

Entre Máquinas Virtuais e Funções *Serverless*: Uma Abordagem Heurística para o Escalonamento de *Workflows* Científicos em Nuvens de Computadores*

Marcello W. M. Ribeiro¹, Miguel Jehle¹, Ubiratam de Paula², Yuri Frota¹, Isabel Rosseti¹,
Daniel de Oliveira¹

¹Universidade Federal Fluminense, Niterói, RJ, Brasil

²Universidade Federal Rural do Rio de Janeiro, Rio de Janeiro, RJ, Brasil

{marcellomessina,miguelbjh}@id.uff.br, upaula@ufrrj.br,
{yuri,rosseti,danielcmo}@ic.uff.br

RESUMO

Os *workflows* científicos são abstrações que representam etapas de simulações computacionais por meio de DAGs e são executados por sistemas especializados. Devido à alta demanda computacional dos *workflows*, eles são frequentemente executados em ambientes de computação de alto desempenho, *e.g.*, nuvens, que oferecem elasticidade e recursos sob demanda. Entretanto, novos recursos das nuvens, como as funções *serverless*, ainda não são plenamente explorados pelos sistemas responsáveis pela execução dos *workflows*. Para preencher essa lacuna, este artigo propõe a heurística construtiva FLEXE, que realiza o escalonamento de ativações de *workflows* na nuvem, integrando o uso de máquinas virtuais (VMs) e funções *serverless*. A abordagem permite otimizar o tempo de execução ou o custo financeiro (ou ainda ponderar ambos, conforme a prioridade definida), considerando as características e limitações de cada recurso. Experimentos com um *workflow* real da área de bioinformática demonstraram o potencial da abordagem proposta.

PALAVRAS CHAVE. *Workflow*, Nuvem de Computadores, *Serverless*, Escalonamento.

ABSTRACT

Scientific workflows are abstractions that represent the stages of computational simulations using DAGs and are executed by specialized systems. Due to their high computational demands, these workflows are often executed in high-performance computing environments, such as clouds, which offer elasticity and on-demand resources. However, new cloud features such as serverless functions are still not fully explored by workflow execution systems. To address this gap, this paper proposes the constructive heuristic FLEXE, which schedules workflow activations in the cloud by integrating the use of virtual machines (VMs) and serverless functions. The approach allows for optimization of either execution time or financial cost (or a balance between the two), taking into account the characteristics and constraints of each resource. Experiments using a real-world bioinformatics workflow demonstrated the potential of the proposed approach.

KEYWORDS. *Workflow*, Cloud, *Serverless*, Scheduling

*Os autores gostariam de agradecer pelo apoio financeiro da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) – Código de Financiamento 001, do CNPq e da FAPERJ.

1. Introdução

Os *workflows* científicos (neste artigo chamados somente de *workflows*) são uma abstração empregada para modelar, de forma estruturada, o conjunto de atividades envolvidas na execução de uma simulação científica [de Oliveira et al., 2019]. Essas atividades, por sua vez, estão associadas à execução de programas e/ou *scripts*, cada qual com seus respectivos parâmetros de entrada e de saída. Os *workflows* são comumente representados por meio de Grafos Acíclicos Dirigidos (do inglês *Directed Acyclic Graphs* — DAGs), nos quais os vértices são as diferentes *atividades*, enquanto os arcos expressam as *dependências de dados* entre essas atividades [Deelman et al., 2015], indicando, por exemplo, a necessidade de um dado de saída de uma atividade anterior para que uma atividade subsequente possa ser executada. Além disso, uma *ativação de atividade* (ou apenas *ativação*) [Bouganin et al., 1996] se refere à execução de uma determinada atividade, *i.e.*, a materialização de uma atividade, sendo um elemento chave no escalonamento do *workflow*.

Os *workflows* são modelados e executados por meio de Sistemas de *Workflows* (SWfs), que oferecem funcionalidades que englobam desde a composição e definição dos *workflows*, passando por sua execução e monitoramento em tempo real, até a coleta e a gerência de dados de proveniência [Freire et al., 2008]. Comumente, um mesmo *workflow* pode ser executado inúmeras vezes com variações nos valores de parâmetros ou nos dados de entrada durante o processo experimental. Esse padrão de uso reiterado, aliado ao volume de dados processados nas execuções, impõe requisitos em termos de infraestrutura necessária. Para atender a tais demandas e garantir que os resultados sejam produzidos dentro de prazos viáveis, torna-se fundamental o uso de ambientes de processamento de alto desempenho (do inglês *High Performance Computing* - HPC) e de computação escalável em dados (do inglês *Data Intensive Scalable Computing* - DISC). Dentre esses ambientes, destacam-se os *clusters*, supercomputadores e as nuvens de computadores [de Oliveira et al., 2019].

De fato, muitos cientistas têm migrado a execução dos *workflows* para esses ambientes, em especial a nuvem [Srirama et al., 2013], impulsionados pela disponibilidade de recursos sob demanda que essas plataformas oferecem, como máquinas virtuais (VMs), áreas de armazenamento persistente, redes virtuais, *etc.* Um dos principais atrativos das nuvens sempre foi a característica de elasticidade, que permite ajustar dinamicamente a alocação de recursos computacionais de acordo com a carga de trabalho demandada. Diversos SWfs, *e.g.*, Pegasus [Deelman et al., 2015] e SciCumulus [de Oliveira et al., 2010], já oferecem suporte nativo para execução de *workflows* em nuvens, com estratégias de escalonamento de ativações variadas. Nesse contexto, o problema de escalonamento consiste em decidir, para cada ativação, em qual VM ela será executada. Esse problema é reconhecido como sendo NP-difícil [Davari, 1985], o que motiva o uso de algoritmos heurísticos.

Entretanto, novos recursos são continuamente adicionados ao catálogo de serviços dos provedores de nuvem, porém ainda não são suportados de forma nativa pelos SWfs [Farahani et al., 2024]. Um exemplo são as funções *serverless* [Garofalo et al., 2024]. Essas funções são blocos pequenos e independentes de código, executados sob demanda na nuvem, sem a necessidade de gerenciamento ou configuração de VMs, *i.e.*, sem que o usuário se preocupe em configurar a infraestrutura. Elas são disparadas por eventos, geralmente são *stateless* e possuem execução efêmera, *i.e.*, liberam os recursos computacionais logo após o encerramento da execução. Atualmente, praticamente todos os provedores de nuvem oferecem suporte a funções *serverless*, *e.g.*, AWS Lambda, Azure Functions e Google Cloud Functions.

O desafio, portanto, passa a ser a definição de uma estratégia eficiente para escalonar as ativações de um *workflow* no ambiente de nuvem, levando em consideração a possibilidade de utilizar tanto VMs quanto funções *serverless* como alternativas viáveis para a execução dessas ativações. Enquanto as funções *serverless* não são indicadas para tarefas com longa duração [Garofalo et al.,

2024], especialmente aquelas que excedam o limite de tempo definido pelo provedor de serviços (e.g. 15 minutos de execução na *AWS Lambda*), elas apresentam um bom custo-benefício quando aplicadas a ativações rápidas, pontuais e de baixa complexidade, como operações simples de filtragem e seleção de dados. No entanto, em cenários em que as ativações envolvem lógicas mais complexas, com múltiplas etapas de processamento, dependências ou necessidade de controle refinado sobre o ambiente de execução, a utilização de funções *serverless* torna-se inadequada. Nesses casos, recomenda-se a adoção de VMs para atender aos requisitos computacionais dessas ativações mais exigentes, conforme exemplo de *workflow* apresentado na Figura 1.

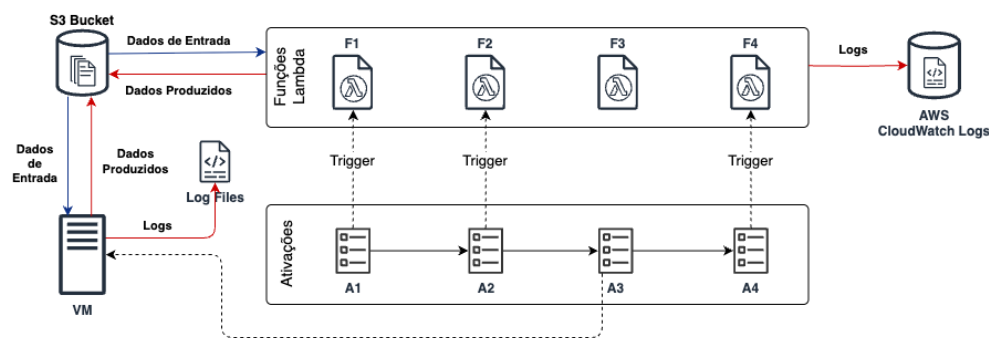


Figura 1: Um exemplo de execução de *workflow* com uso de VMs e funções *serverless*.

Este artigo propõe uma heurística construtiva, denominada FLEXE (*Flexible Execution for Workflows in VMs and Serverless*), voltada à resolução do problema de escalonamento de ativações de *workflows* em ambientes de nuvem, considerando, de forma integrada, tanto o uso de VMs quanto de funções *serverless* como recursos disponíveis para a execução das ativações. A heurística desenvolvida permite ao usuário configurar o critério de otimização desejado, seja a minimização do tempo total de execução (*makespan*), ou seja a redução do custo financeiro associado ao uso dos recursos na nuvem. A heurística considera múltiplos fatores relevantes para o desempenho da execução, tais como a capacidade de processamento de cada tipo de recurso (VMs e funções *serverless*) e as limitações de tempo de execução das funções *serverless*. A partir dessas informações, é gerado um plano de escalonamento que busca atender ao critério de otimização definido, respeitando as restrições operacionais e aproveitando as vantagens oferecidas por cada tipo de recurso. A avaliação empírica foi realizada utilizando um *workflow* real da área de filogenômica [Moraes et al., 2025]. Os resultados demonstraram ganhos em desempenho ou em custo, dependendo do objetivo configurado pelo usuário.

O artigo encontra-se organizado em seis seções além da Introdução. Na Seção 2, apresentamos os conceitos para o entendimento do problema. Na Seção 3, ilustramos o problema de escalonamento de *workflows*. Na Seção 4, descrevemos a heurística construtiva proposta, bem como o algoritmo *multistart* para resolver este problema. Resultados experimentais são mostrados na Seção 5. A Seção 6 apresenta os trabalhos relacionados. Por fim, a Seção 7 conclui o artigo e discute trabalhos futuros.

2. Uma Suave Introdução à Computação *Serverless*

O paradigma de computação *serverless* [Hassan et al., 2021] representa uma mudança em relação aos modelos clássicos de computação em nuvem, que, em geral, se baseiam no provisionamento e gerenciamento de VMs. Em contraste, a abordagem *serverless* permite que os desenvolvedores concentrem seus esforços unicamente na lógica de suas aplicações, delegando ao provedor de nuvem a complexidade da configuração, do provisionamento e da escalabilidade da infraestrutura. Embora

internamente os provedores possam utilizar tecnologias como contêineres (e.g., Docker) para o isolamento e a execução das aplicações, o usuário não precisa gerenciar esses artefatos diretamente. O paradigma *serverless* se manifesta, principalmente, através de duas abordagens distintas [Mampage et al., 2021]: (i) *Backend* como Serviço (BaaS) e (ii) *Função* como Serviço (FaaS).

No modelo BaaS, além da aplicação principal, os desenvolvedores podem integrar suas aplicações a serviços de terceiros, acessíveis através de APIs bem definidas. Esses serviços oferecem funcionalidades como autenticação de usuários, armazenamento de arquivos, bancos de dados, mensageria, etc, permitindo construir aplicações sem a necessidade de implementar e manter essa infraestrutura por conta própria. Por sua vez, o modelo FaaS depende de serviços locais (no *cluster*), de modo que os desenvolvedores são responsáveis por implementar a lógica de suas aplicações em unidades de código isoladas e *stateless*. Essas funções são executadas em contêineres e sob demanda, em resposta a eventos específicos. O provedor de nuvem gerencia completamente o ambiente de execução, incluindo a alocação de recursos e o escalonamento automático.

É importante destacar que os contêineres usados são *stateless*, o que implica desafios para aplicações que requerem persistência de dados. Os contêineres FaaS são efêmeros, i.e., projetados para execução em janelas curtas de tempo, e a maioria dos provedores de nuvem já oferece suporte a esse modelo, e.g., *AWS Lambda*, *Google Cloud Functions*, *Azure Functions*. O presente artigo adota a arquitetura FaaS. Diferentemente dos ambientes IaaS (*Infrastructure as a Service*), baseados em VMs, o modelo FaaS dispensa a criação, configuração e manutenção de VMs (e.g., aplicação de *patches* de segurança), abstraindo os servidores durante o processo de desenvolvimento e simplificando a operação. Nessas arquiteturas, o desenvolvedor pode iniciar diretamente pelo empacotamento do código da aplicação (i.e., função) em contêineres e sua implantação no ambiente *serverless*. Como consequência, o custo financeiro associado às operações de desenvolvimento pode ser reduzido com a adoção do modelo FaaS, especialmente em cenários com baixa frequência de requisições. A cobrança ocorre apenas pelo tempo de computação consumido durante a execução dessas funções, otimizando custos e a eficiência operacional. Outro benefício relevante do FaaS é o tempo de implantação. Ao contrário das VMs, que demandam tempo do usuário para configuração inicial da infraestrutura e um tempo de inicialização, funções em ambientes *serverless* podem ser disponibilizadas em questão de milissegundos.

3. Modelo de Aplicação e Arquitetura

A heurística FLEXE parte do pressuposto de que os *workflows* são representados por meio de um DAG $G = (D_s \cup D_d \cup N, A)$, em que o conjunto de vértices é constituído pelas ativações N e por um conjunto de arquivos de dados $D = D_s \cup D_d$. Nesse contexto, D_s corresponde aos dados de entrada considerados estáticos, enquanto D_d representa os dados dinâmicos, gerados ao longo da execução do *workflow*. Por sua vez, os arcos que integram o conjunto A estabelecem as relações de produção e consumo entre as ativações e os dados, conforme descrito em [Teylo et al., 2017]. Cada ativação é associada à execução de um programa, seja em uma VM, seja em uma função *serverless*. Além disso, tal ativação encapsula os parâmetros e os arquivos de dados que deverão ser consumidos durante sua execução. Assim, define-se $\Delta_{in}(i) \subseteq D$ como o conjunto de dados, sejam eles estáticos ou dinâmicos, necessários para que uma ativação $i \in N$ possa ser executada. Analogamente, $\Delta_{out}(i) \subseteq D_d$ representa o conjunto de dados dinâmicos produzidos como resultado da execução da ativação i .

Além da formalização do *workflow*, é necessário definir as principais características do ambiente de execução em nuvem considerado. Neste artigo, assume-se que tal ambiente é híbrido, i.e., as ativações podem ser executadas tanto em VMs quanto em funções *serverless*. Seja M o conjunto de VMs disponíveis para processamento e armazenamento de dados, e seja B o conjunto de *buckets* utilizados exclusivamente para fins de armazenamento persistente. Diferentemente das

VMs, que possuem capacidade de computação e também podem ser utilizadas para armazenar dados temporária ou permanentemente, os *buckets* servem apenas como repositórios de dados. Adicionalmente, denotamos por K o conjunto de configurações possíveis para funções *serverless*, onde cada configuração é definida por parâmetros como o número de vCPUs, a quantidade de RAM alocada e o *container* específico a ser executado. Assim, é possível representar o ambiente de nuvem híbrido por meio do conjunto $\overline{M} = (B \cup M \cup K)$, que agrega os recursos de armazenamento, computação tradicional e execução *serverless* disponíveis para a realização dos *workflows*.

Assim, dado o ambiente híbrido descrito, podemos expandir a definição do conjunto de ativações para $N = N^{vm} \cup N^{f(x)}$, onde N^{vm} representa o subconjunto de ativações que, por suas características, só podem ser executadas em VMs, enquanto $N^{f(x)}$ corresponde ao subconjunto de ativações que podem ser executadas tanto em VMs quanto por meio da invocação de funções *serverless*. Para cada ativação $i \in N$ e para cada VM $j \in M$, define-se t_{ij}^{vm} como o tempo estimado necessário para o processamento da ativação i na VM j . De forma análoga, para cada ativação $i \in N^{f(x)}$ e para cada configuração $k \in K$ de função *serverless*, define-se $t_{ik}^{f(x)}$ como o tempo estimado para a execução da ativação utilizando tal configuração. Esse tempo inclui o envio dos dados necessários, o tempo de processamento da função e o tempo de recebimento dos dados de saída. Quanto aos custos financeiros associados à utilização dos recursos computacionais, denota-se por c_j^{vm} o custo de contratação da VM $j \in M$ por unidade de tempo, e por $c_{ik}^{f(x)}$ o custo de executar a ativação $i \in N^{f(x)}$ utilizando a configuração $k \in K$ de função *serverless*. Com base nessas estimativas de tempo e custo, a heurística FLEXE deve ser capaz de tomar decisões quanto ao escalonamento eficiente das ativações $i \in N$, determinando não apenas se cada ativação deve ser alocada a uma VM ou a uma função *serverless*, mas também em que ordem essas ativações devem ser executadas.

4. A Heurística Construtiva FLEXE

A heurística construtiva FLEXE proposta neste artigo é apresentada no Algoritmo 1 e tem o objetivo de escalonar ativações do *workflow* em um ambiente híbrido composto por VMs e funções *serverless*. O algoritmo recebe como entrada: (i) o conjunto M de VMs, (ii) o conjunto de configurações de funções *serverless* K , (iii) a especificação do *workflow* G , (iv) os custo financeiro e tempo execução máximos C_M e T_M , (v) o parâmetro α que define o interesse do usuário em minimizar o custo financeiro ou o tempo de execução do *workflow*, (vi) uma semente de geração de números aleatórios e (vii) o parâmetro ϕ que define o fator de corte da lista de soluções candidatas. Em cada iteração, a partir de um conjunto de ativações viáveis AV (*i.e.*, ativações que possuem todas as suas dependências de dados satisfeitas e, portanto, podem ser executadas), uma nova ativação é selecionada de forma aleatória e adicionada à solução, *i.e.*, ao plano de escalonamento. Primeiramente, a solução inicial recebe o conjunto vazio (linha 1) e o conjunto de ativações viáveis AV é gerado a partir de G (linha 2).

A lista de candidatos LC é preenchida com todas as possíveis tuplas de ativações viáveis (*ativação, configuração de função*) e (*ativação, máquina virtual*) (linhas 5 a 16), onde a primeira posição da tupla representa uma ativação viável a ser inserida na solução, e a segunda indica, ou uma das possíveis configurações de função, ou uma das VMs disponíveis. Cada tupla é normalizada em relação ao interesse α do usuário em priorizar, ou a minimização do custo financeiro da solução, ou a minimização do tempo de execução do *workflow*, sendo os candidatos ordenados, de maneira crescente, de acordo com esses valores normalizados (linhas 17 e 18). Após o preenchimento de todos os candidatos possíveis, é gerada a lista restrita de candidatos LRC com os $\phi * |LC|$ melhores candidatos (linhas 19 e 20). Por fim, um candidato é selecionado aleatoriamente para ser inserido na solução S (linhas 21 e 22). Na linha 23, o conjunto de ativações viáveis AV é atualizado. O algoritmo finaliza com todas as ativações escalonadas, retornando a solução S gerada (linha 25).

Algoritmo 1 Construtivo (α , $semente$, ϕ , C_M , T_M , G , M , K)

```

1:  $S \leftarrow \emptyset$ 
2:  $AV \leftarrow \text{criarConjuntoAtivacoesViaveis}(G)$ 
3: Enquanto  $AV \neq \emptyset$  faça
4:    $LC \leftarrow \emptyset$ 
5:   Para cada  $a \in AV$  faça
6:     Para cada configuração de função  $cf \in K$  faça
7:        $tupla \leftarrow \text{criarTupla}(a, cf)$ 
8:        $LC \leftarrow LC \cup \{tupla\}$ 
9:     Fim Para
10:  Fim Para
11:  Para cada  $a \in AV$  faça
12:    Para cada máquina virtual  $mv \in M$  faça
13:       $tupla \leftarrow \text{criarTupla}(a, mv)$ 
14:       $LC \leftarrow LC \cup \{tupla\}$ 
15:    Fim Para
16:  Fim Para
17:   $LC \leftarrow \text{normalizar}(LC, \alpha, C_M, T_M, G, M, K)$ 
18:   $LC \leftarrow \text{ordenarCrescente}(LC)$ 
19:   $corte \leftarrow |LC| * \phi$ 
20:   $LRC \leftarrow \text{criarLRC}(LC, corte)$ 
21:   $candidato \leftarrow \text{selecionarAleatoriamente}(LRC, semente)$ 
22:   $S \leftarrow S \cup \{candidato\}$ 
23:   $AV \leftarrow \text{atualizarConjuntoAtivacoesViaveis}(AV, G)$ 
24: Fim Enquanto
25: Retorne  $S$ 

```

O Algoritmo 2 apresenta o método *multistart* que executa a heurística construtiva FLEXE apresentada no Algoritmo 1 por *max_iter* iterações, retornando a melhor solução S alcançada ao longo dessas iterações. Inicialmente, a melhor solução, o valor de custo normalizado da melhor solução e o número de iterações são inicializados (linhas 1 a 3). Das linhas 4 a 11, em cada uma das iterações, o Algoritmo 1 é invocado, gerando uma nova solução. Se o valor de custo desta nova solução for menor que o da melhor solução atual, a melhor solução é atualizada. Ao final das *max_iter* iterações, a melhor solução obtida é retornada. Os algoritmos associados à heurística FLEXE foram implementadas em C++, utilizando o compilador GNU GCC (versão C/C++ 13.3.0). O código-fonte da FLEXE se encontra disponibilizado no GitHub institucional [Anonimizado].

Algoritmo 2 MultiStart (α , $semente$, max_iter , ϕ , C_M , T_M , G , M , K)

```

1:  $S^* \leftarrow \emptyset$ 
2:  $f^* \leftarrow \infty$ 
3:  $iter \leftarrow 0$ 
4: Enquanto  $iter < max\_iter$  faça
5:    $S \leftarrow \text{Construtivo}(\alpha, semente, \phi, C_M, T_M, G, M, K)$ 
6:   Se  $f(S) < f^*$  então
7:      $S^* \leftarrow S$ 
8:      $f^* \leftarrow f$ 
9:   Fim Se
10:   $iter \leftarrow iter + 1$ 
11: Fim Enquanto
12: Retorne  $S^*$ 

```

5. Avaliação Experimental

Esta seção apresenta a avaliação experimental da FLEXE. Primeiramente, é descrito o estudo de caso, seguido pela caracterização do ambiente experimental, o processo de ajuste de parâmetros e, por fim, a discussão dos resultados obtidos.

5.1. Estudo de Caso e Geração de Instâncias

O *workflow* NMFSt.P [Moraes et al., 2025] foi escolhido como estudo de caso desse artigo e adaptado para execução em ambiente de nuvem, tanto por meio de funções *serverless*, quanto em VMs. O *workflow* tem como objetivo realizar a mineração de árvores filogenéticas, *i.e.*, estudar o comportamento evolutivo de múltiplas espécies a partir de um conjunto de sequências de DNA, RNA e aminoácidos de entrada. O NMFSt.P possui quatro atividades: (i) geração de árvores filogenéticas, (ii) criação de subárvores, (iii) comparação de similaridades entre subárvores e (iv) agregação dos dados.

Foram realizadas 33 execuções do *workflow*, utilizando como entrada *datasets* armazenados em um *bucket S3* da nuvem da AWS, com variação na quantidade de *datasets* entre 2 e 50, resultando em 33 instâncias contendo entre 7 e 151 ativações. Ao usar funções *serverless*, cada execução do *workflow* foi repetida 5 vezes, uma para cada configuração $cf \in K$ que variava a quantidade de RAM: 128MB, 256 MB, 512 MB, 1024 MB e 2048 MB. Ao executar as ativações em VMs, também foram realizadas 33 execuções do *workflow*, utilizando uma VM base para obtenção do tempo médio de CPU, permitindo a comparação com até cinco tipos diferentes de VMs. Os custos, tanto das configurações de função *serverless* quanto das VMs, foram calculados com base nos preços disponibilizados pela Amazon AWS para cada tipo de infraestrutura e configuração utilizada. A nomenclatura adotada para as instâncias geradas neste artigo possui a seguinte estrutura: <número de datasets>_T<número de tarefas>_C<número de configurações de função>_D<quantidade de dados consumidos>_VM<número de máquinas virtuais>.

5.2. Ajuste de parâmetros e Ambiente de Execução

A heurística construtiva apresentada no Algoritmo 1 foi parametrizada utilizando a ferramenta IRACE [López-Ibáñez et al., 2016]. Investigamos o parâmetro ϕ (percentual da *LC* utilizado para construir o conjunto *LRC*). O experimento de ajuste de ϕ com o IRACE foi executado por 1.000 iterações [Bouamama e Blum, 2017; Maschler et al., 2017], sobre um subconjunto de 5 instâncias (005_T16_C5_D49_VM5, 010_T31_C5_D97_VM5, 030_T91_C5_D260_VM5, 040_T121_C5_D284_VM5 e 050_T151_C5_D347_VM5), que não foram utilizadas nos experimentos subsequentes. Após a execução do IRACE, o melhor valor encontrado para ϕ foi 0,3486. Todos os experimentos foram realizados em um servidor do Laboratório [Nome Anonimizado] equipado com um processador Intel®Core(TM) i9-14900KF, 32 CPUs e 130GB de memória RAM, executando o sistema operacional Linux Ubuntu 22.04, com as funcionalidades de processamento paralelo habilitadas.

5.3. Resultados

O Algoritmo 2 foi executado dez vezes, utilizando dez sementes iniciais distintas. Em cada uma de suas execuções, a heurística construtiva proposta foi aplicada 200 vezes. A Tabela 1 apresenta os resultados obtidos: a primeira coluna exibe o nome da instância; a segunda, terceira e quarta colunas apresentam, dentre as 10 execuções, respectivamente, o melhor valor do custo financeiro, o tempo de execução do *workflow* e o tempo médio de execução da heurística, considerando a prioridade na minimização do custo financeiro. A quinta, sexta e sétima colunas seguem o mesmo padrão da segunda, terceira e quarta colunas, porém com prioridade na minimização do tempo de execução do *workflow*. Por fim, as três últimas colunas indicam as mesmas informações (*i.e.*, o melhor valor

do custo financeiro, o tempo de execução do *workflow* e o tempo médio de execução da heurística), considerando que não há prioridade entre custo financeiro e tempo de execução do *workflow*.

A análise dos resultados apresentados na Tabela 1 evidencia que a heurística construtiva FLEXE se comporta de forma alinhada às prioridades estabelecidas pelo usuário, *i.e.*, a definição do valor de α . Quando a prioridade é a minimização do custo financeiro, observa-se que, em todas as instâncias avaliadas, os menores valores de custo (coluna cf_{melhor}) estão concentrados na configuração onde essa métrica foi explicitamente priorizada, *i.e.*, na segunda coluna. Esses valores são, consistentemente, inferiores aos correspondentes das colunas cinco e oito, onde a prioridade é outra ou os critérios são balanceados.

Da mesma forma, ao se priorizar o tempo de execução do *workflow*, os menores tempos foram observados na configuração cujo α prioriza a minimização do tempo de execução, *i.e.*, a sexta coluna, superando os resultados das colunas três (prioridade: custo financeiro) e nove (critérios balanceados). Por exemplo, na instância 050_T151_C5_D411_VM5, o tempo de execução caiu de aproximadamente 2620,17 segundos (com prioridade de custo financeiro) para 1667,50 segundos quando o tempo de execução foi priorizado, uma redução superior a 36%. Em instâncias menores, como 002_T7_C5_D14_VM5, a diferença percentual também é significativa, demonstrando o impacto da priorização mesmo em cenários com menos ativações envolvidas.

Quando não há uma prioridade definida, *i.e.*, busca-se um equilíbrio entre custo financeiro e tempo de execução, os resultados situam-se, como esperado, entre os extremos. A oitava e a nona colunas, referentes à essa configuração, apresentam valores intermediários em relação às colunas de priorização. Por exemplo, na instância 035_T106_C5_D240_VM5, o custo financeiro (0,004125) situa-se entre o menor custo (0,002885) e o maior (0,006048); o mesmo se verifica para o tempo de execução do *workflow* (650,68s contra 887,42s e 602,75s, respectivamente).

Adicionalmente, vale destacar que a heurística demonstrou eficiência em termos de tempo médio de execução (coluna t_{CPU}). Especificamente, nas execuções com prioridade para o custo financeiro, a heurística alcançou o menor tempo médio em 22 das instâncias avaliadas, superando as configurações voltadas à minimização do tempo de execução do *workflow* e as configurações equilibradas. Isso demonstra que, além de eficaz na obtenção de soluções de baixo custo, a heurística também se mostra computacionalmente eficiente sob esse critério. Em síntese, os resultados comprovam a flexibilidade e a adaptabilidade da FLEXE, permitindo que o usuário configure sua execução conforme os objetivos desejados, seja otimizando custo, tempo ou ambos, com consistência nos resultados obtidos em diferentes cenários e tamanhos de instância.

6. Trabalhos Relacionados

O escalonamento eficiente de *workflows* é fundamental para otimizar o desempenho e reduzir os custos financeiros de execuções em nuvens. Essa seção discute os trabalhos relacionados que consideram a execução eficiente de *workflows* em ambientes de nuvem híbridos que consideram tanto VMs quanto funções *serverless*. Nesse contexto, Roy et al. [2021] propõem o *Mashup*, um sistema para distribuição das atividades do *workflow* de acordo com suas características, de forma a aproveitar melhor os recursos computacionais em ambientes híbridos que consideram tanto VMs quanto funções *serverless*. Os autores se baseiam em um algoritmo guloso. No entanto, o *Mashup* foca na minimização do tempo total de execução do *workflow*, não oferecendo ao usuário a opção de balancear o tempo de execução e o custo financeiro. Ramesh et al. [2023] introduzem um método de escalonamento baseado na Otimização por Enxame de Partículas (PSO) que pode ser aplicado em *workflows* em plataformas *serverless multicloud*. O objetivo do algoritmo é a minimização conjunta do tempo total de execução e do custo financeiro dos *workflows*. Entretanto, a solução apresentada se concentra na execução de ativações em funções *serverless*, não explorando a alternativa de alocar ativações do *workflow* para VMs.

Instância	Prioridade: custo financeiro			Prioridade: tempo de execução do workflow			Sem prioridade		
	cf_{melhor}	$t_{melhor}(s)$	$t_{CPU}(s)$	cf_{melhor}	$t_{melhor}(s)$	$t_{CPU}(s)$	cf_{melhor}	$t_{melhor}(s)$	$t_{CPU}(s)$
002.T7.C5.D14.VM5	0,000015	4,931297	0,037387	0,000024	3,061849	0,036670	0,000018	3,131820	0,037028
002.T7.C5.D16.VM5	0,000017	6,091565	0,038671	0,000027	3,481964	0,037944	0,000021	3,588574	0,037930
002.T7.C5.D18.VM5	0,000018	7,453888	0,040639	0,000027	4,247806	0,042304	0,000023	4,268236	0,041871
005.T16.C5.D35.VM5	0,000068	22,363776	0,351969	0,000100	14,411112	0,358543	0,000084	14,792376	0,349915
005.T16.C5.D38.VM5	0,000073	31,665210	0,367187	0,000107	16,362719	0,380237	0,000093	16,703247	0,373542
010.T31.C5.D70.VM5	0,000255	78,299273	2,427346	0,000364	53,603337	2,504076	0,000326	54,071005	2,496193
010.T31.C5.D71.VM5	0,000252	89,764056	2,454390	0,000368	54,903231	2,520265	0,000337	55,017915	2,504059
015.T46.C5.D105.VM5	0,000554	175,343336	8,550783	0,000792	118,598799	8,771997	0,000770	119,248532	8,717456
015.T46.C5.D108.VM5	0,000583	208,725484	8,694107	0,000950	124,197932	8,855739	0,000801	125,188815	8,871198
015.T46.C5.D140.VM5	0,000975	253,852265	12,707531	0,001557	189,830146	12,688139	0,001216	190,650503	12,818340
020.T61.C5.D138.VM5	0,000972	289,526649	21,509830	0,001683	201,968966	21,883446	0,001390	212,831487	21,716721
020.T61.C5.D145.VM5	0,001070	350,374728	22,818236	0,001789	221,970305	22,989537	0,001506	235,089973	22,986904
020.T61.C5.D180.VM5	0,001536	489,210499	31,909689	0,002560	315,086597	32,971669	0,002314	327,068194	32,500096
025.T76.C5.D171.VM5	0,001443	472,080120	43,844992	0,002572	307,912081	45,223303	0,002376	311,767035	44,507347
025.T76.C5.D176.VM5	0,001506	470,270554	45,793877	0,002715	326,258420	46,634068	0,002346	338,458461	46,412418
025.T76.C5.D220.VM5	0,002371	729,207626	66,802874	0,003827	472,924189	67,893132	0,003579	483,193268	67,451097
030.T91.C5.D209.VM5	0,002307	647,777106	85,329140	0,004177	456,700795	87,232955	0,003299	479,475331	86,202588
030.T91.C5.D212.VM5	0,002194	729,150848	86,362574	0,004249	471,898409	88,136551	0,003275	500,419729	87,198544
035.T106.C5.D240.VM5	0,002885	887,416019	140,636850	0,006048	602,753581	143,409605	0,004125	650,687704	142,629679
035.T106.C5.D253.VM5	0,003427	938,327067	152,075439	0,006544	665,531352	155,203265	0,004564	725,072091	154,365663
035.T106.C5.D300.VM5	0,004548	1266,109791	216,198989	0,008739	879,712038	213,731966	0,006338	946,998426	216,049253
040.T121.C5.D275.VM5	0,003861	1142,812658	226,000167	0,007918	787,309546	231,059010	0,005562	844,160206	229,778756
040.T121.C5.D340.VM5	0,005748	1661,997421	347,035916	0,011337	1131,152123	350,574550	0,008277	1207,849982	349,902127
045.T136.C5.D311.VM5	0,004815	1462,552537	345,062571	0,009769	1003,283863	352,497765	0,007316	1064,112998	350,857198
045.T136.C5.D314.VM5	0,005057	1551,722170	357,936070	0,010275	1027,098332	363,022212	0,007410	1090,960677	358,758942
045.T136.C5.D376.VM5	0,007233	2149,773081	510,523111	0,013847	1389,948039	519,049813	0,009991	1506,478827	511,750615
050.T151.C5.D355.VM5	0,006465	1876,472469	544,017066	0,013001	1299,030841	554,138071	0,009234	1383,280195	549,920278
050.T151.C5.D411.VM5	0,008702	2620,173908	738,694211	0,016599	1667,500898	740,068370	0,011839	1806,122290	731,596975

Tabela 1: Resultados obtidos pela heurística FLEXE.

Burckhardt et al. [2022] abordam o problema de execução de tarefas em ambiente *serverless*, por meio do Netherite, uma arquitetura distribuída e elástica para executar *workflows serverless* de forma eficiente e confiável, com manutenção do estado. Ele se concentra em melhorar a latência da comunicação e a vazão através de técnicas como particionamento de objetos e o uso do *framework* FASTER para gerenciar estado e permitir recuperação rápida de partições. De forma semelhante a Ramesh et al. [2023], o Netherite não prevê a possibilidade de execução em VMs dedicadas, apenas utiliza funções *serverless*. Majewski et al. [2021] apresentam dois novos algoritmos de escalonamento: SMOHEFT e SML. O SMOHEFT é uma adaptação do algoritmo MOHEFT, um método baseado em lista e multiobjetivo que visa minimizar o tempo e o custo de execução de um *workflow*. O SML, por sua vez, é uma extensão do SMOHEFT que utiliza conceitos do algoritmo LOSS para encontrar soluções que atendam ao custo financeiro, mas que extrapolem as restrições de tempo definidas. Assim como Ramesh et al. [2023] e Burckhardt et al. [2022] o trabalho se concentra em atender apenas o ambiente *serverless*.

Kumari et al. [2023] propõem um modelo analítico para estimar o tempo e custo de execução de *workflows* em ambientes *serverless*. O modelo consegue lidar com estruturas complexas, como laços, ciclos, auto-relacionamentos e paralelismo, utilizando uma representação estendida de DAGs. Adicionalmente, os autores propõem uma heurística baseada em uma abordagem gulosa, que avalia iterativamente diferentes configurações de memória para as funções *serverless* a fim de minimizar o tempo de resposta, respeitando uma restrição orçamentária predefinida. Experimentos realizados nos ambientes do *AWS Lambda* e *Microsoft Azure*, demonstraram a precisão das estimativas do modelo. Kotni et al. [2021] propõem o Faastlane, uma abordagem que busca otimizar a execução das atividades do *workflow* dentro de uma única instância de contêiner *serverless*. Atividades sequenciais são executadas como *threads* dentro de um único processo, enquanto atividades com potencial de paralelismo são executadas em processos distintos ou em múltiplos contêineres.

Li et al. [2022] propõem o FaaSFlow e Li et al. [2023] propõem o DataFlower, ambos são soluções para escalonamento de *workflows* em ambientes *serverless*. O FaaSFlow tem um escalonamento baseado em heurística e aplica o conceito de localidade de dados e armazenamento adaptativo (*Fast Store*), privilegiando a execução de funções em agrupamentos de nós *workers* que tenham os dados de entrada disponíveis. Expandindo esse conceito, o DataFlower explora o paradigma de *dataflow* para a orquestração de *workflows serverless*. Na abordagem *data-flow*, a orquestração do *workflow* é implícita, logo as funções são disparadas de acordo com a disponibilidade dos dados de entrada. Além disso, o DataFlower propõe uma divisão lógica dos contêineres em *Function Logic Unit* (FLU) e *Data Logic Unit* (DLU), permitindo a transmissão assíncrona de dados e a sobreposição entre computação e comunicação. Ambos os trabalhos reforçam a tendência de repensar a orquestração tradicional em ambientes *serverless* para alcançar maior eficiência (redução de latência e aumento da vazão da dados).

Os trabalhos de Ramesh et al. [2023], Burckhardt et al. [2022], Majewski et al. [2021], Kumari et al. [2023], Kotni et al. [2021], Li et al. [2022] e Li et al. [2023] exploram a execução de *workflows* inteiramente em ambientes *serverless*. De forma similar à FLEXE, Majewski et al. [2021] consideram tanto o tempo de execução do *workflow*, quanto o custo financeiro como parâmetros para o escalonador. Malawski [2016] e Malawski et al. [2020] discutem modelos que contemplem características híbridas, mas não apresentam algoritmos ou heurísticas de escalonamento específicas para ambientes heterogêneos, visando múltiplos critérios de otimização, como proposto em nosso trabalho. O sistema *Mashup*, introduzido por Roy et al. [2021], apresenta a possibilidade de escalonamento de tarefas tanto em funções *serverless* quanto em VMs, concentra-se na minimização do tempo total de execução do *workflow*. Esta é uma distinção importante em relação à nossa heurística construtiva FLEXE, que opera em um ambiente híbrido permitindo que as ativações do *workflow*

sejam dinamicamente alocadas entre VMs e funções *serverless*, buscando otimizar o tempo de execução ou o custo financeiro conforme a balanceamento definido pelo usuário.

7. Conclusão e Trabalhos Futuros

Este artigo apresenta a heurística construtiva FLEXE, voltada ao escalonamento eficiente de ativações de *workflows* em ambientes de nuvem que combinam VMs e funções *serverless*. A abordagem busca explorar, de forma flexível e configurável por parte do usuário, as características complementares desses dois tipos de recursos, permitindo ao usuário priorizar entre a minimização do custo financeiro ou do tempo total de execução do *workflow*. A heurística considera restrições operacionais importantes, como os limites de execução das funções *serverless*, bem como diferenças de desempenho e custo entre as alternativas computacionais disponíveis.

A avaliação experimental, conduzida com base em um *workflow* real da área de filogenômica, demonstrou que a FLEXE é eficaz em adaptar-se aos critérios de otimização definidos pelo usuário. Os resultados obtidos confirmam que, ao priorizar o custo, a heurística gera escalonamentos com menor despesa financeira, enquanto, ao priorizar o tempo, são produzidas soluções com significativa redução no tempo de execução do *workflow*. Em configurações sem prioridade definida, a FLEXE obtém soluções de compromisso, que equilibram ambas as métricas de forma satisfatória. Como trabalhos futuros, pretende-se ampliar a abordagem para considerar ambientes de múltiplos provedores de nuvem (*multi-cloud*) e a aplicação de metaheurísticas como o GRASP. Outro trabalho futuro envolve a integração da heurística FLEXE em SWfs existentes, como Pegasus e SciCumulus, de modo a facilitar sua adoção pela comunidade científica.

Referências

- Bouamama, S. e Blum, C. (2017). A population-based iterated greedy algorithm for the knapsack problem with setup. In *MIC'17*, p. 558–565.
- Bouganin, L., Florescu, D., e Valduriez, P. (1996). Dynamic load balancing in hierarchical parallel database systems. In *VLDB'96*, p. 436–447. Morgan Kaufmann.
- Burckhardt, S., Chandramouli, B., Gillum, C., Justo, D., Kallas, K., McMahon, C., Meiklejohn, C. S., e Zhu, X. (2022). Netherite: efficient execution of serverless workflows. *Proc. VLDB Endow.*, 15(8):1591–1604. ISSN 2150-8097.
- Davari, S. (1985). *Scheduling periodic - time-critical tasks on multiprocessor computing systems (np-hard, cyclic, task allocation)*. PhD thesis, The University of Oklahoma. AAI8603509.
- de Oliveira, D., Liu, J., e Pacitti, E. (2019). *Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments*. Springer.
- de Oliveira, D., Ogasawara, E. S., Baião, F. A., e Mattoso, M. (2010). Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In *IEEE CLOUD 2010, Miami, FL, USA, 5-10 July, 2010*, p. 378–385. IEEE Computer Society. ISBN 978-1-4244-8207-8.
- Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P., Mayani, R., Chen, W., da Silva, R. F., Livny, M., e Wenger, R. K. (2015). Pegasus, a workflow management system for science automation. *Future Generation Comp. Syst.*, 46:17–35.
- Farahani, R. et al. (2024). Serverless workflow management on the computing continuum: A mini-survey. In *ICPE '24 Companion*, p. 146–150, New York, NY, USA. ACM. ISBN 9798400704451.

- Freire, J., Koop, D., Santos, E., e Silva, C. T. (2008). Provenance for Computational Tasks: A Survey. *Computing in Science & Engineering*, p. 20–30.
- Garofalo, M. et al. (2024). Workflow engines in the compute continuum: a comparative analysis. In *UCC '23*, New York, NY, USA. ACM. ISBN 9798400702341.
- Hassan, H. B., Barakat, S. A., e Sarhan, Q. I. (2021). Survey on serverless computing. *J. Cloud Comput.*, 10(1):39.
- Kotni, S., Nayak, A., Ganapathy, V., e Basu, A. (2021). Faastlane: Accelerating Function-as-a-Service workflows. In *ATC'21*, p. 805–820. USENIX. ISBN 978-1-939133-23-6.
- Kumari, A., Sahoo, B., e Behera, R. K. (2023). Workflow aware analytical model to predict performance and cost of serverless execution. *Concurrency and Computation: Practice and Experience*, 35(22):e7743.
- Li, Z. et al. (2022). Faasflow: enable efficient workflow execution for function-as-a-service. In *ASPLOS '22*, p. 782–796, New York, NY, USA. ACM. ISBN 9781450392051.
- Li, Z., Xu, C., Chen, Q., Zhao, J., Chen, C., e Guo, M. (2023). Dataflower: Exploiting the data-flow paradigm for serverless workflow orchestration.
- López-Ibáñez, M. et al. (2016). The IRACE package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.
- Majewski, M., Pawlik, M., e Malawski, M. (2021). Algorithms for scheduling scientific workflows on serverless architecture. In *CCGrid'21*, p. 782–789.
- Malawski, M. (2016). Towards serverless execution of scientific workflows – hyperflow case study. URL <https://api.semanticscholar.org/CorpusID:8229008>.
- Malawski, M. et al. (2020). Serverless execution of scientific workflows: Experiments with hyperflow, aws lambda and google cloud functions. *FGCS*, 110:502–514. ISSN 0167-739X.
- Mampage, A. et al. (2021). A holistic view on resource management in serverless computing environments: Taxonomy, and future directions. *CoRR*, abs/2105.11592. URL <https://arxiv.org/abs/2105.11592>.
- Maschler, J., Hackl, T., Riedler, M., e Raidl, G. R. (2017). An enhanced iterated greedy metaheuristic for the particle therapy patient scheduling problem. In *MIC'17*, p. 465–474.
- Moraes, J. V. et al. (2025). Exploring evolutionary patterns: A jupyter notebook for discovering frequent subtrees in phylogenetic tree databases. To Appear - JIDM.
- Ramesh, M., Chahal, D., Phalak, C., e Singhal, R. (2023). Multi-objective workflow scheduling to serverless architecture in a multi-cloud environment. In *IC2E*, p. 173–183.
- Roy, R. B., Patel, T., e Tiwari, D. (2021). Mashup: Making serverless computing useful for hpc workflows. URL <https://doi.org/10.5281/zenodo.5733395>.
- Srirama, S. N., Ivanistsev, V., Jakovits, P., e Willmore, C. (2013). Direct migration of scientific computing experiments to the cloud. In *HPCS*, p. 27–34.
- Teylo, L. et al. (2017). A hybrid evolutionary algorithm for task scheduling and data assignment of data-intensive scientific workflows on clouds. *Future Gener. Comput. Syst.*, 76:1–17.