideração apenas os resultados do balanceamento de carga.

A Figura 4.11 mostra como a técnica de balanceamento se comporta ao variar o número de nós de transcodificação de 1 a 4 e mantendo os mesmos 8 processos por máquina. No caso desse vídeo que difere do anterior quase que exclusivamente pela duração maior, vemos uma aceleração mais próxima do caso linear ideal, chegando a 3,8x com 4 nós.

Este fato traz a possibilidade de aumentar o tamanho do cluster e ainda obter resultados satisfatórios.

Observando-se os tempos de transcodificação da Figura 4.12, que apresentam a mesma variação de nós da figura anterior, também vemos que desde o caso

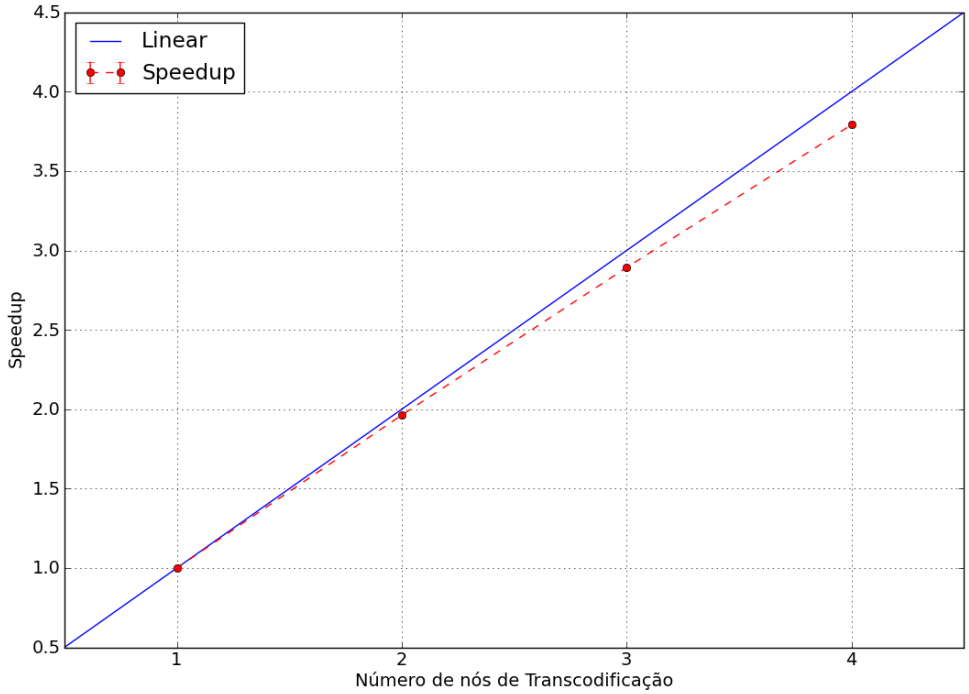

Figura 4.11: Aceleração real da técnica de balanceamento (com reta de aceleração linear teórico para comparação) - BBC.

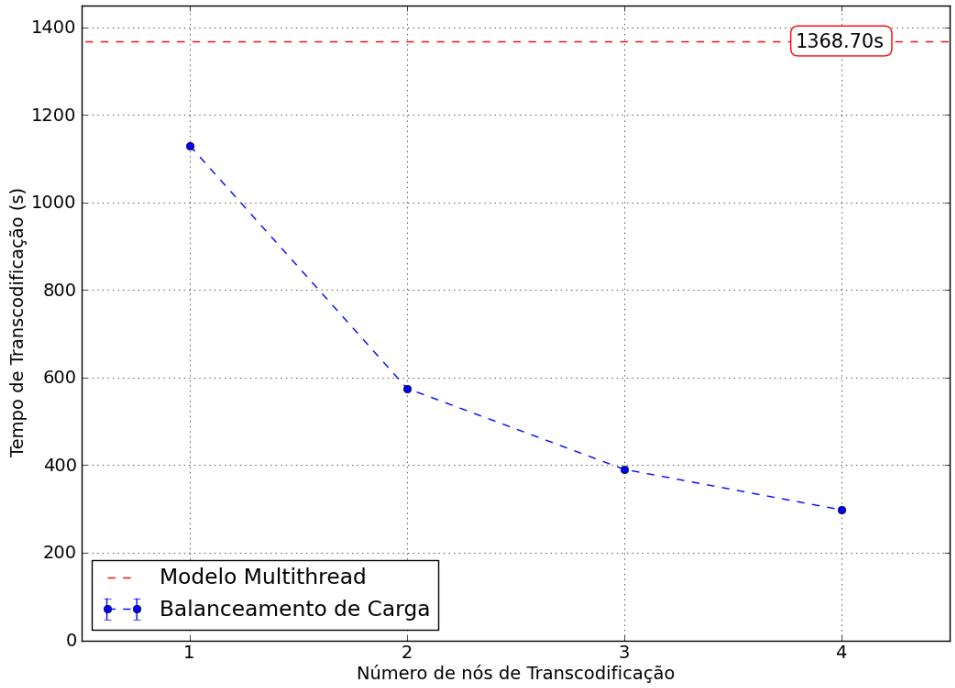

Figura 4.12: Tempos de Transcodificação com Balanceamento de Carga comparado ao modelo puramente multithread – BBC.

com apenas uma máquina já há ganhos visíveis de desempenho ao comparar com o tempo de transcodificação puramente multithread para este vídeo. Isto se dá pelos mesmos fatores de melhor utilização de recursos de hardware vistos na seção 4.2.5.

4.4 Video Médio – Sintel

Para o último experimento, foi escolhido um vídeo [19] de duração intermediária entre os outros e, desta vez com uso do codec VP8 e um GOP de maior duração. Este vídeo está inicialmente codificado com a taxa de 3000 kbps e resolução de 1920×818 pixels. Outros detalhes estão na Tabela 4.7.

O vídeo tem por título "Sintel" e apresenta uma duração intermediária entre os outros dois testados. Pelo fato de inicialmente este vídeo ser codificado em VP8 será possível verificar as diferenças de decodificação entre este codec e o H.264. De acordo com *Bankoski et al.* [20], a compressão intra-frame do codec VP8 exige menos processamento que a existente no *profile* "High" do H.264 (CABAC), fazendo com que a decodificação seja mais eficiente.

Formato	WEBM
Codecs	VP8 e Vorbis
Resolução	1920×818 px
Bitrate de Vídeo	3000 kbps
Bitrate de Áudio	128 kbps
Duração	14min 48s
Taxa de Quadros	24 fps
GOP médio	113 frames (4.71s)

Tabela 4.7: Ambiente Experimental – Sintel

4.4.1 Balanceamento de Carga

Da mesma forma como no vídeo anterior, analisamos a transcodificação com balanceamento de carga para 8 processos por máquina com 8 *threads* por processo simultaneamente.

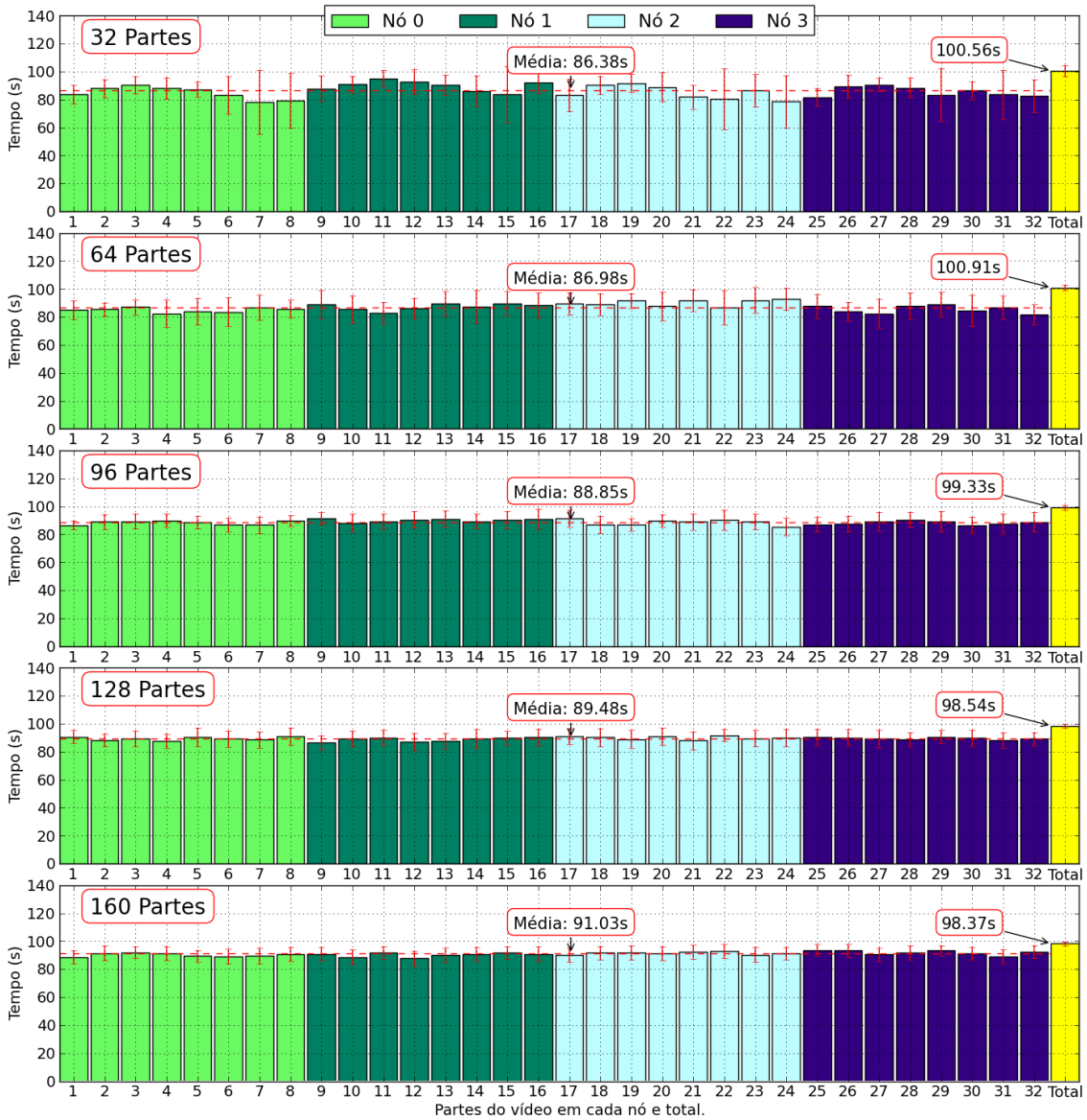

Figura 4.13: Variação na quantidade de segmentos de vídeo com transcodificação simultânea de 32 segmentos em 4 máquinas (32 a 160 segmentos) – Sintel.

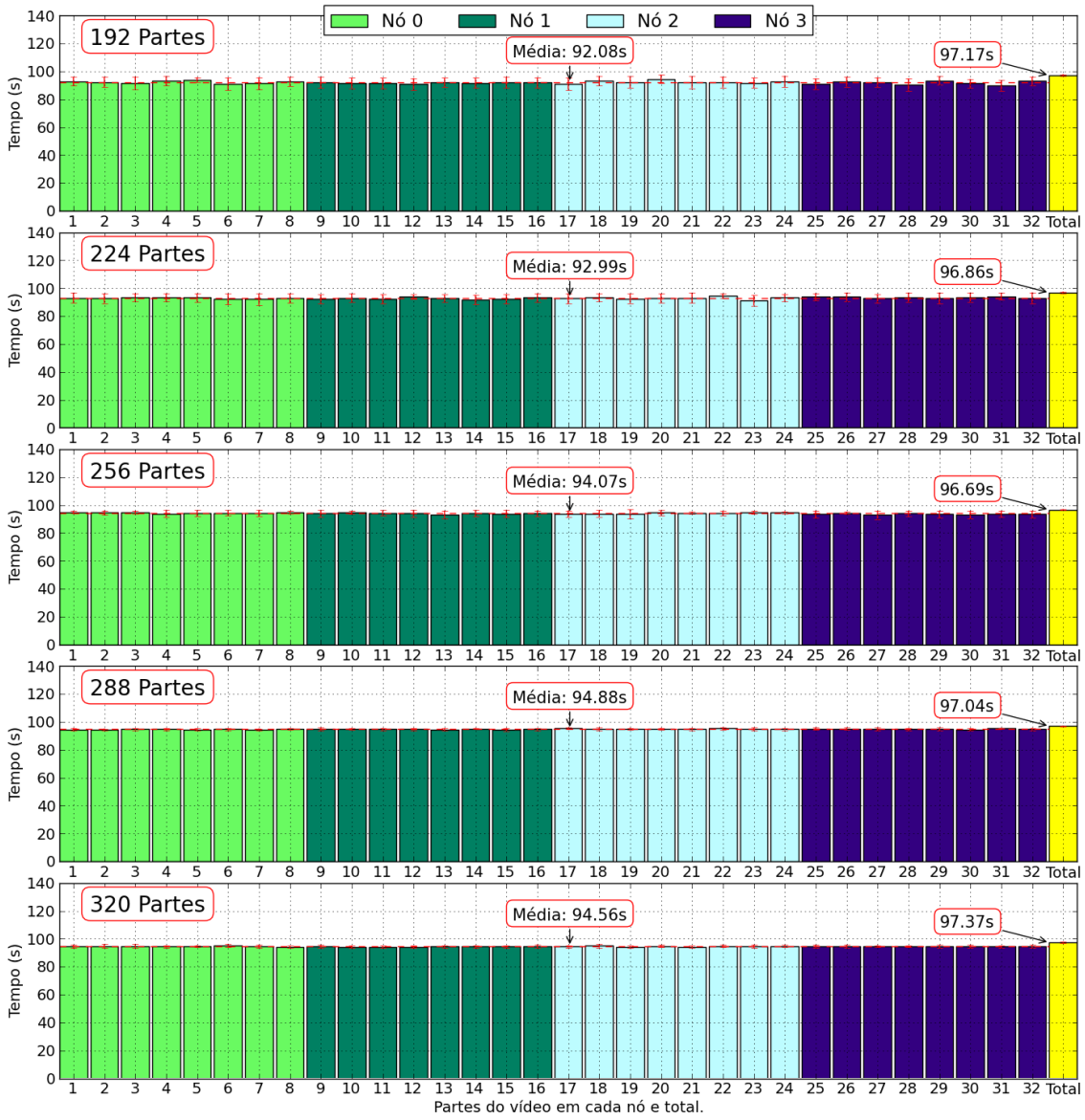

Figura 4.14: Variação na quantidade de segmentos de vídeo com transcodificação simultânea de 32 segmentos em 4 máquinas (192 a 320 segmentos) – Sintel.

Nas Figuras 4.13 e 4.14, são executados os processos com 32 a 320 segmentos de vídeo simultaneamente, variando com passo de 32.

Da mesma forma como ocorrido nos outros vídeos, o aumento do número de segmentos melhora o balanceamento de carga, mas aumenta a média do tempo de transcodificação por causa da quantidade de decodificação extra. Novamente, em 256 segmentos, temos um desbalanceamento menor que 5% e este é o caso onde ocorre o melhor tempo total de execução.

4.4.2 Aceleração da Técnica e Tempos de execução

Finalmente, vemos os dados de aceleração para o último vídeo de teste na Figura 4.15. Esse resultado apresenta uma curva muito mais próxima da reta ideal que os outros caso, apresentando um valor de 3,92x no caso com 4 nós de transcodificação.

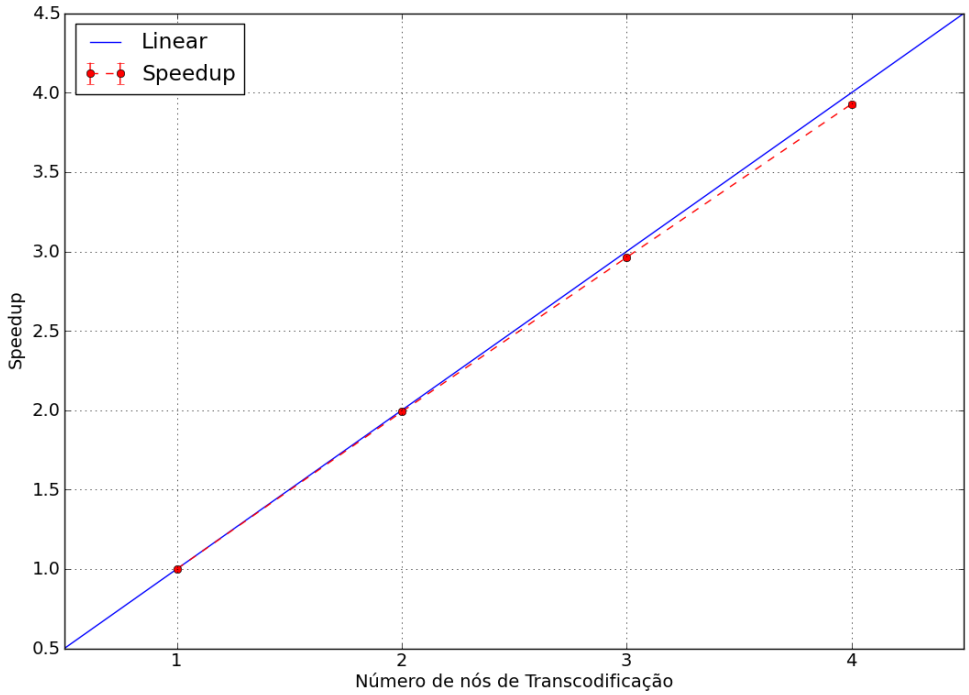

Figura 4.15: Aceleração real da técnica de balanceamento (com reta de aceleração linear teórico para comparação) – Sintel.

O motivo aparente para essa melhoria na aceleração em relação ao último vídeo (BBC), é o fato da decodificação do codec VP8 ser mais eficiente que a do H.264, permitindo que a quantidade de decodificação extra realizada seja menos impactante e permita um balanceamento de carga mais efetivo [20].

O gráfico da Figura 4.16 mostra a progressão do sistema com a mesma variação de 1 a 4 nós de transcodificação, apresentando os resultados já esperados

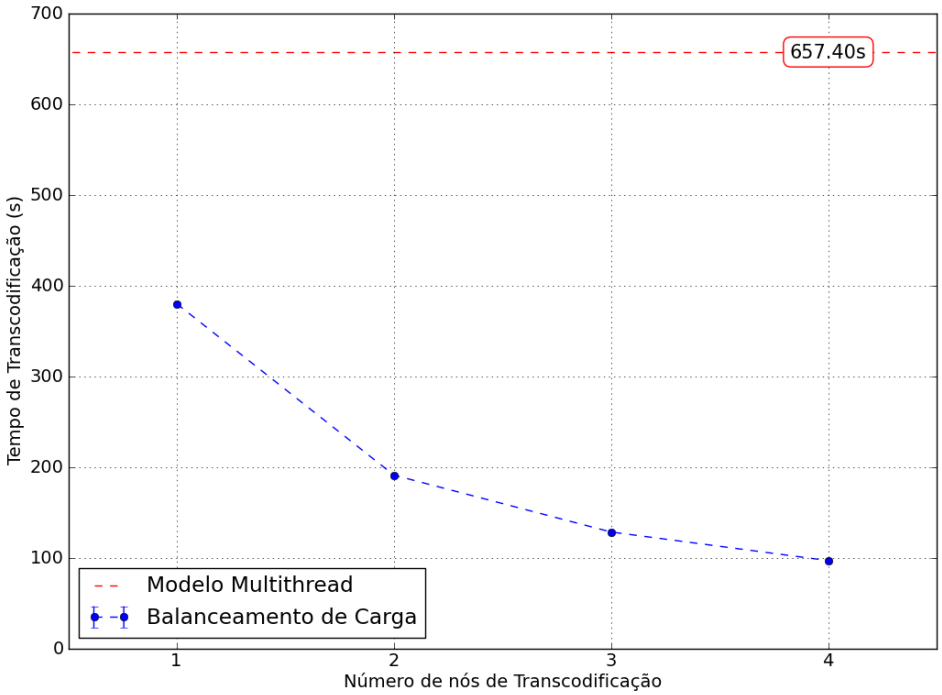

Figura 4.16: Tempos de Transcodificação com Balanceamento de Carga comparado ao modelo puramente multithread – Sintel.

de melhora no tempo de processamento em comparação com o caso puramente multithread desde o ponto com apenas uma máquina executando a conversão.

4.5 Resultados e Trabalhos Relacionados

O intuito inicial do projeto foi desenvolver um método capaz de segmentar um arquivo de vídeo com compressão temporal de forma eficiente e sem a necessidade de decodificar completamente o clipe. O motivo para essa determinação foi o estudo de trabalhos de cunho similar onde, em Pereira et al. [2] o procedimento se dá a partir de um vídeo já decodificado temporalmente sem haver consideração do gasto de tempo inicial para executar esta tarefa.

Os maiores problemas desta abordagem são: Necessidade de um longo passo sequencial antes de iniciar a codificação, grande aumento na taxa de bits do vídeo, causando aumento na banda de rede utilizada da ordem de 20 a 30 vezes e possível degradação da qualidade de imagem já que o vídeo decodificado é armazenado em MJPEG.

Em vista desses problemas que tornam a técnica inviável para sistemas de vídeo sob demanda que trabalham quase que exclusivamente com vídeos com compressão temporal, elaboramos nossa técnica de forma a não executar nenhuma etapa de decodificação ou codificação anterior à distribuição. Isto pode ser visto nos tempos da seção 4.2.3 onde o tempo de análise e extração do áudio é de 0,19 segundo, constante para variações de quantidade de segmentos e comprimento dos vídeos.

O trabalho de Sambe et al. [5] apresenta uma visão mais realista do problema, trabalhando diretamente com o vídeo codificado temporalmente. A sua técnica envia GOPs de vídeo a cada um dos nós de processamento, fazendo com que a quantidade de dados trafegados fosse mínima. Entretanto, o seu procedimento de segmentação ainda acontece de forma sequencial e, de acordo com dados apresentados, permite escalar a técnica para cerca de 10 processos simultâneos apenas.

A técnica por nós exposta apresenta, de acordo com os testes das seções 4.2.4, 4.3.1 e 4.4.1, resultados da aceleração próximos ao linear com a execução de

32 processos simultâneos no *cluster* de testes, deixando espaço para expansão para uma quantidade superior de máquinas, onde espera-se que haja ganhos de desempenho consideráveis.

Capítulo 5

Conclusão e Trabalhos Futuros

5.1 Conclusão

O objetivo deste trabalho foi criar e avaliar a possibilidade de dividir a tarefa de transcodificação de vídeo por múltiplos processos e máquinas interconectados de forma a permitir melhor aproveitamento de recursos de hardware disponíveis.

A escolha deste projeto surgiu para acelerar o processo de transcodificação de arquivos em um sistema de vídeo sob demanda, onde clipes produzidos em diversos formatos são convertidos para um único ou alguns poucos formatos, tendo algumas das suas características variadas, como taxa de bits e resolução. Este processo de transformação, ainda bastante sequencial, será transformado para aproveitar as características dos atuais processadores *multicores*.

O desenvolvimento de um modelo de segmentação de vídeo com base em tempo reduziu a quantidade de dependências necessárias para paralelizar a forma de transcodificação do vídeo, permitindo que fosse possível disparar diversos processos independentes sem necessidade de processamento prévio do conteúdo do vídeo.

Com uma técnica de segmentação efetiva, foi possível aplicar um modelo de balanceamento de carga ao sistema de forma a propiciar a divisão do vídeo em

partes menores, fazendo com que as durações dos tempos de transcodificação dos segmentos fossem mais homogêneas, evitando que algumas máquinas pudessem ficar mais ocupadas que outras, o que gera desperdício de uso de CPU.

Finalmente, vemos que a quantidade máxima de segmentos de vídeo produzida para que o processo ainda seja eficiente está ligada ao tamanho do GOP, com variações de acordo com o *codec* utilizado. Por causa da decodificação extra que a técnica proporciona, é interessante que os segmentos de vídeo sejam maiores que o tamanho médio do GOP de forma a existir um equilíbrio entre as vantagens do balanceamento com as desvantagens da decodificação.

Portanto, a partir dos tempos de transcodificação obtidos, podemos depreender que o processo desenvolvido já traz ganhos no tempo total de transcodificação com a mesma máquina que seria utilizada no caso puramente sequencial, já que este não é capaz de utilizar todos os recursos de CPUs *multicores*. Portanto, ao aumentar a quantidade de nós de transcodificação, ainda percebemos um alto ganho de desempenho quando comparado com o modelo original.

5.2 Trabalhos Futuros

Este trabalho teve a premissa de avaliar a possibilidade e implementar um sistema de transcodificação de vídeo distribuído para uso com *codecs* que utilizem compressão temporal. Dado que essas tarefas foram concluídas, novas vertentes de pesquisa são possíveis e algumas delas são aqui analisadas:

5.2.1 Análise de GOP e Variação dinâmica do tamanho dos Segmentos

De acordo com a pesquisa de Sambe et al. [5], a segmentação de vídeo com base no GOP exige um pré-processamento alto o suficiente para limitar a técnica em cerca de 10 processos simultâneos. Também é visto que o GOP não é uma boa unidade de trabalho por ser variável na maioria dos vídeos com codecs H.264 e VP8 – os mais comuns no mercado.

Entretanto, foi visto aqui que a média dos tamanhos dos GOPs de um vídeo (GOP médio) é uma métrica que influencia diretamente a quantidade de decodificação extra que a técnica aqui proposta terá que realizar. Desta forma, existe a necessidade de analisar esse parâmetro e esta análise não pode ser feita sequencialmente para não limitar a técnica.

Uma solução a ser estudada é a de escolher o tamanho de segmentação do vídeo de acordo com uma estimativa de GOP médio e iniciar o envio de segmentos para transcodificação. Enquanto os segmentos estão sendo convertidos, alguns processos podem fazer análise do tamanho do GOP em diversas partes do vídeo e estimar o GOP médio com melhor precisão.

Quando o resultado da análise de GOP chegar, a parte do vídeo que ainda não foi convertida pode ser dividida novamente com um tamanho mais adequado ao vídeo.

De acordo com os resultados para os três vídeos do capítulo 4, o fato da carga estar melhor balanceada pode oferecer até certo ponto um melhor tempo de processamento que a redução da transcodificação extra. Com isso, a transcodificação pode iniciar de forma não ótima e ser corrigida posteriormente sem grandes perdas no tempo.

5.2.2 Cluster heterogêneo

Neste trabalho, todas as máquinas utilizadas para testes tinham a mesma capacidade de processamento, memória, rede etc. No entanto, é possível levar essa técnica a ambientes em que exista grande variedade de máquinas e não será possível aplicar a mesma carga para todas elas.

Com isso, para poder trabalhar com um cluster heterogêneo, é preciso estabelecer um modo de aplicar carga de forma a todas as máquinas ficarem igualmente ocupadas e evitar desbalanceamento de carga.

Para implementar esse comportamento, o balanceamento de carga não deve levar em conta apenas o número de processadores disponíveis, mas sim a capacidade computacional de cada nó. Isso poderia ser feito de antemão e quando o sistema fosse iniciado, já é sabido quantos segmentos uma certa máquina suporta codificar simultaneamente.

Uma outra forma de obter o efeito seria ir aumentando dinamicamente a carga em cada nó e fazer análise de uso de recursos para saber se a carga deve ser aumentada ou diminuída. Além disso, máquinas com maior poder de processamento poderiam trabalhar com segmentos maiores que as máquinas mais simples, para assim tentar reduzir a ociosidade das máquinas.

5.2.3 Sistemas de Arquivos Distribuído e Balanceamento de Carga com Roubo de tarefa

Neste trabalho foi testada a distribuição de carga com 4 servidores conectados numa rede local e compartilhamento de arquivos por meio de NFS. Esse sistema de arquivos é preparado para esse tipo de ambiente e pode não ser tão efetivo caso haja um aumento considerável de nós de transcodificação.

Uma proposta para continuar esse estudo pode ser a mudança para outro sistema de arquivos que não seja obrigatoriamente centralizado, como é o caso do NFS.

Com um sistema distribuído de armazenamento, o arquivo a ser processado pode estar parcialmente presente em diversos pontos do sistema. Assim, existe possibilidade de incrementar a eficiência se for implementado um balanceamento de carga por roubo de tarefa para cada nó transcodificar partes do arquivo de vídeo que estejam próximas na rede.

5.2.4 Análise de Qualidade/Taxa de Bits

Para o funcionamento da técnica, cada segmento codificado insere um Quadro I no seu início, onde possivelmente isso não seria realizado na transcodificação puramente sequencial.

Esse provável aumento na quantidade de quadros I pode ter dois impactos diferentes de acordo com a forma de transcodificação.

Caso o vídeo esteja sendo convertido com uma taxa de bits aproximadamente constante, deve ser verificado um impactos na qualidade pois esse tipo de quadro ocupa normalmente mais espaço que outros. No caso onde a qualidade é mantida constante (CRF[I]), deve ser possível notar um aumento na taxa de bits do vídeo.

Esse comportamento será agravado com o aumento do número de divisões no vídeo transcodificado e isto deve ser analisado para ser possível escolher um comportamento que seja eficaz e ainda produzir boa qualidade de imagem ou bom controle da taxa de bits.

[I]Constant rate factor.

5.2.5 Transcodificação de Áudio

Por mais que esta técnica seja limitada à transcodificação de vídeo, é interessante produzir um análogo para transcodificação de áudio e assim permitir que esta também seja feita de forma mais eficiente.

Como visto na Seção 3.5 e nos testes executados na Seção 4.2.6, o tempo de transcodificação de áudio pode ser um limitante da técnica sendo portanto um novo alvo de estudo para permitir um número cada vez maior de nós de transcodificação.

O motivo inicial para evitar a transcodificação paralela de áudio foi o fato de surgirem imperfeições, ruídos e chiados no arquivo final ao juntar segmentos. Entretanto, isso pode ser devido a falhas nos softwares utilizados no momento da pesquisa (FFmpeg e Mencoder).

Com o lançamento do FFmpeg 2.0[13], a documentação aponta que a funcionalidade de busca (*seek*) para áudio produz acurácia do tamanho da amostragem, talvez podendo ser utilizado para essa funcionalidade.

Referências Bibliográficas

[1] OLIVEIRA, J., DUTRA, D., WHATELY, L., *et al.*, "Transcodificação de Vídeo Distribuída de Alto Desempenho", 19th Brazilian Symposium on Multimedia and the Web – Webmedia, 2013.

[2] PEREIRA, R., BREITMAN, K., "An Architecture for Distributed High Performance Video Processing in the Cloud", *IEEE 3rd International Conference on Cloud Computing (CLOUD)*, pp. 482 – 489, 2010.

[3] ITU-T, "ITU-T H.264 Recomendation - Version 16", `http://www.itu.int/rec/T-REC-H.264`, 2013, Acessado: 15/06/2013.

[4] ISO, *ISO/IEC 13818-7:2006: Generic coding of moving pictures and associated audio information – Part 7: Advanced Audio Coding (AAC)*, 2006. Available in English only.

[5] SAMBE, Y., WATANABE, S., YU, D., *et al.*, "High-Speed Distributed Video Transcoding for Multiple Rates and Formats", *IEICE - Trans. Inf. Syst.*, v. E88-D, n. 8, pp. 1923–1931, Aug. 2005.

[6] YARBUS, A. L., *Eye Movements and Vision*. Plenum Press, New York, 1967.

[7] OSBERGER, W., *Perceptual Visual Models for Picture Quality Assessment and Compression Applications*. Ph.D. dissertation, Queensland University of Technology, Queensland, Australia, 1999.

[8] RICHARDSON, I., *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*. Wiley, 2004.

[9] POYNTON, C., *Digital Video and HDTV: Algorithms and Interfaces*, Electronics & Electrical. Morgan Kaufmann Publishers, 2003.

[10] SADKA, A., *Compressed Video Communications*. Wiley, 2002.

[11] FIELDING, R., IRVINE, U., GETTYS, J., *et al.*, "Hypertext Transfer Protocol – HTTP/1.1 – RFC 2616", `http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html`, 1999, Acessado: 03/07/2013.

[12] WIEGAND, T., SULLIVAN, G., BJONTEGAARD, G., *et al.*, "Overview of the H.264/AVC video coding standard", *Circuits and Systems for Video Technology, IEEE Transactions on*, v. 13, n. 7, pp. 560–576, 2003.

[13] FFMPEG, "FFmpeg", `http://ffmpeg.org/`, 2013, Acessado: 10/06/2013.

[14] FFMPEG, "Seeking with FFmpeg", `http://ffmpeg.org/trac/ffmpeg/wiki/Seeking%20with%20FFmpeg`, 2012, Acessado: 12/06/2013.

[15] FFMPEG, "FFmpeg and x264 Encoding Guide", `https://trac.ffmpeg.org/wiki/x264EncodingGuide`, 2012, Acessado: 10/08/2013.

[16] HAMMACK, B., "CCD: The heart of a digital camera", `http://www.youtube.com/watch?v=wsdmt0De8Hw`, 2012, Acessado: 27/06/2013.

[17] ISO, *ISO/IEC 14496-12:2012: Information technology – Coding of audio-visual objects – Part 12: ISO base media file format*, 2012.

[18] BBC, "Natural World – A Highland Haven", `http://www.bbc.co.uk/programmes/b00p9210` e `http://www.youtube.com/watch?v=VMh_lWvqsI8`, 2009, Acessados: 10/08/2013.

[19] FOUNDATION, B., "Sintel – Third Open Movie", `http://www.sintel.org/` e `http://www.youtube.com/watch?v=eRsGyueVLvQ`, 2010, Acessado: 10/07/2013.

[20] BANKOSKI, J., WILKINS, P., XU, Y., "Technical overview of VP8, an open source video codec for the web." In: *ICME*, pp. 1–6, 2011.

Apêndice A

Comandos do FFmpeg

Esta seção é dedicada a apresentar os comandos do FFmpeg utilizados para converter vídeo de forma paralela. O FFmpeg também fornece uma ferramenta de análise de vídeo chamada FFprobe que foi utilizada na parte inicial do processo.

Em todos os comandos, o arquivo de vídeo original a ser convertido será referenciado por "`INPUT.MP4`", os arquivos intermediários serão "`INPUT-XXX.YYY`" ou "`OUTPUT-XXX.YYY`" (onde "XXX" é qualquer texto possivelmente autoexplicativo e "YYY" a extensão utilizada) e o arquivo final produzido será "`OUTPUT.MP4`".

Algumas opções da linha de comando serão repetidas em vários casos e serão explicadas na primeira ocorrência.

O sinal "$" significa o início de um comando a ser executado num terminal de texto (Sh, Bash etc) e assume-se que os comandos digitados são encotrados sem necessidade de especificar o caminho completo para o binário.

Alguns comandos estão separados por uma "\" para evitar linhas muito longas. Estes devem ser tratados como apenas um comando.

Um comando do FFmpeg é formado por argumentos de entrada e (inseridos antes de um "-i") e argumentos de saída (inseridos após um "-i"), além de um argumento posicional que é o arquivo a ser gerado. Normalmente este é o último argumento do comando.

A.1 Análise de Metadados

Para analisar o conteúdo de um arquivo de vídeo, foi utilizado o comando FFprobe na forma a seguir para produzir na saída padrão do processo um objeto JSON[I] com os metadados do contêiner e dos fluxos de áudio e vídeo.

```
$ ffprobe -v quiet -print_format json -show_streams \
          -show_format -i INPUT.MP4
```

`-v quiet`	Verbosidade: Não escreve nada na saída de erro.
`-i XXX`	Arquivo de entrada.
`-print_format json`	Formato de saída JSON para detalhes do vídeo.
`-show_streams`	Mostra o conteúdo dos streams do vídeo.
`-show_format`	Mostra o conteúdo do contêiner do vídeo.

Tabela A.1: Descrição do comando de análise de metadados.

A.2 Extração do Áudio

Extrair o fluxo de áudio do arquivo é um passo opcional, mas facilita o envio desta parte para ser executado em outro servidor. Note que o arquivo resultante contém apenas áudio e o original não é modificado.

Neste passo, não há decodificação ou codificação. Apenas é executado um reempacotamento do fluxo de áudio um contêiner M4A.

```
$ ffmpeg -i INPUT.MP4 -vn -c:a copy INPUT-audio.M4A
```

[I] JavaScript Object Notation - Serialização de estruturas de dados - `http://www.json.org/`

`-vn`	Não insere vídeo.
`-c:a copy`	Copia o codec de áudio sem transcodificação.
`INPUT-audio.M4A`	Arquivo de saída.

Tabela A.2: Descrição do comando de extração de áudio.

Esse arquivo resultante pode ser convertido separadamente e gerado como '`OUTPUT-audio.M4A`' que será posteriormente unido ao vídeo transcodificado.

A.3 Segmentação e Codificação do Segmento

Com um mesmo comando do FFmpeg é executado o processo de segmentação e transcodificação de um segmento. Este processo exige duas etapas de busca (*seek*), onde a primeira é feita de forma indexada e não atinge necessariamente o tempo pedido e a segunda é somada a primeira e atinge o tempo total da soma delas com precisão.

No comando a seguir, é pedido para o FFmpeg converter 100 quadros de vídeo começando em 5 segundos e utilizando 8 threads de codificação. O comando tem espaço para outras configurações do H.264 que vão variar com a forma de utilização desejada para o vídeo [3].

O arquivo gerado no final irá conter o quadros pedidos codificados em H.264 sem um contêiner e, para tal, é necessário especificar o *Byte Stream Format* definido no Anexo B da especificação H.264.

```
$ ffmpeg -ss 00:00:04.999 -i INPUT.MP4 -ss 00:00:00:001 -an \
         -frames:v 100 -c:v libx264 -bsf:v h264_mp4toannexb \
         -threads 8 ... OUTPUT-123.h264
```

`-ss 00:00:04.999`	Seek rápido de aprox. 5 segundos.
`-ss 00:00:00.001`	Seek lento do que falta para 5 segundos.

`-an`	Não insere áudio.
`-frames:v 100`	Codificar apenas os próximos 100 quadros.
`-c:v libx264`	Codifica vídeo em H.264.
`-bsf:v h264_mp4toannexb`	Seleção de *Byte Stream Format*.
`-threads 8`	Uso de 8 threads de codificação.
`...`	Outras configurações de codificação.
`OUTPUT-123.H264`	Arquivo de saída.

Tabela A.3: Descrição do comando de segmentação.

Dentre as outras configurações possíveis para codificação estão: Tamanho máximo do GOP (`-g`), Taxa de Bits (`-b:v` e `-b:a`), Fator de Qualidade (`-crf`) etc. O FFmpeg também oferece o argumento `-x264opts` para acessar variáveis específicas da biblioteca libx264. Com isso, é possível ajustar detalhes como o número máximo de referências que os quadros com predição podem ter, por exemplo.

A.4 Concatenação dos Segmentos do Vídeo e Áudio

Neste passo, serão unidos os segmentos de vídeo e o áudio já transcodificados num só arquivo MP4. Este passo exige cópia de ambos os fluxos para serem reempacotados sem que haja transcodificação.

Para unir os segmentos de vídeo, é utilizado o protocolo "concat" do FFmpeg que lê uma série de arquivos em sequência como se fosse um só.

```
$ ffmpeg -i "concat:OUTPUT-01.H264|OUTPUT-02.H264|..." \
         -i OUTPUT-audio.M4A -c:v copy -c:a copy OUTPUT.MP4
```

`-i "concat:...|...|..."`	Concatenação de arquivos de entrada: vídeo.
`-i OUTPUT-audio.M4A`	Outro arquivo de entrada: áudio.
`-c:v copy`	Copia o codec de vídeo sem transcodificação.
`-c:a copy`	Copia o codec de áudio sem transcodificação.
`OUTPUT.MP4`	Arquivo de saída.

Tabela A.4: Descrição do comando de concatenação.