ESCOLA
DE ENGENHRARIA
DA UNIVERSIDADE
DO MINHO

M

Mestrado
Engenharia Informática

“Projeto de Computação Avançada
Módulo HTC”

Trabalho Realizado por:

Bruno Santos PG44414

Guilherme Palumbo PG42832

João Silva PG42834

Docente:

António Sousa

Perfil de Ciência de Dados

Computação Avançada

4º/1º Ano, 1º Semestre

Ano letivo 2020/2021

Conteúdo

Capítulo 1

Introdução

1.1 Identificação do Projeto e Objetivos

No âmbito da unidade curricular "Computação Avançada"do perfil "Ciência de Dados", foi-nos proposta a realização de um trabalho prático, cujo objetivo seria instalar e configurar um cluster de HTCondor com o mínimo de 3 nós e utilizar o mesmo na resolução de uma tarefa de processamento. Sendo que para isso, e de forma a comprovar o funcionamento do cluster, deveremos desenvolver uma aplicação resizing de vídeo, capaz de converter um vídeo com resolução fullHD (1080p) num vídeo com resolução SD (720p).

No final deste trabalho prático, deveremos ter uma aplicação capaz de otimizar os recursos disponíveis, correndo tantos processos quantas as unidades de processamento disponíveis. Para além disso, deve ser entregue um relatório onde é descrito todo o trabalho desenvolvido, nomeadamente a descrição do cluster utilizado, os parâmetros de configuração do cluster e as decisões tomadas para melhorar o desempenho da aplicação.

Para a elaboração do presente relatório, optamos por utilizar o software LaTeX.

1.2 Estrutura do Relatório

O presente relatório divide se em vários capítulos, cada capítulo está repartido em secções. A organização dos capítulos está baseada da seguinte forma:

No primeiro capítulo colocamos uma breve introdução, onde identificamos o projeto e os seus objetivos.

No segundo capítulo colocamos uma descrição do cluster HTCondor que implementamos.

O terceiro capítulo descreve de forma sucinta o procedimento de envio de uma tarefa para o cluster implementado.

No quarto capítulo, colocamos alguns testes e resultados do programa.

No quinto capítulo colocamos a conclusão.

Capítulo 2

Cluster HTCondor implementado

O HTCondor é um software de computação distribuída, de código aberto e que permite aumentar o rendimento na computação. É utilizado para trabalhos de computação intensiva. Os utilizadores enviam os trabalhos para o HTCondor, e este coloca-os numa fila, escolhe quando e onde executar os trabalhos com base em políticas, monitora cuidadosamente o progresso e, por fim, informa os utilizadores após a conclusão [1].

No sentido de simular um ambiente de computação de alto rendimento utilizando apenas um único computador pessoal, recorremos ao software de código aberto Vagrant [2] e ao software de virtualização VirtualBox [3]. O cluster HTCondor implementado é constituído por três máquinas virtuais, condor-1, condor-2 e condor-3. A máquina física (host) onde as máquinas virtuais estão instaladas usa o sistema operativo Linux Ubuntu 20.04.1 LTS. As máquinas virtuais (guest) usam o sistema operativo Linux CentOS 8.

Como supramencionado, para instalar as máquinas virtuais e implementar o nosso cluster com três nodos, recorremos ao software Vagrant. Para tal, foi necessário construir um ficheiro Vagrantfile. O ficheiro Vagrantfile (todos os ficheiros usados no projeto são enviados com o presente relatório) tem como objetivo descrever e provisionar (vagrant provision) as máquinas virtuais necessárias para o projeto [4]. No respetivo ficheiro, para cada máquina, atribuímos um endereço ip, um nome, o número de cpu a utilizar e, recorrendo ao ansible, executamos um playbook. No playbook encontram-se todo o software e os serviços necessários. Apenas para a máquina condor-1 é feito o upload de um arquivo da máquina host, contendo os vídeos a converter.

Os ficheiros de configuração de cada máquina, executados através do playbook, encontram-se na pasta *condor_configs*. Concebidos os ficheiros necessários, através do comando *vagrant up*, as três máquinas são iniciadas. Através do comando *vagrant provision*, os vídeos a converter são carregados para a máquina condor-1.

Na figura 2.1 encontra-se representado o esquemático do cluster implementado.

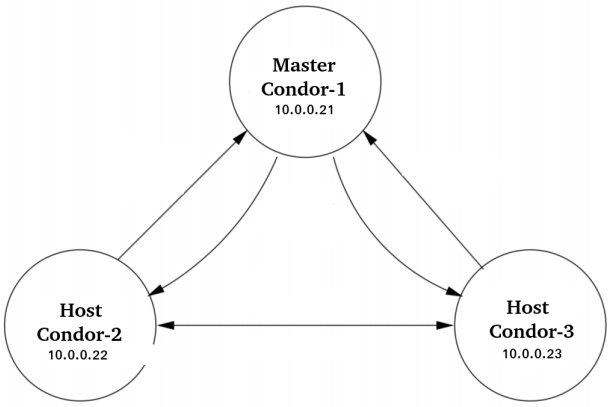

Figura 2.1: Esquema do cluster

Nas seguintes subsecções apresentamos os serviços e softwares instalados em cada máquina

virtual.

2.1 Condor-1 - Master

O nodo (máquina virtual) condor-1 é a máquina Master do nosso condor pool. O seu endereço é 10.0.0.21.

A presente máquina, através das configurações inseridas, é a única que pode submeter um *job* e gerir os vários jobs entre as restantes máquinas. Para além disso, ainda os pode executar. Apenas possui 1 CPU e possui 2048 megabytes de memória. Como referido no atual capítulo, possui a versão 3.0.38 do sistema operativo centOS 8.

Através da máquina condor-1 é possível realizar vários serviços, sendo instalados os seguintes softwares/programas: python, ansible, wget, condor, utils, docker, ffmpeg e java.

É através da máquina condor-1 que submetemos o nosso ficheiro dag.

2.2 Condor-2 - Host

O nodo (máquina virtual) condor-2 é uma das duas máquinas host do nosso condor pool. O seu endereço é 10.0.0.22.

Em detrimento da máquina Master, a máquina condor-2 apenas executa Jobs. A presente máquina encontra-se em estreita comunicação com a máquina condor-1 através das configurações passadas. Possui 1 CPU e 2048 megabytes de memória. Como referido no atual capítulo, possui a versão 3.0.38 do sistema operativo centOS 8.

Os softwares/programas instalados são: python, ansible, wget, condor, utils, ffmpeg e docker.

2.3 Condor-3 - Host

O nodo (máquina virtual) condor-3 é uma das duas máquinas host do nosso condor pool. O seu endereço é 10.0.0.23.

A máquina condor-3, como a condor-2, apenas executa Jobs. A presente máquina encontra-se em estreita comunicação com a máquina condor-1 através das configurações passadas. Possui 1 CPU e 2048 megabytes de memória. Como referido no atual capítulo, possui a versão 3.0.38 do sistema operativo centOS 8.

Os softwares/programas instalados são: python, ansible, wget, condor, utils, ffmpeg e docker.

Capítulo 3

Envio da tarefa para o Cluster

O objetivo do presente projeto é reduzir o tempo necessário para a conversão de um vídeo, por exemplo, partindo o vídeo original em vários segmentos, fazer a conversão de cada um dos segmentos e juntá-los todos para o vídeo de resultado.

No sentido de o presente programa ser capaz de fazer *resizing* de qualquer vídeo, e de partir em vários segmentos de acordo com o utilizador, fizemos um programa em java que recolhe as respetivas informações e constrói todos os ficheiros necessários para a respetiva tarefa.

O programa em java pergunta ao utilizador o nome do vídeo a converter, a qualidade do vídeo atual e final e o número de segmentos em que o vídeo é partido. Na figura seguinte é apresentado um exemplo.

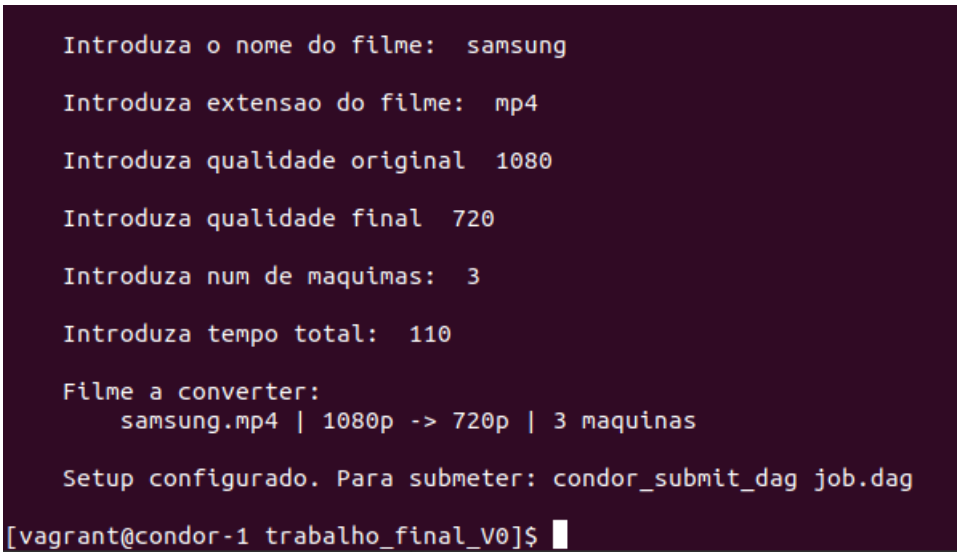

Figura 3.1: configuração do job

No exemplo apresentado na figura 3.1, o utilizador digitou o nome do vídeo a converter, samsung.mp4, a qualidade original, 1080p, a qualidade final, 720p e o número de segmentos em que o vídeo deve ser dividido, 3.

Depois de introduzidas as informações, o programa devolve todos os ficheiros necessários para a submissão do job. Na figura seguinte é apresentado todos os ficheiros concebidos após o programa java.

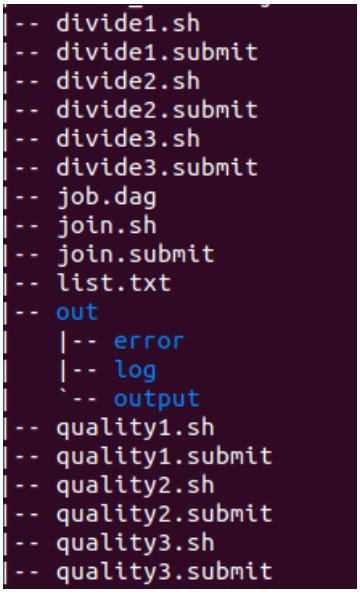

Figura 3.2: Ficheiros formados após programa java

Qualquer *job* submetido no nosso cluster é composto por três partes. Na primeira parte o vídeo é dividido em segmentos de igual período de tempo. A segunda parte tem como objetivo alterar a qualidade de cada segmento e por fim, a terceira parte, tem como finalidade juntar os três segmentos de menor qualidade num só.

3.1 Divisão do video

Como supracitado, o utilizador, no programa em java, decide em quantos segmentos o vídeo é dividido. Para usar a máxima computação do nosso condor pool, recomenda-se que o vídeo seja dividido em três segmentos, uma vez que apenas possuímos três máquinas condor.

Para que o vídeo seja dividido em três segmentos, são precisos três jobs, cada um responsável por construir cada segmento. Para cada job é necessário um ficheiro de descrição e o respetivo executável.

Os três *jobs* responsáveis pela divisão do vídeo em três partes iguais são bastantes similares, mudando apenas no script em bash o tempo inicial em que se começa a dividir.

Os ficheiros denotados por divide1 correspondem ao job responsável por construir o primeiro segmento, divide2 o segundo segmento e assim sucessivamente.

Nas duas figuras seguintes são exibidos um ficheiro de descrição de um job e o respetivo programa que o acompanha. O job em questão é responsável por construir o primeiro segmento. O ficheiro de submissão corresponde ao ficheiro *divide1.submit* e o programa está presente no ficheiro *divide1.sh*.

```
1 Universe = vanilla
2 executable = divide1.sh
3
4 log = out/log/log-divide1-$(Process).log
5 output = out/output/out-divide1-$(Process).txt
6 error = out/error/erro-divide1-$(Process).txt
7
8 request_cpus   = 1
9 request_memory = 1024
10 Rank = memory
11
12 should_transfer_files  = Yes
13 when_to_transfer_output = ON_EXIT
14 transfer_input_files  = videos/samsung.mp4
15 queue
```

Figura 3.3: Ficheiro de descrição divide1

A imagem anterior mostra a submissão do programa divide1.sh como um *job* do universo Vanilla, ou seja, enfileira o programa divide1.sh para execução em algum lugar do nosso condor pool. Como é possível a ocorrência de erros, nos ficheiros de descrição digitamos que os erros e as várias informações acerca do job, serão enviados para a pasta *out*. O job requer máquinas com pelo menos 1024 MB de memória física e um cpu de apenas um núcleo.

Através de *transfer_input_files*, informamos o HTCondor qual o ficheiro de entrada é preciso enviar juntamente com o script e o ficheiro de descrição.

Finalmente, a instrução *queue* informa que o HTCondor terminou de descrever o trabalho e deve enviá-lo para a fila para o processamento.

Na imagem seguinte é possível observar o programa que acompanha o anterior ficheiro de descrição.

```
1 #!/bin/bash
2 tempo_corte=36
3 nome_video_entrada=samsung.mp4
4 hora=0
5 minuto=0
6 segundo=0
7 ffmpeg -i samsung.mp4 -ss 0:0:0 -t 36 1.mp4
```

Figura 3.4: Executavel divide1

O executável em bash, através do tempo de corte e do tempo inicial, corta o vídeo *samsung.mp4* em 36 segundo a partir do segundo 0. Isso é possível recorrendo ao comando *ffmpeg*. O ficheiro de descrição e o executável em bash dos restantes jobs são bastantes similares.

Da submissão de cada job na presente parte, o output corresponde a um segmento do vídeo original. Deste modo, do job Divide1 resulta o segmento 1.mp4, do job Divide2 o segmento 2.mp4 e assim sucessivamente.

Os três jobs, através de um DAG retratado nas seções subsequentes do relatório, são enviados simultaneamente. Deste modo, o nosso condor pool apresenta as três máquinas concorrentemente a trabalhar.

3.2 Redução da qualidade

À semelhança do que acontece na divisão do vídeo, em que o utilizador decide em quantos segmentos o vídeo é dividido, na redução da qualidade o utilizador indica qual a resolução inicial do vídeo e qual a resolução que pretende. Neste caso, o objetivo é converter um vídeo com resolução fullHD (1080p) num vídeo com resolução SD (720p).

Uma vez que o vídeo foi divido em três segmentos (subsecção anterior), são necessários três jobs para reduzir a qualidade do vídeo, sendo que para cada job, e à semelhança do que acontece na divisão do vídeo, é necessário um ficheiro de descrição e o respetivo executável.

Os três jobs responsáveis pela redução da qualidade do vídeo são exatamente iguais.

Os ficheiros denotados por quality correspondem ao job responsável por reduzir a qualidade dos segmentos, sendo que quality1 reduz a qualidade do 1.mp4, o quality2 reduz a qualidade do 2.mp4 e o quality3 reduz a qualidade do 3.mp4.

Nas Figuras 3.5 e 3.6 está o ficheiro de descrição de um job e o respetivo programa que o acompanha. O job em questão é responsável por reduzir a qualidade do primeiro segmento (1.mp4). O ficheiro de submissão corresponde ao ficheiro quality1.submit e o programa está presente no ficheiro quality1.sh.

```
1 Universe = vanilla
2 executable = quality1.sh
3
4 log = out/log/log-quality1-$(Process).log
5 output = out/output/out-quality1-$(Process).txt
6 error = out/error/erro-quality1-$(Process).txt
7
8 request_cpus = 1
9 request_memory = 1024
10
11 should_transfer_files  = Yes
12 when_to_transfer_output = ON_EXIT
13 transfer_input_files  = 1.mp4
14 queue
```

Figura 3.5: Ficheiro de descrição quality1

À semelhança do que acontece na divisão dos segmentos, é possível a ocorrência de erros e sempre que isso aconteça estes serão enviados para a pasta out. Cada um dos jobs, para reduzir a qualidade dos segmentos, requer máquinas com pelo menos 1024 MB de memória física e um cpu com apena um núcleo.

Para informar o HTCondor qual o ficheiro de entrada que é necessário enviar juntamente com o script e o ficheiro descrição usamos o transfer_input_files.

Por último, temos a instrução queue que informa o HTCondor que o trabalho foi descrito e deve enviá-lo para a fila para processamento.

O executável em bash (figura 3.6), através do comando ffmpeg, reduz a qualidade do vídeo para a resolução pretendida. Inicialmente começamos por definir qual o vídeo de entrada (1.mp4), de seguida definimos qual resolução pretendida (720) e por último damos um nome output (1_720.mp4).

Os três jobs, através de um DAG retratado nas secções subsequentes do relatório, são enviados simultaneamente. Deste modo, o nosso condor pool apresenta as três máquinas concorrentemente a trabalhar.

```
1 #!/bin/bash
2 ffmpeg -i ./1.mp4 -vf scale=-1:720 1_720.mp4
```

Figura 3.6: Executavel quality1.sh

3.3 Junção dos segmentos

A última parte corresponde à junção dos segmentos renderizados resultantes da secção anterior. Ou seja, depois de dividir o vídeo em vários segmentos e de reduzir a qualidade a cada, é necessário juntá-los num único segmento.

Para realizar esta parte é apenas necessário um único job. O ficheiro de descrição é denotado por *join.submit* e o executável por *join.sh*. Na figura seguinte é apresentado o ficheiro de descrição gerado pelo programa java para a presente tarefa.

```
1 Universe = vanilla
2 executable = join.sh
3
4 log = out/log/log-join-$(Process).log
5 output = out/output/out-join-$(Process).txt
6 error = out/error/erro-join-$(Process).txt
7
8 request_cpus = 1
9 request_memory = 1024
10
11 should_transfer_files   = Yes
12 when_to_transfer_output = ON_EXIT
13 transfer_input_files   = list.txt, 1_720.mp4, 2_720.mp4, 3_720.mp4
14 queue
```

Figura 3.7: Ficheiro de descrição join.submit

O universo de execução continua a ser o Vanila. O executável join.sh é enviado para uma máquina do pool que apresenta pelo menos um cpu e uma memória superior a 1024 megabytes. Todos os passos decorrentes do presente job (logs) é colocado na pasta out/log, assim como os erros na pasta out/error.

Para executar o presente job são necessários os segmentos renderizados no passo anterior e de um ficheiro, que denominamos por *list.txt*, que contem o nome dos vídeos renderizados. No ficheiro de descrição do job, através da instrução *should_transfer_file = yes* explicitamos que o job pode transferir ficheiros entre as máquinas e, na instrução seguinte, *transfer_input_files = list.txt, 1_720.mp4, 2_720.mp4, 3_720.mp4*, explicitamos quais os ficheiros a transferir.

O job, depois de concluído, devolve como output a concatenação dos três segmentos. Através da instrução *when_to_transfer_output = ON_EXIT* explicitamos que o output apenas deve ser devolvido no fim da concatenação.

O script em bash (executável) descrito no ficheiro de descrição anterior, é apresentado na seguinte imagem.

```
1 #!/bin/bash
2 ffmpeg -f concat -safe 0 -i list.txt -c copy samsung_720.mp4
```

Figura 3.8: Executável join.sh

Do mesmo modo que nos pontos anteriores, usamos o comando *ffmpeg* para a concatenação dos três segmentos. O ficheiro *list.txt* é inserido no comando ffmpeg. O comando ffmpeg concatena todos os segmentos presentes no ficheiro *list.txt* e devolve como output o vídeo concatenado denominando-o *samsung_720.mp4*.

3.4 DAG

Depois de o programa java inicial gerar todos os ficheiros e executáveis, retratados nas secções anteriores, responsáveis por cada job em cada uma das três partes, é necessário colocar uma ordem na execução de cada job. Por exemplo, o job responsável pela concatenação dos segmentos renderizados não pode ser executado antes de os jobs responsáveis pela divisão dos segmentos terem findado.

Para colocar uma determinada ordem na execução dos jobs, utilizamos o DAGMan (Directed Acyclic Graph Manager). O DAGman gere as dependências entre jobs. Assim sendo, o programa inicial java, de acordo com as preferências do utilizador, gera um DAG (directed acyclic graph) onde é colocado um conjunto de programas em que a entrada, saída ou execução de um ou mais programas depende de outros programas.

Voltando ao exemplo das secções anteriores, em que o vídeo a ser renderizado é denotado por *samsung.mp4* e o utilizador pretende dividir em três partes. O ficheiro DAG gerado pelo programa java para as presentes preferências encontra-se na figura seguinte.

```
1 JOB Divide1 divide1.submit
2 JOB Divide2 divide2.submit
3 JOB Divide3 divide3.submit
4 JOB Quality1 quality1.submit
5 JOB Quality2 quality2.submit
6 JOB Quality3 quality3.submit
7 JOB Join join.submit
8
9 PARENT Divide1 CHILD Quality1
10 PARENT Divide2 CHILD Quality2
11 PARENT Divide3 CHILD Quality3
12 PARENT Quality1 Quality2 Quality3  CHILD Join
```

Figura 3.9: Ficheiro DAG

Para as preferências do utilizador, serão sete os jobs a serem enviados. Três responsáveis pela divisão em três segmentos, três pela renderização de cada segmento e um job responsável pela concatenação dos segmentos renderizados.

No ficheiro DAG da figura anterior, nas linhas 1 até 7 são explicitados os sete jobs. Nas linhas 9, 10 e 11 é dito que o job Quality1 só pode começar a execução após o job Divide1 terminar, que o job Quality2 só pode começar a execução após o job Divide2 terminar e que o job Quality3 só pode começar a execução após o job Divide3 terminar. Por fim, na linha 12, é explicitado que a concatenação dos segmentos renderizados só pode ser feita após a renderização dos respetivos segmentos estiver concluída.

Na figura seguinte é apresentado um diagrama gráfico que mostra as dependências entre os jobs presentes no ficheiro DAG.

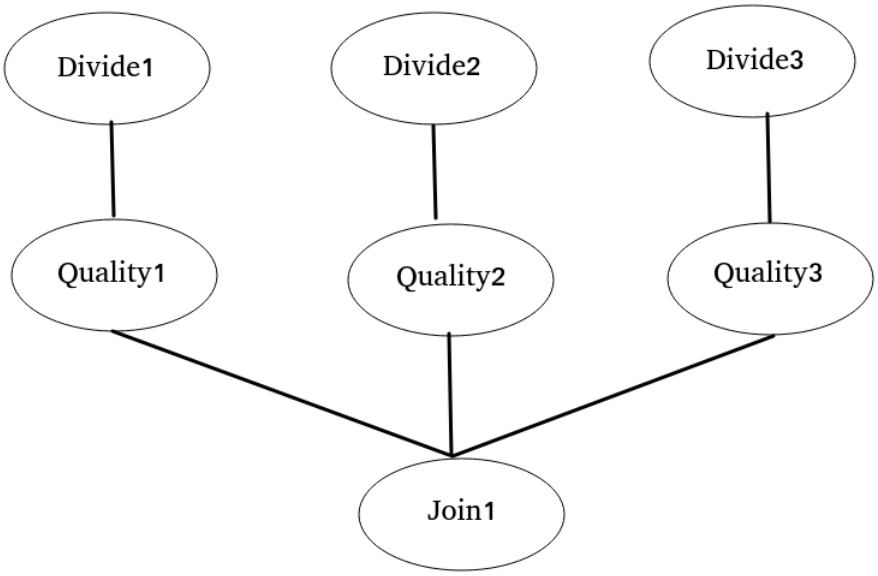

Figura 3.10: Diagrama DAG

Por fim, para submeter o DAG, com todas as dependências entre os jobs, na máquina Condor-1 (MASTER), utilizamos o seguinte comando: *condor_submit_dag job.dag*.

Capítulo 4

Análise de resultados

O presente capítulo tem como objetivo apresentar os resultados obtidos no presente projeto.

Para que o utilizador consiga renderizar um vídeo através do nosso condor pool, precisa de enviar o vídeo a renderizar e o programa em java para a máquina condor-1 (MASTER). De seguida executa o programa java e insere qual o vídeo a renderizar e as restantes preferências, como é mostrado na figura 3.1. Finda a execução do programa em java, todos os ficheiros necessários para os jobs são construídos, incluindo o DAG. Nas imagens seguintes são baseadas nas preferências do utilizador presentes na imagem 3.1, ou seja, o vídeo a renderizar é denotado por samsung.mp4 e será dividido em três segmentos.

Para submeter a tarefa completa, submetemos o ficheiro dag, denominado job.dag, através do seguinte comando: *condor_submit_dag job.dag*. Na figura seguinte é mostrado a submissão do ficheiro dag.

```
[vagrant@condor-1 trabalho_final_V0]$ condor_submit_dag job.dag

------------------------------------------------------------------------
File for submitting this DAG to HTCondor           : job.dag.condor.sub
Log of DAGMan debugging messages                 : job.dag.dagman.out
Log of HTCondor library output                     : job.dag.lib.out
Log of HTCondor library error messages             : job.dag.lib.err
Log of the life of condor_dagman itself          : job.dag.dagman.log

Submitting job(s).
1 job(s) submitted to cluster 222.
------------------------------------------------------------------------
```

Figura 4.1: Submissão job.dag

Após a submissão, são criados os ficheiros de log onde se encontra informações acerca da respetiva submissão.

Para consultar o estado do nosso cluster, usamos o seguinte comando: *condor_status*. Na figura seguinte é apresentado o output do respetivo comando.

```
[vagrant@condor-1 trabalho_final_V0]$ condor_status
Name          OpSys      Arch   State     Activity LoadAv Mem   ActvtyTime

condor-1      LINUX      X86_64 Claimed   Busy      0.000 1816  0+00:00:03
condor-2      LINUX      X86_64 Claimed   Busy      0.000 1827  0+00:00:03
condor-3      LINUX      X86_64 Claimed   Busy      0.000 1827  0+00:00:03

                Machines Owner Claimed Unclaimed Matched Preempting  Drain

  X86_64/LINUX        3     0       3         0       0          0      0

         Total        3     0       3         0       0          0      0
```

Figura 4.2: Output condor_status

Através da imagem anterior é possível denotar que as três máquinas estão a executar jobs concorrentemente da tarefa em questão. Cada um dos jobs responsáveis pela segmentação do vídeo estão a ser executados pelas três máquinas presentes no nosso condor pool. Para ver informações acerca de jobs em fila presentes no nosso cluster, executamos o comando *condor_q*. O output do referido comando encontra-se na figura seguinte.

```
-- Schedd: condor-1 : <10.0.0.21:9618?... @ 12/19/20 20:48:21
OWNER    BATCH_NAME      SUBMITTED   DONE   RUN    IDLE  TOTAL JOB_IDS
vagrant job.dag+230   12/19 20:45      _      2       _      1 231.0 ... 232.0

Total for query: 2 jobs; 0 completed, 0 removed, 0 idle, 2 running, 0 held, 0 suspended
Total for vagrant: 2 jobs; 0 completed, 0 removed, 0 idle, 2 running, 0 held, 0 suspended
Total for all users: 2 jobs; 0 completed, 0 removed, 0 idle, 2 running, 0 held, 0 suspended

[vagrant@condor-1 trabalho_final_V0]$
```

Figura 4.3: Output condor_q

Da imagem anterior, é possível visualizar, que nesse momento, no condor pool existem 2 jobs, cujos ID's são 231 e 232, encontrando-se os dois em execução.

Finda a tarefa, ou seja, findo os sete jobs presentes no ficheiro job.dag, o vídeo samsung.mp4, cuja qualidade inicial é de 1080p, é enviado para a máquina condor-1 (Master) com a qualidade requerida, 720p. Para ver se a tarefa foi bem executada podemos visualizar os ficheiros *job.dag.dagman.out* e *job.dag.dagman.log*. Na figura seguinte encontra-se um excerto do ficheiro *job.dag.dagman.out*.

```
12/19/20 20:51:26 DAG status: 0 (DAG_STATUS_OK)
12/19/20 20:51:26 Of 7 nodes total:
12/19/20 20:51:26  Done     Pre   Queued    Post   Ready   Un-Ready   Failed
12/19/20 20:51:26   ===      ===      ===     ===     ===        ===      ===
12/19/20 20:51:26     7        0        0       0       0          0        0
12/19/20 20:51:26 0 job proc(s) currently held
12/19/20 20:51:26 DAGMan Runtime Statistics: [ EventCycleTimeStd = 0.02065673756282354; EventCycleTimeMax =
140995; EventCycleTimeSum = 0.3653814792633057; EventCycleTimeCount = 66.0; SleepCycleTimeStd = 0.1211666373
429688; SubmitCycleTimeCount = 67.0; LogProcessCycleTimeAvg = 0.0002252622084184127; LogProcessCycleTimeCoun
92602539; SubmitCycleTimeSum = 0.3511703014373779; SubmitCycleTimeAvg = 0.005241347782647432; SleepCycleTime
9835357666; SleepCycleTimeAvg = 4.997798450065382; SleepCycleTimeMax = 5.035477876663208; SleepCycleTimeCoun
12/19/20 20:51:26 Wrote metrics file job.dag.metrics.
12/19/20 20:51:26 Metrics not sent because of PEGASUS_METRICS or CONDOR_DEVELOPERS setting.
12/19/20 20:51:26 DAGMan Runtime Statistics: [ EventCycleTimeStd = 0.02065673756282354; EventCycleTimeMax =
140995; EventCycleTimeSum = 0.3653814792633057; EventCycleTimeCount = 66.0; SleepCycleTimeStd = 0.1211666373
429688; SubmitCycleTimeCount = 67.0; LogProcessCycleTimeAvg = 0.0002252622084184127; LogProcessCycleTimeCoun
92602539; SubmitCycleTimeSum = 0.3511703014373779; SubmitCycleTimeAvg = 0.005241347782647432; SleepCycleTime
9835357666; SleepCycleTimeAvg = 4.997798450065382; SleepCycleTimeMax = 5.035477876663208; SleepCycleTimeCoun
12/19/20 20:51:26 **** condor_scheduniv_exec.230.0 (condor_DAGMAN) pid 9624 EXITING WITH STATUS 0
```

Figura 4.4: *Ficheiro job.dag.dagman.out*

Através da imagem anterior, concluímos que a tarefa foi bem executada, uma vez que o STATUS de saída teve o valor de 0. Também concluímos que dos sete jobs, os sete estão marcados como *Done*.

Na imagem seguinte encontra-se o vídeo original, *samsung.mp4*, e o vídeo renderizado, *samsung_720.mp4*. É possível notar que, o vídeo original possui, aproximadamente, 41 MB e o vídeo renderizado 32 MB. Esta diferença é devido a que o vídeo renderizado apresenta uma resolução de 720p, inferior aos 1080p do vídeo original.

```
-rw-rw-r-- 1 vagrant vagrant 40685366 Dec 19 20:45 samsung.mp4
-rw-r--r-- 1 vagrant vagrant 32021245 Dec 19 20:51 samsung_720.mp4
drwxrwxr-x 2 vagrant vagrant       27 Dec 19 21:29 videos
```

Figura 4.5: Video original e renderizado - comparação

Capítulo 5

Conclusão

O principal objetivo deste trabalho era sensibilizar e motivar os alunos para a conceção e desenvolvimento de um projeto onde o foco era instalar e configurar um cluster HTCondor com o mínimo de 3 nós e utilizar o mesmo na resolução de uma tarefa de processamento. Sendo que para isso, e de forma a comprovar o funcionamento do cluster, desenvolvemos uma aplicação resizing de vídeo, capaz de converter um vídeo com resolução fullHD (1080p) num vídeo com resolução SD (720p).

Através da Computação de Alta Capacidade (High Throughput Computing) podemos processar várias tarefas de forma independente entre si, aumentando assim a capacidade da máquina, sem comprometer a segurança e confiabilidade do sistema.

Com este projeto, apesar de sua dimensão ser em pequena escala, permite-nos compreender um pouco mais o impacto de HTC sobre os processos de grande escala e benefícios que as organizações obtêm ao utilizarem HTC, que comumente necessitam grande capacidade de computação durante longos períodos de tempo.

Bibliografia

[1] What is HTCondor? https://research.cs.wisc.edu/htcondor/description.html. [Online; consultado 11-dez-2020].

[2] Vagrant - instalacao/. https://www.vagrantup.com/. [Online; consultado 11-dez-2020].

[3] Virtual Box - instalacao/. https://www.virtualbox.org/. [Online; consultado 12-dez-2020].

[4] Vagrantfile. https://www.vagrantup.com/docs/vagrantfile. [Online; consultado 14-dez-2020].