FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Processamento concorrente de Vídeo no MEO Kanal

João Carlos Santos

PREPARAÇÃO DA DISSERTAÇÃO

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Nome do Orientador

13 de Fevereiro de 2016

Processamento concorrente de Vídeo no MEO Kanal

João Carlos Santos

Mestrado Integrado em Engenharia Informática e Computação

13 de Fevereiro de 2016

Resumo

Hoje em dia existem várias soluções para o armazenamento e reprodução de vídeos na cloud, como é o caso do Youtube, Dailymotion, Vimeo e MEO Kanal. O esforço computacional associado ao armazenamento, classificação, descodificação e reprodução destes conteúdos é considerável, quer no que toca ao processamento dos vídeos, quer às operações de transferência de dados. Tendo em conta um sistema multi-utilizador em que a duração dos vídeos submetidos é bastante variável, o processamento destes não pode ser atribuído a simples filas de espera. Imaginemos um cenário em que um utilizador submete um vídeo com uma duração de uma hora, seguido de outro utilizador que submete um vídeo com uma duração de um minuto. Se o processamento fosse atribuído de acordo com uma fila de espera, o utilizador que submeteu o vídeo de 1 minuto teria que esperar pelo total processamento do vídeo do outro utilizador para poder ver o seu conteúdo online, o que não se traduz numa boa QoS para o utilizador. A solução proposta a este problema é a implementação e teste de várias técnicas de escalonamento de "trabalhos"que definam prioridades de acordo com parâmetros de QoS, sendo depois escolhido o algoritmo que maximize esta mesma QoS. Assume-se então como objetivo aproximar o tempo que demoraria a processar cada vídeo ao tempo que este processo demoraria sem concorrência. Para fazer um escalonamento equilibrado, os ficheiros serão divididos em "chunks"com a mesma duração para serem processados concorrentemente, sendo que no caso de não haver sobrecarga (haverem processadores desocupados), poderão ser processados em ainda menos tempo do que o esperado. Os dados para testes são ficheiros de vídeo de diferentes formatos e durações, de modo a tentar simular as condicões com que lidam os sistemas da MEO ao receber vídeos para processamento. O resultado esperado é um sistema mais equilibrado no que toca à distribuição do poder de processamento por cada utilizador e mais rápido devido à paralelização do processo.

Abstract

Nowadays there is a variety of solutions for the storage and reproduction of video content in the cloud, such as Youtube, Dailymotion, Vimeo and MEO Kanal. The computational effort associated with the storage, classification, decoding and reproduction of these contents is considerable, whether in terms of computational effort in processing these videos or in terms of data transfer operations. When taking into account a multi-user system where the duration of submitted videos is an unknown variable, their processing cannot be attributed to simple queues. Imagining a scenario where a user submits a video with a duration of one hour, followed by another submission by another user of a video with a duration of 1 minute, if the processing of these videos was scheduled using a queue, the user that submitted the 1 minute video would have to wait for the processing of the 1 hour video plus the processing of his own video to be able to see it online. This does not translate into a good Quality of Service (QoS) to the user. The proposed solution to the problem is the implementation and testing of several scheduling techniques for *jobs* that define priorities in processing according to QoS parameters. After testing, one technique is chosen as the best solution to this problem. The main goal is to minimize the difference between the time it takes to process a video in the system and the time it would take to process the same video if there were no other videos in the queue. In order to balance the processing of videos efficiently, they are first divided into different c̈hunksẅith the same duration. This allows the different chunks to be processed concurrently, being that in a case where there is no overload (there are free processing cores), videos are processed even faster than expected. The data on which the tests are performed is a collection of video files with different formats and durations. These videos were selected to best simulate the conditions that MEO's systems deal with every day. The results is a better balanced system in accounts to the distribution of processing power per user and a faster system due to the parallelization of the process.

"Some people don't like change, but you need to embrace change if the alternative is disaster."

Elon Musk

Conteúdo

CONTEÚDO

Lista de Figuras

LISTA DE FIGURAS

Lista de Tabelas

Abreviaturas e Símbolos

QoS	Quality of Service / Qualidade do Serviço
Rank Hybd	Algoritmo Rank Hybrid
OWM	Algoritmo Online Workflow Management
FDWM	Algoritmo Fairness Dynamic Workflow Scheduling
MW-DBS	Algoritmo Multi-Workflow Deadline-Budget Scheduling
DAG	Grafo Acíclico Direccionado
LAN	Local Area Network / Rede local
WWW	*World Wide Web*

Capítulo 1

Introdução

1.1 Contexto/Enquadramento

Hoje em dia cada vez mais se recorre a serviços *cloud* para o alojamento e reprodução de conteúdos multimédia. São exemplos deste tipo de serviço o Youtube, Dailymotion, Vimeo e o MEO Canal. Nos bastidores, estes serviços consistem em *clusters* de servidores interligados por uma rede onde os conteúdos multimédia são processados e alojados para futuro consumo. Na maior parte destes serviços como o MEO Kanal, os vídeos têm primeiro que passar por uma fase de pré-processamento para uniformizar uma série de propriedades destes conteúdos e prepará-los para serem reproduzidos em várias resoluções. Neste tipo de serviços, uma das medidas mais importantes de avaliação de performance é a QoS, especialmente o tempo que um utilizador tem que esperar desde o momento em que submete o seu vídeo para o serviço até ao momento em que esse mesmo vídeo fica disponibilizado no serviço. Neste momento, o MEO Kanal dispõe de um mecanismo simplista, pouco optimizado e sequencial para a fase de pré-processamento. Este mecanismo tem um grande potencial para optimização que pode levar a grandes ganhos em termos de QoS para o cliente. Nesta dissertação é proposto um método de transformação da fase de pré-processamento num processo concorrente, dividindo os ficheiros recebido em *chunks* para que possam ser processados concorrentemente, e são analisadas várias técnicas de escalonamento do processamento para determinar qual destas consegue mais aproximar o tempo de processamento de um vídeo submetido por um utilizador num sistema partilhado ao tempo que demoraria o processamento caso apenas o vídeo do utilizador tivesse que ser processado. Esta dissertação está inserida no âmbito do SapoLabs, inserido no projeto "ELEGANT : MEO KanaL vidEo processinG optimizAtioN by Parallel compuTing"

1.2 Problema

Devido ao processamento sequencial de vídeos que chegam ao MEO Kanal, existem várias situações em que a QoS de um cliente que submete um vídeo poderia ser melhorada. O MEO Kanal conta neste momento com três filas de processamento:

- Uma fila para vídeos com duração inferior a 6 minutos;
- Uma fila para vídeos com duração superior;
- Uma fila para clientes VIP;

Com esta estrutura, podem acontecer cenários como um em que um utilizador X faz um upload de um vídeo com 7 minutos pouco depois de um utilizador Y fazer um upload de um vídeo com 2 horas. Nesta situação, o utilizador X terá que esperar pelo processamento total do vídeo do utilizador Y, o que leva a um tempo de processamento aparente do seu vídeo muito elevado. Este tipo de situação pode ser resolvida com uma implementação concorrente do sistema que pode por sua vez ser optimizada para ser justa na atribuição de poder de processamento aos vídeos de cada utilizador.

1.3 Motivação e Objetivos

A principal motivação desta dissertação é conseguir melhorar significativamente a QoS dos utilizadores do MEO Kanal, que se deverá traduzir em esperas menos demoradas no que toca ao processamento dos vídeos submetidos neste serviço. Para o fazer, foram delineados os seguintes objetivos:

- Criação de uma aplicação que divida os ficheiros submetidos em vários *chunks* para permitir o processamento concorrente dos mesmos.
- Implementação de vários algoritmos de escalonamento direcionados a sistemas onde novos trabalhos surgem ao longo do tempo.
- Simulação de cada um dos algoritmos num computador pessoal, de modo a comparar a performance dos algoritmos implementados.
- Simulação de cada um dos algoritmos no simulador SIMGRID, emulando o cluster da MEO, comparando os resultados de performance obtidos aos resultados anteriores.
- Escolha da técnica de escalonamento ótima e finalização da aplicação

1.4 Estrutura da Dissertação

Para além da presente secção, esta monografia contém mais 5 capítulos. No capítulo 2 é descrito o estado da arte e são demonstradas as várias técnicas de escalonamento que podem ser utilizadas na aplicação a desenvolver. É também apresentada a plataforma utilizada para a simulação do *cluster* da MEO. No capítulo 3 é descrito o problema de uma forma mais detalhada, bem como a metodologia utilizada para o resolver. No capítulo 4 é exposta a implementação da solução. No capítulo 5 são expostos os resultados das experiências realizadas. No capítulo 6 é determinada a melhor solução encontrada e se a solução conseguiu cumprir os objetivos estipulados.

Capítulo 2

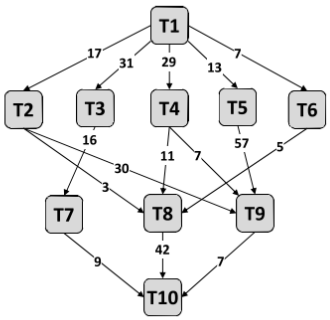
Revisão Bibliográfica

2.1 Workflow

Um *workflow* consiste num conjunto de tarefas com dependências lógicas ou de dados que podem ser executadas por diferentes nós num sistema computacional. Para obter uma execução eficiente de um conjunto de *workflows* é necessária uma técnica de escalonamento que minimize o *Turnaround Time*.

Um típico workflow científico pode ser representado por um grafo acíclico direcionado (DAG). Num DAG, nós representam tarefas e arestas direcionadas representam dependências de execução e a quantidade de comunicação entre nós. Um *workflow* é modelado através de um DAG $G = (V, E)$, onde $V = nj, j = 1..v$ representa o conjunto de v tarefas a executar e E representa o conjunto de arestas direcionadas e representa requisitos de comunicação entre tarefas [ABAB13].

Num dado DAG, uma tarefa sem predecessores é denominada tarefa de entrada e uma tarefa sem sucessores é denominada tarefa de saída. Assumimos que um DAG tem exatamente uma tarefa de entrada e uma tarefa de saída. Se um DAG tem múltiplas tarefas de entrada ou de saída, uma tarefa de entrada ou saída falsa com peso 0 e 0 arestas é adicionada ao grafo [AB]



Task	P1	P2	P3
T1	22	21	36
T2	22	18	18
T3	32	27	43
T4	7	10	4
T5	29	27	35
T6	26	17	24
T7	14	25	30
T8	29	23	36
T9	15	21	8
T10	13	16	33

Figura 2.1: Exemplo de um workflow

Um exemplo de um DAG pode ser visto na Figura 2.1, que representa um DAG e um sistema de computação com 3 processadores e os correspondentes custos de comunicação e computação.

Na figura 2.1, o peso de cada aresta representa o seu custo de comunicação médio e os número na tabela o tempo de computação em cada um dos 3 processadores. [ABAB13].

2.2 Métricas de Desempenho

Para quantificar a performance de uma técnica de escalonamento de *workflows* existem uma série de métricas que têm sido aplicadas pelos investigadores desta área. Nesta secção é explicado o seu significado.

2.2.1 Makespan

Makespan ou *Schedule Length* é o tempo entre o início da execução do nó de entrada de um *workflow* e o final da execução do nó de saída desse mesmo *workflow*. É definido por: $makespan = AFT(nsada) - AST(nentrada)$ onde AFT(nsaída) simboliza o *Actual Finish Time* (tempo actual de finalizaçao) do nó de saída e AST(nentrada) simboliza o *Actual Start Time* (tempo actual de começo) do nó de entrada. [ABAB13]

Também por vezes denominado *Schedule Length* A maior parte dos algoritmos de escalonamento utilizam esta métrica para mostrar e avaliar os seus resultados e soluções em comparação com outros algoritmos. Um menor *makespan* implica uma melhor performance. [ABAB13]

2.2.2 Turnaround Time

O *Turnaround Time* é a diferença de tempo entre a submissão de um *workflow* e a conclusão do processamento do mesmo. É diferente do *makespan*na medida em que inclui o tempo que o *workflow* fica em espera. É utilizado para medir a performance e satisfação na perspectiva do utilizador.[ABAB13] Esta métrica vai ser importante neste trabalho, visto ser direcionada a medir a satisfação do utilizador, o nosso parâmetro mais importante para medir QoS.

2.2.3 Turnaround Ratio

O *Turnaround Ratio* mede o tempo adicional que cada *workflow* gasta no sistema à espera de ser executado em relação ao *makespan* desse *workflow*.[ABAB13]

2.3 Clusters

Um *cluster* é um tipo de sistema paralelo ou distribuído que consiste num conjunto de computadores independentes interconectados (nós), que trabalham em conjunto como um único recurso computacional integrado. Um nó computacional pode ser um sistema com um ou mais processadores (PCs, *workstations*, servidores, etc.), com memória, processadores gráficos e um sistema operativo [Buy99]. *Cluster* geralmente refere-se a 2 ou mais computadores (nós) ligados entre si. Os nós podem existir numa única prateleira ou podem estar fisicamente separados e conectados por uma LAN. A figura 2.2 ilustra uma típica arquitetura de um cluster.

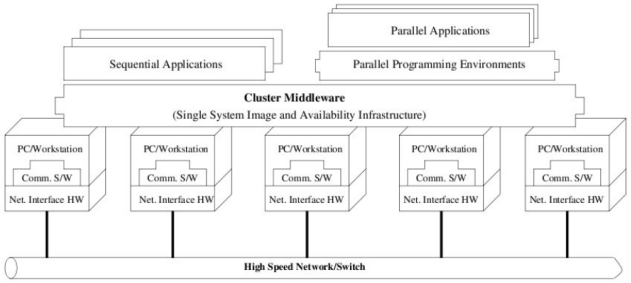


Figura 2.2: Arquitetura de um Cluster

Os algoritmos de escalonamento de workflows concorrentes são úteis quando o número de workflows excede o número de processadores nos vários nós de um cluster. Caso contrário, os workflows poderiam usar um conjunto de processadores exclusivamente sem concorrência. No caso da MEO, o cluster consiste em três servidores conectados na mesma rede. Numa rede comutada, a execução de tarefas e comunicação podem ser conseguidas em cada processador simultaneamente e sem contenção. Este facto ajuda a simplificar a computação dos custos de comunicação nos DAGs, considerando apenas os parâmetros de comunicação médios [ABAB13].

2.4 Técnicas de Escalonamento para workflows concorrentes

Uma operação de escalonamento consistem em definir um mapeamento e ordem para execução de tarefas[ABAB13]

Recentemente, vários algoritmos têm sido propostos para escalonamento de *workflows* concorrentes com o objetivo de melhorar o tempo de execução de várias aplicações em sistemas computacionais. A maior parte destes algoritmos foram desenvolvidos para escalonamento off-line, ou seja, em que todas as aplicações são conhecidas ao mesmo tempo. Esta abordagem, apesar de relevante, impõe limitações na gestão de um sistema dinâmico em que utilizadores podem submeter trabalhos a qualquer altura. Para este propósito, existem alguns algoritmos que foram concebidos para lidar com escalonamento em aplicações dinâmicas.[ABAB13].

Para um dedo cenário que inclua um conjunto de *workflows* submetidos em diferentes alturas por vários utilizadores, o objetivo é encontrar um mapeamento de tempo que garanta cumprir certas restrições de QoS, como por exemplo, o *Turnaround Time* do pré-processamento de um vídeo não pode demorar mais do que 2 vezes o tempo de processamento sem concorrência (*deadline*). O tempo de conclusão (ou *Turnaround Time*), como referido anteriormente, inclui tanto o tempo de espera como o tempo de execução de um *workflow*, estendendo a definição de *makespan*. [KA99]

2.4.1 Escalonamento Off-line

O escalonamento off-line caracteriza-se por ter os *workflows* disponíveis antes do início da execução. Quando um escalonamento é produzido e aplicado, mais nenhum *workflow* é considerado. Esta abordagem, apesar de limitada, aplica-se em muitos casos reais, como por exemplo,

quando um utilizador tem um conjunto de nós para correr um conjunto de *workflows*. Esta metodologia costuma ser aplicada pelas ferramentas de gestão de recursos mais comuns quando um utilizador reserva um conjunto de nós para executar os seus trabalhos exclusivamente.

Vários algoritmos foram propostos para escalonamento off-line nos quais *workflows* competem por recusos. O objectivo é garantir uma distribuição justa desses recursos e minimizar o *Turnaround Time* de cada *workflow*.

Zhao e Sakellariou [ZS06] apresentaram duas abordagens baseadas numa estratégia justa de escalonamento de *workflows* concorrentes. A justiça é definida na base do atraso (rácio entre o tempo de execução esperado para um DAG quando escalonado com outro(s) *workflows* e sozinho) que cada *workflow* experienciaria. Propuseram dois algoritmos, um com uma política de justiça baseada no tempo de término e um com uma política de justiça baseada no tempo atual. Inicialmente, ambos os algoritmos mapeiam cada DAG para todos os processadores utilizando uma técnica de escalonamento estático (como HEFT [THW02] ou Hybrid.BMCT [SZ04]) como algoritmo de escalonamento de articulação. Posteriormente, guardam o mapeamento atribuído e o seu *makespan* como o valor de atraso do DAG. Depois, todos os *workflows* são ordenados em ordem decrescente de atraso. Finalmente, enquanto houverem *workflows* por processar na lista, o algoritmo seleciona o DAG com o maior atraso e a primeira tarefa disponível que ainda não foi agendada neste DAG. A ideia do algoritmo é avaliar o valor do atraso de cada DAG depois de escalonar uma tarefa e decidir qual deverá ser o próximo DAG a ser selecionado para escalonar a próxima tarefa. A diferença entre os dois algoritmos propostos é que o com a política de equidade baseado no tempo de término apenas calcula o atraso do DAG selecionado, enquanto que o com a política de equidade baseada no tempo atual o valor do atraso é recalculado para cada DAG.

N'takpé e Suter [NS09] propuseram várias estratégias baseadas na partilha proporcional de recursos. Esta partilha proporcional foi definida com base no comprimento do caminho crítico, largura ou esforço associado a cada DAG. Foi também proposta uma partilha proporcional ponderada que representa um melhor equilíbrio entre partilha justa de recursos e a redução do *makespan* dos *workflows*. As estratégias foram propostas e aplicadas a aplicações paralelas mistas, onde cada tarefa pode ser executada em mais que um processador. A partilha proporcional baseada no esforço necessário para executar um *workflow* obteve os resultados menos demorados em média, mas por outro lado foi também menos justa no que toca à utilização de recursos, ou seja, a variação entre os atrasos experienciados foi a mais alta.

Bittencourt e Madeira [BM10] propuseram uma heurística de *clustering* de caminhos que combina a técnica de escalonamento de clustering para gerar grupos (*clusters*) de tarefas e uma tecnica de lista de escalonamento para selecionar tarefas e processadores. Com base nesta metodologia, Bittencourt e Madeira propõem e comparam quatro algoritmos, que são:

- escalonamento sequencial, onde *workflows* são agendados um a seguir ao outro
- algoritmo de procura de abertas, que é semelhante ao anterior mas procura abertas entre tarefas já agendadas
- algoritmo de intercalação, onde partes de cada workflow são agendadas por turnos

- *workflows* de grupo, onde os *workflows* são agregados para formar um único workflow e depois escalonados como tal.

A avaliação foi feita em termos do comprimento do agendamento e justiça e concluiu-se que intercalar os workflows leva a um menor *makespan* em média e maior equidade quando múltiplos workflows partilham o mesmo conjunto de recursos. Os resultados, apesar de relevantes, consideram o *makespan* médio, que não distingue o impacto do atraso em cada *workflow* quando comparado com uma execução exclusiva.

Casanova et al. [CDS10] avaliaram extensivamente algoritmos de escalonamento off-line de tarefas concorrentes paralelas num único *cluster* homogéneo. Os DAGs ou workflows submetidos por utilizadores diferentes partilham um conjunto de recursos e estão prontos a iniciar a sua execução ao mesmo tempo. O objetivo é optimizar a percepção do utilizador de desempenho e justiça. Os autores propuseram 3 métricas para quantificar a qualidade de um escalonamento relacionadas com desempenho e equidade entre os diferentes DAGs de tarefas paralelos.

Carbajal et al. [HCTY+12] apresentaram 2 algoritmos de escalonamento de *workflows*:

- MWGS4 - Multiple Workflow Grid Scheduling 4 stages
- MWGS2 - Multiple Workflow Grid Scheduling 2 stages

Estes algoritmos consistem em 4 e 2 fases: *labeling*, alocação adaptativa, priorização e escalonamento de máquinas paralelas. Ambos os algoritmos estão classificados como estratégias off-line e mapeiam um conjunto de trabalhos disponíveis e prontos a ser executados pertencentes a um lote de trabalhos. Todos os trabalho que chegarem num intervalo de tempo serão processados num lote e começarão a ser executados quando o processamento de lote de trabalho anterior estiver concluído. Foi mostrado que as estratégias propostas superam outras estratégias em termos de tempo de espera médio do caminho crítico e atraso do caminho crítico.

2.4.2 On-line Scheduling

O escalonamento on-line é caracterizado por ter um comportamento dinâmico onde *workflows* podem ser submetidos por utilizadores a qualquer altura. Quando se faz um escalonamento de vários *workflows* independentes que representam trabalhos de utilizadores e são submetidos em instantes diferentes no tempo, o *Turnaround Time* toma em conta quer o tempo de espera quer o tempo de execução de um dado *workflow*, estendendo a definição de *makespan* para escalonamento de um único *workflow* [KA99]. A métrica para avaliar um escalonador dinâmico de *workflows* independentes tem que representar o tempo individual de conclusão em vez de uma medida global para o conjunto de *workflows*, de modo a medir a QoS experienciada pelos utilizadores, estando esta relacionada com o tempo de conclusão de cada submissão de um utilizador.

Alguns algoritmos foram propostos para escalonamento de *workflows* on-line. Três outros foram propostos especificamente para escalonar workflows concorrentes com o objetivo de melhorar o QoS individual de cada *workflow*. Estes algoritmos, On- line Workflow Management (OWM),

Rank Hybrid (Rank Hybd) e Fairness Dynamic Workflow Scheduling (FDWS), são detalhados e comparados na próxima secção.

Liu et al. [LCJY09] propuseram o algoritmo MMA(Min-Min Average) para escalonar eficientemente *workflows* com muitas transações que envolvem *overheads* de comunicação consideráveis em grelhas computacionais. O algoritmo MMA é baseado no algoritmo Min-Min, mas usa uma estratégia diferente, que lhe permite adaptar-se às variações de velocidade da rede que liga a grelha automaticamente. *Workflows* com muitas transações são muitas instâncias de um *workflow* e o objetivo do MMA é otimizar a taxa de transferência, ao invés da performance individual de cada *workflow*.

Xu et al. [XCWB09] propuseram um algoritmo para escalonamento de múltiplos *workflows* com múltiplas restrições ed QoS para a *Cloud*. O algoritmo *Multiple QoS Constrained Scheduling Strategy of Multiple Workflows* (MQMW) minimiza o *makespan* e o custo dos recursos e aumenta a taxa de sucesso de escalonamento. O algoritmo considera dois objetivos, tempo e custo, que podem ser adaptados às necessidades dos utilizadores. Este algoritmo foi comparado ao *Rank Hybd* e quando tempo era o principal requisito de QoS, o *Rank Hybd* obteve melhor desempenho. Visto que o tempo é o nosso principal requisito de QoS, este algoritmo não vai ser testado.

Barbosa e Belmiro [BM11] propuseram um algoritmo dinâmico para minimizar o *makespan* de um lote de *workflows* paralelos com tempos de chegada diferentes. O algoritmo foi proposto para escalonamento on-line mas com o objetivo de minimizar uma métrica global. Este modelo é aplicado em cenários reais como videovigilância, registo de imagens e outros, nos quais os *workflows* estão relacionados e como tal apenas o resultado colectivo é significativo. Esta abordagem é diferente da que diz respeito a esta dissertação.

2.5 Algoritmos de Escalonamento

2.5.1 Algoritmo Rank Hybrid

Yu e Shi in [YS08] propuseram uma estratégia *planner-guided* strategy, de nome Rank Hybd, para lidar com o escalonamento dinâmico de *workflows* submetidos por utilizadores distinto em intervalos de tempo diferentes.

$$rank_u(n_i) = \overline{w_i} + \max_{n_j \in succ(n_i)} \{\overline{c_{i,j}} + rank_u(n_j)\}$$

Figura 2.3: Fórmula de $rank_u$

O algoritmo *Rank Hybd* atribui uma classificação a todas as tarefas utilizando a medida de prioridade $rank_u$ [THW02], que representa o tamanho do caminho mais longo de uma tarefa n_i até ao nó de saída de um DAG, incluindo o custo computacional de n_i e pode ser visto na Figura 2.3. Aqui, succ(n_i) representa o conjunto de sucessores imediatos da tarefa n_i, $c_{i,j}$ é o custo médio de

comunicação da aresta(i,j) e w_i é o custo médio de computação da tarefa n_i. Para a tarefa de saída, $rank_u(n_{exit}) = 0$.

```
1: if a new workflow has arrived then
2:     calculate rank_u for all tasks of the new workflow
3: end if
4: Ready_Pool ← Read all ready tasks from all DAGs
5: Resource_free ← get all idle resources
6: Assign rank_u as task priority to each task in the Ready_Pool
7: multiple ← number of different DAGs which have ready tasks in Ready_Pool
8: if multiple == 1 then
9:     sort all tasks in Ready_Pool in descending order of priority
10: else
11:     sort all tasks in Ready_Pool in ascending order of priority
12: end if
13: return Ready_Pool
```

Figura 2.4: Algoritmo 1 do Rank Hybd

```
1: while there are not scheduled workflows do
2:     Ready_Pool ← CheckWorkflows()
3:     while Ready_Pool AND Resource_free are not empty do
4:         task_select ← the first task in Ready_Pool list
5:         resource_select ← the processor with lowest finish time among of all
           resources in Resource_free list
6:         Assign task_select to resource_select
7:         remove resource_select from Resource_free list
8:         remove task_select from Ready_Pool list
9:     end while
10: end while
```

Figura 2.5: Algoritmo 2 do Rank Hybd

A cada passo, o algoritmo lê todas as tarefas disponíveis em todos os DAGs e seleciona a próxima tarefa a mapear de acordo com a sua classificação. Se as tarefas disponíveis pertencem a DAGs diferentes, o algoritmo seleciona a tarefa com a classificação mais baixa. Caso contrário, a tarefa com classificação mais alta é selecionada. A heurística do *Rank Hybd* é formalizada na figura 2.5 [ABAB13].

Recorrendo a esta estratégia, o algoritmo *Rank Hybd* permite que o DAG com classificação mais baixa (e *makespan* mais baixo) seja mapeado primeiro para reduzir o tempo de espera do DAG no sistema. No entanto, esta estratégia não é particularmente justa para todos os *workflows*. Isto porque dá sempre preferência a workflows mais pequenos e adia os *workflows* maiores. Se um *workflow* grande estiver a ser executado e forem submetidos vários *workflows* mais pequenos, o escalonador adiará a execução do *workflow* maior para dar prioridade aos mais pequenos.

2.5.2 Algoritmo Online Workflow Management

Hsu et al. [HHW11] propuseram o algoritmo OnlineWork- flow Management (OWM) para escalonamento on-line de múltiplos workflows. O algoritmo começa por selecionar uma tarefa disponível de cada DAG, a que tiver maior classificação ($rank_u$, Figura 2.3) e colocá-las na lista de tarefas prontas. Seguidamente, enquanto houverem DAGs por concluir, o algoritmo seleciona a tarefa com maior prioridade da lista de tarefas prontas. A partir daí, calcula o tempo mínimo de conclusão (EFT, *Earliest Finish Time*) para a tarefa selecionada em cada processador disponível e seleciona o que tiver menor EFT. Se o processador estiver disponíven nessa altura, o algoritmo

mapeia a tarefa selecionada ao processador selecionado. Caso contrário, mantém a tarefa na lista de tarefas prontas para ser mapeada mais tarde.

```
1:  while there are not scheduled workflows do
2:      Ready_Pool ← CheckWorkflows()
3:      while Ready_Pool AND Resource_free are not empty do
4:          task_select ← the first task in Ready_Pool list
5:          resource_select ← the processor with lowest finish time among of all
            available resources
6:          if FreeNumberOfCluster == 1 AND EFT on busy cluster is lower
            than Finish Time on resource_select then
7:              keep task_select for next schedule call
8:          else
9:              Assign task_select to resource_select
10:             remove resource_select from Resource_free list
11:             remove task_select from Ready_Pool list
12:         end if
13:     end while
14: end while
```

Figura 2.6: Heurística do OWM

A heurística do OW está formalizada na Figura 2.6.

2.5.3 Algoritmo Fairness Dynamic Workflow Scheduling

Arabnejad et al. [AB12] propuseram o algoritmo *Fairness Dynamic Workflow Scheduling* (FDWS). O algoritmo FDWS implementa novas estratégia tanto no aspecto de seleção de tarefas da lista de tarefas disponíveis como na atribuiçao de processadores de modo a reduzir o *Turnaround Time* individual dos *workflows*. Este algoritmo tem 3 componentes principais.

1. *Workflow Pool*

2. Selecção de Tarefas

3. Alocação de Processadores

A *Workflow Pool* contém os *workflows* submetido que chegam à medida que os utilizadores submetem os seus trabalhos. A cada ronda de escalonamento, este componente encontra todas as tarefas disponíveis de cada *workflow*. No algoritmo FDWS, tal como no algoritmo OWM, apenas uma tarefa disponível com a prioridade mais alta de cada DAG é adicionada à lista de tarefas prontas. Para atribuir a prioridade às tarefas, é utilizada a classificação $rank_u$ (Figura 2.3).

O componente de Seleção de Tarefas atribui uma classificação diferente para selecionar a tarefa a ser executada da lista de tarefas prontas. Para ser inserido na lista de tarefas prontas, a classificação $rank_u$ é calculada para cada *workflow* individualmente. Para selecionar uma tarefa da lista, é calculado o $rank_r$ para cada tarefa t_i pertencente a um DAG DAG_j, e a tarefa com maior classificação é selecionada.

```
1: while there are not scheduled workflows do
2:     Ready_Pool ← CheckWorkflows()
3:     while Ready_Pool AND Resource_free are not empty do
4:         task_select ← the first task in Ready_Pool list
5:         resource_select ← the processor with lowest finish time among of all
           available resources
6:         if FreeNumberOfCluster == 1 AND EFT on busy cluster is lower
           than Finish Time on resource_select then
7:             keep task_select for next schedule call
8:         else
9:             Assign task_select to resource_select
10:            remove resource_select from Resource_free list
11:            remove task_select from Ready_Pool list
12:        end if
13:    end while
14: end while
```

Figura 2.7: Heurística do OWM

A fórmula do $rank_r$ pode ser vista na Figura **??**. Esta métrica considera a percentagem de tarefas restantes (PRT, Percentage of Remaining Tasks) de um DAG e o seu comprimento do caminho crítico (CPL, Critical Path Length). O PRT dá mais prioridade a DAGs que estejam quase completos e tenham apenas um número de tarefas reduzido para executar. O CPL implemente uma estratégia diferente do que o tempo de processamento restante mais curto (SRPT, Smallest Remaining Processing Time) [MAS+99] que, a cada passo, seleciona e mapeia a aplicação com o menor tempo de processamento restante. O tempo restante de processamento é o tempo necessário para executar todas as tarefas restantes de um *workflow*. No entanto, o tempo necessário para completar todas as tarefas de um DAG não tem em conta a largura de um DAG. Um DAG mais largo tem um CPL menor que outro DAG com o mesmo número de tarefas, tendo também um tempo esperado de conclusão mais baixo. Assim sendo, neste caso, o FDWS daria maior importância a DAGs com menores CPLs.

O componente de Alocação de Processadores apenas considera os processadores livres e o processador com o menor tempo de conclusão para a tarefa selecionada.

```
1: while Workflow Pool is NOT Empty do
2:     if new workflow has arrived then
3:         calculate rank_u for all tasks of the new Workflow
4:         Insert the Workflow into Workflow Pool
5:     end if
6:     Ready_Pool ← one ready task from each DAG (highest rank_u)
7:     compute rank_r(t_{i,j}) for each task t_i belonging to DAG_j in Ready_Pool
8:     while Ready_Pool ≠ φ AND CPUs_free ≠ 0 do
9:         T_sel ← the task with highest rank_r from Ready_Pool
10:        EFT(T_sel, P_j) ← Earliest Finish Time of T_sel on free Processors P_j
11:        P_sel ← the processor with lowest EFT for task T_sel
12:        Assign Task T_sel to processor P_sel
13:        remove Task T_sel from Ready_Pool
14:    end while
15: end while
```

Figura 2.8: Algoritmo FDWS

Na Figura 2.8 encontra-se a formalização do algoritmo FDWS.

2.5.4 Algoritmo Multi-Workflow Deadline-Budget Scheduling

O algoritmo Multi-Workflow Deadline-Budget Scheduling (MW-DBS), tem como objectivo encontrar uma solução de escalonamento de *workflows* concorrentes tendo em conta restrições e

orçamentos definidos pelos utilizadores. Este algoritmo é uma abordagem baseada em escalonamento de listas e, como tal, é composto por duas fases principais:

1. Uma tarefa disponível de cada *workflow* é selecionada e é-lhe atribuída uma prioridade baseada no seu *deadline* (data e hora limite) individual.

2. É determinado um recurso apropriado para executar a tarefa atual que satisfaz os parâmetros de QoS do *workflow* a que a tarefa pertence.

O algoritmo considera simultaneamente o orçamento do utilizador e as restrições de *deadline* para executar o seu escalonamento [AB].

O MW-DBS foi concebido como uma estratégia heurística que em cada fase de seleção de processadores, obtém um mapeamento que cumpra sempre a restrição de custo (orçamento) e que possívelmente cumpra a restrição de *deadline* do *workflow* a que a tarefa pertence. Se a restrição de tempo for cumprida, é obtido um mapeamento bem-sucedido. Caso contrário é considerado um falhanço e o *workflow* deve ser terminado. O algoritmo é avaliado baseado na sua taxa de sucesso [AB].

Segue-se uma descrição mais detalhada das duas fases do algoritmo.

2.5.4.1 Fase de seleção de tarefas

Esta fase consiste na seleção da tarefa apropriada entre todas as tarefas disponíveis em cada *workflow*. Geralmente, dois métodos são utilizados para preencher a lista de tarefas prontas.

1. Obter uma única tarefa disponível de cada workflow [AB12], [AB14], [ABS14], [HHW11].

2. Obter todas as tarefas disponíveis pertencentes a cada *workflow* por concluir [YS08].

Adicionar todas as tarefas disponíveis de cada workflow tem como resultado uma estratégia injusta, visto que o número elevado de tarefas disponíveis pode levar a que certos *workflows* não participem na presente ronda de escalonamento. No algoritmo MW-DBS apenas uma tarefa, com o maior $rank_u$, é escolhida e adicionada à lista de tarefas prontas. Assim sendo, é necessária uma segunda atribuição de prioridades baseadas nos parâmetros de QoS para atribuír uma classificação secundária a cada tarefa na list de tarefas prontas. Como tanto o factor tempo como o de custo são restrições, a classificação secundária deve considerar ambas as medidas. Para além disso, resultados em [AB12] e [ABS14] mostram que ter em conta o histórico do workflow das tarefas mapeadas leva a melhor performance. Para este algoritmo foi utilizada uma nova classificação ($rank_D$) para atribuir uma prioridade secundária a cada tarefa t_i pertencente a um workflow j na lista de tarefas prontas.

$$rank_D(t_i, j) = \frac{1}{DEADLINE_j} \times STR_j$$

Figura 2.9: Fórmula de $rank_D$

A figura 2.9 demonstra a fórmula do algoritmo $rank_D$, onde $DEADLINE_j$ é o *deadline* de um *workflow* j, o rácio de tarefas mapeadas (STR, Sheduled Tasks Ratio) STR_j é o rácio entre número de tarefas mapeadas e o total de tarefas de um *workflow* j.

O valor do $rank_D$ é o produto de dois factores:

1. O inverso do factor de limitação de tempo *deadline* para o workflow j, dando mais prioridade a *workflows* submetidos e ainda não concluídos com menor valor de deadline.

2. A fracção do *workflow* j que está mapeado no sistema.

A razão do primeiro factor é que o escalonador vai considerar primeiro *workflows* que tenham menores limitações de tempo antes de falharem. A razão para o rácio de tarefas mapeadas é dar maior prioridade a *workflows* que foram submetidos mais cedo, de forma a que um *workflow* maior com várias tarefas já executadas possa ter prioridade sobre um *workflow* pequeno mais recente. No caso de dois *workflows* terem a mesma *deadline*, a tarefa que pertence ao *workflow* com maior rácio de tarefas executadas terá prioridade. Finalmente, a tarefa com o maior $rank_D$ na lista de tarefas prontas é selecionada para ser mapeada na fase seguinte.

2.5.4.2 Fase de seleção de Processador

Esta fase consiste na procura de um recurso apropriado para executar a tarefa selecionada. Em ambas as fases, o algoritmo precisa de considerar os parâmetros de QoS de modo a ter soluções de maior qualidade e maior desempenho do sistema. A fase de seleção de processador tem como objetivo selecionar um recurso acessível (em termos de custo) para a tarefa atual. Aqui foi proposta uma nova estratégia de seleção de processador baseada em parâmetros de QoS. Como cada workflow tem restrições de tempo e custo, ambos os parâmetros devem ser considerados na fase de seleção de processador. De modo a controlar o custo e tempo consumidos, um valor delimitado para cada fator é necessário. Abaixo são descritos os primeiros valores delimitados para custo e tempo e de seguida a estratégia de seleção do processador.

Limite de custo: existe uma limitação no consumo do orçamento por parte da tarefa atual t_curr baseada no orçamento utilizado por tarefas já executadas. No que toca a esta dissertação, o custo assumido vai ser sempre 1, visto que não é importante para os parâmetros de QoS estabelecidos.

Limite de tempo: Uma *sub-deadline* baseada no *deadline* do *workflow* é atribuída a cada tarefa do workflow. É aplicada a estratégia de distribuição para planeamento de *sub-deadlines* comum e direta. O valor da *sub-deadline* (DL) para cada tarefa t_i é calculado recursivamente percorrendo o grafo de tarefas para cima, começando pera tarefa de saída. Neste caso, a *sub-deadline* é definida considerando o tempo mínimo de execução da tarefa atual.

$$DL(t_{curr}) = \min_{t_{child} \in succ(t_{curr})} \Big[DL(t_{child}) - \overline{C}_{(t_{curr} \to t_{child})} - ET_{max}(t_{child}) \Big]$$

Figura 2.10: Fórmula de cálculo da *sub-deadline*

$$Q(t_{curr}, p_k) = Time_Q(t_{curr}, p_k) + Cost_Q(t_{curr}, p_k)$$

Figura 2.11: Fórmula de cálculo da medida de qualidade Q

$$RB_j = RB_j - AC(t_{curr})$$

Figura 2.12: Fórmula de cálculo do RB Q

$$RCA_j = RCA_j - Cost_{min}(t_{curr})$$

Figura 2.13: Fórmula de cálculo do RCA Q

Na figura 2.10 podemos ver a fórmula para o cálculo da *sub-deadline*, onde ET_max simboliza o tempo máximo de execução da tarefa t_curr entre os processadores disponíveis. Para a tarefa de saída, a *sub-deadline* é igual à *deadline* determinada pelo utilizador.

```
1:  procedure MW-DBS
2:      for t_{i,j} ∈ Ready Tasks pool do
3:          Assign a priority rank_D(t_{i,j})
4:      end for
5:      P_free ← free processors p_k ∈ P
6:      while (Ready Tasks ≠ φ & P_free ≠ φ) do
7:          t_curr ← task with highest priority
8:          for p_k ∈ P_admissible do
9:              Calculate Quality Measure Q(t_curr, p_k)
10:         end for
11:         P_sel ← Processor p_k with highest Q
12:         Assign current task t_curr to P_sel
13:         Update RB_j and RCA_j
14:         P_free ← P_free − P_sel
15:         Remove Task t_curr from Ready Tasks pool
16:     end while
17: end procedure
```

Figura 2.14: Algoritmo MW-DBS

O algoritmo MW-DBS pode ser visto na Figura 2.14

O algoritmo funciona da seguinte forma:

- todas as tarefas prontas na lista de tarefas prontas são classificadas pelo valor de prioridade $rank_D$.
- Para preencher a lista de tarefas prontas, o sistema recolhe uma única tarefa disponível com o maior valor de classificação primário $rank_u$ de cada *workflow* submetido por executar.
- Enquanto houver pelo menos uma tarefa pronta e por mapear na lista de tarefas prontas e houverem processadores disponíveis, a tarefa pronta a executar atual t_curr é selecionada e a sua medida de qualidade Q (Figura 2.11) para todos os processadores admissíveis.
- A tarefa atual t_curr é mapeada ao processador P_sel que tem a maior medida de qualidade Q.
- Finalmente, o orçamento não consumido restante (RB, Remaining Budget) (Figura 2.12) e a atribuição restante mais barata (RCA, Remain Cheapest Assignment) (Figura 2.13) são atualizados para o workflow j ao qual pertence a tarefa t_curr.

2.5.5 Comparativo

Nesta secção vão ser apresentadas algumas comparações entre os algoritmos acima mencionados. Entre as principais diferenças entre os algoritmos, as principais são as seguintes:

1. Enquanto que os algoritmos *Rank Hybd* e *OWM* dão mais importância à melhoria do tempo médio de finalização de todos os *workflows*, o algoritmo FDWS privilegia na QoS experienciada por cada utilizador, minimizando os tempos de espera e execução de cada *workflow* individual.
2. O algoritmo MQMW é o único que considera o custo (monetário) da execução de um *workflow*, o que é aplicável em situações em que o utilizador paga por ciclos de CPU, como por exemplo, alugando uma máquina num serviço *Cloud* como o Microsoft Azure.
3. Nos resultados apresentados em [HHW11], o algoritmo OWM obtém melhor performance que o Rank Hybd e o FDWS em *workflows* on-line.
4. Tal como o algoritmo Rank Hybd, o algoritmo OWM usa uma estratégia justa, mas ao invés de mapear DAGs mais pequenos primeiro, seleciona e mapeia tarefas dos DAGs maiores primeiro. No entanto, o algoritmo OWM tem uma melhor estratégia para preencher a lista de taregas prontas, selecionando uma tarefa de cada DAG, para dar uma hipótese a todos os DAGs de serem escolhidos na mesma altura para mapeamento.
5. Enquanto que o Rank Hybd adiciona todas as tarefas disponíveis à lista de tarefas prontas, o algoritmo OWM apenas adiciona uma tarefa disponível de cada DAG. Considerando todas as tarefas disponíveis de cada DAG leva a uma preferência imparcial para DAGs maiores e o consequente adiamento de DAGs mais pequenos, do qual resulta uma partilha do poder de processamento injusta.

6. Enquanto que os algoritmos Rank Hybd e OWM usam apenas o $rank_u$ para selecionar tarefas a adicionar à lista de tarefas prontas, tanto o FDWS como o MW-DBS utilizam métricas de classificação secundárias que tomam em conta o historial de um DAG na lista de tarefas prontas.

2.6 SIMGRID

Nesta dissertação vai ser utilizada uma simulação para avaliar os algoritmos da secção 2.5. Os simuladores permitem executar um número de testes estatisticamente significante numa configuração de máquinas definida pelo utilizador. Nesta dissertação, é utilizado o *toolkit* SIMGRID [CLQ08] como base para as nossas simulações. O SIMGRID disponibiliza as abstrações fundamentais necessárias à simulação de eventos discretos de aplicações paralelas em ambientes distribuídos. Foi especialmente desenvolvido para a avaliação de algoritmos de escalonamento. Basear-nos num *toolkit* de simulação bem estabelecido permite-nos tirar vantagem de modelos robustos de sistemas distribuídos. Em muitos artigos de pesquisa sobre escalonamento, os autores assumem um modelo de rede em que os processadores podem simultaneamente enviar ou receber dados de tantos processadores como determinarem, isto sem que sejam prejudicados em termos de performance. Este tipo de modelo, denominado *multi-port* não reflete uma rede numa infraestrutura. Inversamente, o modelo de rede disponibilizado pelo SIMGRID corresponde a um modelo *multi-port* teoreticamente delimitado. Neste modelo, um processador pode comunicar com vários outros simultaneamente, mas cada fluxo de comunicação é limitado pela largura de banda da rota percorrida, e comunicações que usam uma ligação comum têm que partilhar a largura de banda disponível. Isto corresponde ao comportamento de ligações TCP numa rede LAN. A validade deste modelo de rede foi demonstrado em [VL09].

Capítulo 3

Problema e Metodologia

3.1 Problema

3.2 Metodologia

Capítulo 4

Implementação

Capítulo 5

Resultados

Capítulo 6

Conclusões e Trabalho Futuro

6.1 Satisfação dos Objetivos

Referências

[AB] Hamid Arabnejad e Jorge G Barbosa. Multi-Workflow QoS-Constrained Scheduling for Utility Computing.

[AB12] Hamid Arabnejad e Jorge Barbosa. Fairness Resource Sharing for Dynamic Workflow Scheduling on Heterogeneous Systems. *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pages 633–639, 2012. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6280354, doi:10.1109/ISPA.2012.94.

[AB14] Hamid Arabnejad e Jorge G. Barbosa. A Budget Constrained Scheduling Algorithm for Workflow Applications. *Journal of Grid Computing*, 12(4):665–679, 2014. doi:10.1007/s10723-014-9294-7.

[ABAB13] Hamid Arabnejad, Jorge Barbosa, Hamid Arabnejad e Jorge Barbosa. Dynamic Scheduling of Workflows in Heterogeneous Systems. *Fair Resource Sharing for Dynamic Scheduling of Workflows on Heterogeneous Systems, 1st Edition*, 1:1–22, 2013.

[ABS14] Hamid Arabnejad, Jorge G. Barbosa e Fr??d??ric Suter. Fair Resource Sharing for Dynamic Scheduling of Workflows on Heterogeneous Systems. In *High-Performance Computing on Complex Environments*, pages 145–167. 2014. doi:10.1002/9781118711897.ch9.

[BM10] Luiz Fernando Bittencourt e Edmundo R M Madeira. Towards the scheduling of multiple workflows on computational Grids. *Journal of Grid Computing*, 8(3):419–441, 2010. doi:10.1007/s10723-009-9144-1.

[BM11] Jorge G. Barbosa e Belmiro Moreira. Dynamic scheduling of a batch of parallel task jobs on heterogeneous clusters. In *Parallel Computing*, volume 37, pages 428–438, 2011. doi:10.1016/j.parco.2010.12.004.

[Buy99] R Buyya. High Performance Cluster Computing: Architectures and Systems, Volume 1. *Prentice Hall PTR*, 82(Journal Article):327–350, 1999.

[CDS10] Henri Casanova, Frdric Desprez e Frdric Suter. On cluster resource allocation for multiple parallel task graphs. *Journal of Parallel and Distributed Computing*, 70(12):1193–1203, 2010. doi:10.1016/j.jpdc.2010.08.017.

[CLQ08] Henri Casanova, Arnaud Legrand e Martin Quinson. SimGrid: A Generic Framework for Large-Scale Distributed Experiments. *Tenth International Conference on Computer Modeling and Simulation (uksim 2008)*, pages 126–131, 2008. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4488918, doi:10.1109/UKSIM.2008.28.

[HCTY+12] Ad??n Hirales-Carbajal, Andrei Tchernykh, Ramin Yahyapour, Jos?? Luis Gonz??lez-Garc??a, Thomas R??blitz e Juan Manuel Ram??rez-Alcaraz. Multiple workflow scheduling strategies with user run time estimates on a Grid. *Journal of Grid Computing*, 10(2):325–346, 2012. doi:10.1007/s10723-012-9215-6.

[HHW11] Chih-Chiang Hsu, Kuo-Chan Huang e Feng-Jian Wang. Online scheduling of workflow applications in grid environments. *Future Generation Computer Systems*, 27(6):860–870, 2011. URL: http://dx.doi.org/10.1016/j.future.2010.10.015, doi:10.1016/j.future.2010.10.015.

[KA99] Yu-Kwong Kwok e Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, 1999. doi:10.1145/344588.344618.

[LCJY09] Ke Liu, Jinjun Chen, Hai Jin e Yun Yang. A Min-Min average algorithm for scheduling Transaction-intensive grid workflows. *Conferences in Research and Practice in Information Technology Series*, 99:41–48, 2009.

[MAS+99] M. Maheswaran, S. Ali, H.J. Siegal, D. Hensgen e R.F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. *Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99)*, 131:107–131, 1999. doi:10.1109/HCW.1999.765094.

[NS09] Tchimou N'Takpé e Frédéric Suter. Concurrent scheduling of parallel task graphs on multi-clusters using constrained resource allocations. In *IPDPS 2009 - Proceedings of the 2009 IEEE International Parallel and Distributed Processing Symposium*, 2009. doi:10.1109/IPDPS.2009.5161161.

[SZ04] R Sakellariou e Henan Zhao. A hybrid heuristic for DAG scheduling on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 111–123, 2004. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1303065, doi:10.1109/IPDPS.2004.1303065.

[THW02] Haluk Topcuoglu, Salim Hariri e M Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *Parallel and Distributed Systems, ...*, 13(3):260–274, 2002. URL: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=993206, doi:10.1109/71.993206.

[VL09] Pedro Velho e Arnaud Legrand. Accuracy study and improvement of network simulation in the {SimGrid} framework. *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, pages 1–10, 2009. URL: http://eudl.eu/doi/10.4108/ICST.SIMUTOOLS2009.5592, doi:10.4108/ICST.SIMUTOOLS2009.5592.

[XCWB09] Meng Xu, Lizhen Cui, Haiyang Wang e Yanbing Bi. A multiple QoS constrained scheduling strategy of multiple workflows for cloud computing. In *Proceedings - 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2009*, pages 629–634, 2009. doi:10.1109/ISPA.2009.95.

[YS08] Zhifeng Yu e Weisong Shi. A planner-guided scheduling strategy for multiple workflow applications. In *Proceedings of the International Conference on Parallel Processing Workshops*, pages 1–8, 2008. doi:10.1109/ICPP-W.2008.10.

[ZS06] Henan Zhao e Rizos Sakellariou. Scheduling multiple DAGs onto heterogeneous systems. In *20th International Parallel and Distributed Processing Symposium, IPDPS 2006*, volume 2006, 2006. doi:10.1109/IPDPS.2006.1639387.