# Otimização do Agendamento de Tarefas em Plataformas de Integração de Aplicações Empresariais Baseada na Meta-heurística Particle Swarm Optimization

Daniela F. Sellaro
UNIJUÍ Ijuí RS
dselllaro@unijui.edu.br

Rafael Z. Frantz
UNIJUÍ Ijuí RS
rzfrantz@unijui.edu.br

Inma Hernández
Universidade de Sevilha Espanha
inmahernandez@us.es

Fabricia Roos-Frantz
UNIJUÍ Ijuí RS
frfrantz@unijui.edu.br

Sandro Sawicki
UNIJUÍ Ijuí RS
sawicki@unijui.edu.br

## ABSTRACT

As empresas buscam alternativas tecnológicas que proporcionem competitividade para seus processos de negócios. Dentre essas alternativas, existem as plataformas de integração que conectam as aplicações que compõem seu ecossistema de software, o qual é composto pelas aplicações locais e pelos serviços da computação em nuvem, como SaaS e PaaS, e ainda interage com as mídias sociais. As plataformas de integração implementam soluções de integração, que fazem com que as aplicações funcionem de forma sincronizada, oferendo acesso às informações e funcionalidades de forma rápida e segura. As plataformas de integração geralmente fornecem uma linguagem de domínio específico, um kit de ferramentas de desenvolvimento, um motor de execução e uma ferramenta de monitoramento. A eficiência do motor em agendar e executar tarefas de integração tem um impacto direto no desempenho de uma solução e este é um dos desafios enfrentados pelas plataformas de integração. Nossa revisão da literatura identificou que os motores de integração adotam algoritmos de agendamento de tarefas baseados em políticas, como a de prioridade e a *First-In-First-Out*, as quais não se adequam a alguns tipos de cenários, como no caso do aumento da carga de trabalho inicial. Portanto, é oportuno buscar um algoritmo de agendamento de tarefas que otimize desempenho da solução de integração em diferentes cenários. Esse artigo propõe um algoritmo agendamento de tarefas baseado na técnica de otimização meta-heurística *Particle Swarm Optimization*, que atribui as tarefas para os recursos computacionais, considerando o tempo de espera na fila de tarefas prontas e a complexidade computacional de cada tarefa, a fim de otimizar o desempenho da solução de integração.

## KEYWORDS

Integração de Aplicações Empresariais; Otimização; Computação em Nuvem; Motor de Execução; Algoritmo de Agendamento de Tarefas.


*SBES 2017, Fortaleza, Ceará Brasil*
 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUÇÃO

As empresas possuem um ecossistema de software composto por diversas aplicações que geralmente são desenvolvidas internamente ou adquiridas de terceiros. O avanço das tecnologias de desenvolvimento de aplicações e a incorporação de serviços de software disponíveis na internet têm deixado os ecossistemas de software ainda mais heterogêneos. Os processos de negócio de uma empresa precisam ser suportados por um conjunto de aplicações e serviços de software que integram seu ecossistema, porém, frequentemente, tais aplicações e serviços não estão preparados para trabalhar de forma conjunta. A Integração de Aplicações Empresariais (EAI) é o campo de estudo que oferece metodologias, técnicas e ferramentas para que os processos de negócio funcionem de forma sincronizada, promovendo repostas rápidas e confiáveis.

As plataformas de integração são softwares especializados que permitem projetar, executar e monitorar soluções de integração, as quais conectam funcionalidades e dados de diferentes aplicações. Uma solução de integração implementa um fluxo de integração composto por distintas tarefas atômicas que são executadas ao longo desse fluxo. Gregor Hophe e Bobby Woolf [15] documentaram um conjunto de padrões de integração que tem inspirado o desenvolvimento de plataformas de integração de código aberto e que por sua vez organizam o fluxo de integração seguindo uma arquitetura Pipes&Filters [3]. Dentre essas plataformas, destacam-se Camel [17], Spring [11], Mule [8], Guaraná [13], Jitterbit [22] e WSO2 ESB [18]. Usualmente, essas plataformas fornecem uma linguagem de domínio específico, um kit de ferramentas de desenvolvimento, um motor de execução e uma ferramenta de monitoramento. A linguagem de domínio específico possibilita a descrição de modelos conceituais para soluções de integração. O kit de desenvolvimento é um conjunto de ferramentas de software que permite a implementação de soluções, ou seja, transforma uma solução conceitual em código executável. O motor proporciona todo o suporte necessário à execução das soluções de integração. A ferramenta de monitoramento é utilizada para detectar erros que possam ocorrer durante a execução de soluções de integração. O motor é o responsável pela execução das soluções de integração [13].

As tarefas da solução de integração são executadas por meio de recursos computacionais presentes no motor de execução, dentre os quais estão as *threads* de execução. Neste contexto, as *threads* são usadas para proporcionar que as tarefas sejam executadas de forma simultânea, por meio da programação *multithreads* [7, 24].

Nesse tipo de programação, a criação de *threads* pode impactar o desempenho da execução de uma solução de integração.

Um algoritmo de agendamento de tarefas inadequado aumenta o tempo de execução, impactando o desempenho da solução de integração. A abordagem mais comum é a contratação de mais recursos computacionais, porém essa alternativa é financeiramente onerosa e nem sempre viável. Nossa revisão da literatura identificou que os motores de integração adotam como política de agendamento de tarefas as políticas de prioridade e a *First-In-First-Out* (FIFO). Há propostas de algoritmos de agendamento de tarefas para máquina virtuais em sistemas distribuídos [21, 2], porém não foram identificadas propostas que foquem na otimização de desempenho de motores de execução de plataformas de integração de sistemas. Este trabalho apresenta um algoritmo que atribui as tarefas para as *threads*, considerando o tempo de espera na fila de tarefas prontas e a complexidade computacional de cada tarefa, utilizando a meta-heurística *Particle Swarm Optimization* (PSO).

O resto deste artigo está organizado como segue. A Seção 2 apresenta a formulação do problema. A Seção 3 descreve sucintamente a técnica PSO. A Seção 4 expõe a abordagem proposta. E a Seção 5 apresenta nossas conclusões e perspectivas de trabalhos futuros.

## 2 FORMULAÇÃO DO PROBLEMA

Numa solução de integração, baseada no estilo arquitetural *Pipes&Filters* [3], os *pipes* são representados por canais de mensagens e os *filters* por tarefas atômicas que implementam um padrão de integração concreto e processam dados encapsulados em mensagens. A Figura 1 mostra um exemplo mostra o modelo conceitual de uma solução de integração para o problema Café, introduzido por Gregor Hohpe [16] modelado com Guaraná [13]. O modelo conceitual pode ser representado como fluxos de trabalho modelados como Grafos Direcionados Acíclicos, definidos por W(T,E), onde $T = \{t_1, t_2, ..t_n\}$ é o conjunto de tarefas e $E$ é o conjunto de arestas direcionadas. Uma aresta $e_{ij}$ da forma $(t_i, t_j)$ existe, se houver uma dependência de dados entre $t_i$ e $t_j$, onde $t_i$ é tarefa pai de $t_j$ e $t_j$ é tarefa filha de $t_i$. Logo, uma tarefa filha não pode ser executada até que todas as suas tarefas pai estejam concluídas. A Figura 2 mostra fluxos de trabalho da solução de integração Café, onde cada vértice representa uma tarefa e cada aresta possui um peso, que representa o tempo de espera da mensagem na fila de tarefas prontas. Considera-se que o motor de execução oferece uma variedade de recursos computacionais para execução das tarefas da solução de integração. Essa variedade de recursos é definida por *pool* de *threads*, com diferentes números de *threads*, onde uma *thread* é a unidade básica de processamento. Assim, assume-se que um motor de execução pode ter mais de um *pool* de *threads* e que o número de *threads* em cada um pode ser diferente, ou seja, um *pool* com uma quantidade $x$ *threads* e outro com uma quantidade $y$. Considera-se ainda, que um motor de execução tem uma quantidade limitada de recursos computacionais $\delta_r$, sendo um recurso computacional *Pool*, um *pool* de *threads* que tem um tipo $Pool_i$, uma capacidade de processamento $PPool_j$. O tipo diferencia os *pools*, a capacidade de processamento é a quantidade de *threads* do *pool*. A capacidade de memória não é tratada; assume-se que é suficiente para executar as tarefas do fluxo de trabalho. Assume-se que para cada tipo de recurso, a capacidade de processamento é definida em termos de instruções por ciclo (IPC), que pode ser estimada [1]. Esta informação é usada no algoritmo, para calcular o tempo de execução de uma tarefa em um determinado *pool* de *threads Pool*. A variação do desempenho pode ser modelada pelo ajuste da capacidade do *Pool* e introduzindo uma degradação de desempenho $deg_{Pool_j}$ [21]. O tempo de execução $TE_{t_i}^{Pool_j}$ da tarefa $t_i$ em um *Pool* de tipo $Pool_j$ é estimado pelo tamanho $Tam_{t_i}$ da tarefa em instruções por ciclo (IPC), calculado pela Equação 1.

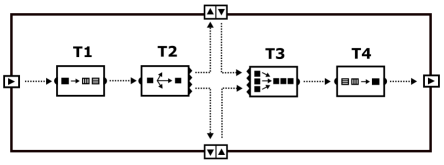


**Figura 1: Solução de integração Café**

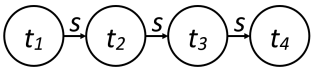


**Figura 2: Workflow da solução de integração Café.**

$$TE_{t_i}^{Pool_j} = Tam_{t_i}/(P_{Pool_j} * (1 - \deg_{Pool_j})) \quad (1)$$

O tempo médio de espera na fila de tarefas $TFi_{e_{ij}}$ é definido como o tempo para transferir dados entre uma tarefa pai $t_i$ e sua tarefa filha $t_j$ e assume-se que ele pode ser monitorado e medido.

Finalmente, o tempo total de processamento $TP_{t_i}^{Pool_j}$ de uma tarefa em um *Pool* é calculado na Equação 2, onde k é o número de arestas, $t_i$ é uma tarefa pai e $s_k$ representa o tempo gasto pelo motor na troca de *pool* de threads, de maneira que $s_k = 0$, quando $t_i$ e $t_j$ são processadas no mesmo *pool* e $s_k = 1$, caso contrário.

$$TP_{t_i}^{Pool_j} = TE_{t_i}^{Pool_j} + (\sum_{1}^{k} TF_{ij} + s_k) \quad (2)$$

O objetivo é encontrar um agendamento de tarefas que possibilite executar as tarefas da solução de integração em *pools* de *threads* do motor de execução, minimizando o tempo total de execução e sem aumentar a quantidade de recursos computacionais. O agendamento de tarefas é definido por $A = (R, M, TR, TTE)$, sendo $R$ um conjunto de recursos; $M$ o mapeamento de tarefas em recursos, $TR$, $TR = |R| = n$, o total de recursos, $TTE$ o tempo total de execução. Um exemplo é mostrado na Figura 3, representando o agendamento para o fluxo de trabalho da Figura 2, em que cada uma das quatro tarefas é mapeada para ser executada por um dos três recursos disponíveis, e onde as tarefas pais são executadas antes das sua tarefas filhas, mantendo assim a dependência dos dados. $R = \{r_1, r_2, ..., r_n\}$ é o conjunto de recursos (*pools* de *threads*) do motor de execução, onde cada recurso $r_i$ tem associado a ele um *Pool* do tipo $Pool_i$, um tempo de início estimado para alocação do recurso estimado $TIni_{r_i}$. e um tempo de finalização estimado $TFim_{r_i}$. M representa um mapeamento para cada uma das tarefas do fluxo de trabalho e é constituído por tuplas $m_{t_i}^{r_j} = (t_i, r_j, TIni_{t_i}, TFin_{t_i})$, significando

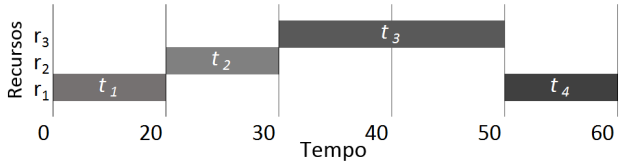


**Figura 3: Agendamento para o workflow do Café.**

que a tarefa $t_i$ será executada pelo recurso $r_j$, com o início da execução agendado para $TIni_{t_i}$ e previsão de término em $TFin_{t_i}$. A Equação 3 mostra como o tempo total de execução $TTE$ é calculado:

$$TTE = \max\{TFim_{t_i} : t_i \in T\} \tag{3}$$

Assim, o problema pode ser formulado como: *encontrar um agendamento A com o menor tempo total de execução TTE da solução de integração, sem exceder um valor pré-definido para o total de recursos TR*. Essa formulação é representada pela Equação 4:

$$Minimize TTE \tag{4}$$
$$\text{sujeito a } TR \leq \delta_r$$

## 3 PARTICLE SWARM OPTIMIZATION

*Particle Swarm Optimization* (PSO) foi introduzido por Kennedy e Eberhart em 1995 [9], e foi inspirado no comportamento social de organismos biológicos, mais especificamente na habilidade de algumas espécies de animais de trabalhar em conjunto para localizar boas regiões com fontes de alimento, assim como ocorre em cardumes e em bandos de pássaros [5]. Em outras palavras, é baseado em um enxame de partículas (*Particle Swarm*) que se movem pelo espaço e se comunicam para determinar uma direção de busca ideal. O PSO tem melhor desempenho computacional para este tipo de problema de otimização de funções não-lineares de alta dimensionalidade com variáveis contínuas, do que outros algoritmos [9, 5, 4] e tem menos parâmetros para ajustar, facilitando sua implementação. O PSO vem sendo utilizado com sucesso na solução de problemas da ciência e da engenharia devido à sua simplicidade, eficácia e robustez [14, 20, 23, 25, 10, 4, 21, 2].

Cada partícula $i$ do enxame $S$ é representada por sua posição e sua velocidade. A posição é um vetor de $n$ dimensões, cujos componentes representam os parâmetros da função objetivo. As partículas controlam a sua melhor posição *pbest* e a melhor posição global *gbest*, a melhor solução conhecida dentro de sua vizinhança.

Inicialmente, as partículas do enxame possuem posições aleatórias no espaço de busca, obedecendo uma distribuição de probabilidade uniforme. Posteriormente, a posição $x_i\,(t)$ de cada partícula $i$ na iteração $t$ é modificada por uma velocidade estocástica $v_i\,(t)$ que depende da distância que a partícula está da sua melhor solução conhecida e da distância para a melhor solução conhecida dentro de sua vizinhança. Cada partícula $i \in S$ se movimenta em cada dimensão $j \in \{1, 2, ..n\}$ do espaço de busca em um instante discreto de tempo $t$, segundo as Equações 5 e 6:

$$\overrightarrow{x}_i\,(t+1) = \overrightarrow{x}_i\,(t) + \overrightarrow{v}_i\,(t) \tag{5}$$

$$\overrightarrow{v}_i\,(t+1) = w\overrightarrow{v}_i\,(t) + c_1 r_1 \left(\overrightarrow{x}_i^*\,(t) - \overrightarrow{x}_i\,(t)\right) + c_2 r_2 \left(\overrightarrow{x}^*\,(t) - \overrightarrow{x}_i\,(t)\right) \tag{6}$$

onde: $w$ = inércia

$c_i$ = coeficiente de aceleração, $i = (1, 2)$

$r_i$ = número aleatório pertencente a uma distribuição de probabilidade uniforme , $i = (1, 2)$ e $r_i \in [0, 1]$

$\overrightarrow{x}_i^*\,(t)$ = melhor posição da partícula $i$

$\overrightarrow{x}^*\,(t)$ = posição da melhor partícula da população

$\overrightarrow{x}_i\,(t)$ = posição atual da partícula $i$

Quando a vizinhança das partículas consiste no enxame inteiro, a posição $\overrightarrow{x}_i\,(t)$ é denominada de *gbest*. O vetor velocidade é quem orienta o processo de otimização, usando tanto o conhecimento adquirido particularmente pela partícula quanto o conhecimento adquirido pela partícula baseada na interação com sua vizinhança. O termo $c_1 r_1 \left(\overrightarrow{x}_i^*\,(t) - \overrightarrow{x}_i\,(t)\right)$ da equação de atualização da velocidade é a componente cognitiva e representa a experiência da partícula. Essa componente é a responsável pela tendência que a partícula tem de voltar para a melhor solução encontrada por ela no passado. O termo $c_2 r_2 \left(\overrightarrow{x}^*\,(t) - \overrightarrow{x}_i\,(t)\right)$, por sua vez, é conhecido como componente social da equação da velocidade, e representa o conhecimento coletivo do enxame, sendo responsável por atrair cada partícula para a melhor solução encontrada por alguma partícula de sua vizinhança. A Figura 4 mostra a representação do movimento de uma partícula em um espaço de busca de dimensão igual a 2. O vetor velocidade $\overrightarrow{v}_i$ é a soma vetorial das componentes

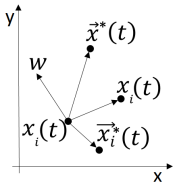


**Figura 4: Movimento da partícula em um espaço de busca.**

cognitiva e social com a inércia da partícula. A inércia $w$ atua como uma memória da direção da velocidade anterior da partícula e impede que haja alterações bruscas na direção da velocidade partícula [10]. Assim, caso a partícula esteja se dirigindo a uma boa região, a descoberta de um novo líder social não alterará completamente essa direção. O papel da componente cognitiva é aproximar a partícula na direção da melhor posição encontrada por ela desde o início da busca. Dessa forma, a partícula deverá encontrar uma boa posição que provavelmente está próxima do seu líder cognitivo atual. Através da componente social, as partículas comunicam a informação sobre as melhores posições encontradas por elas desde o início do processo de busca, sendo considerada como a mais importante componente na equação da velocidade das partículas.

O pseudo-código do PSO é apresentado no Algoritmo 1. A cada passo, o algoritmo irá mudar a velocidade de cada partícula em direção às posições *pbest* e *gbest*, onde a posição e a velocidade da partícula são atualizadas conforme as Equações 5 e 6, respectivamente. Um termo aleatório $r$ pondera o quanto a partícula se movimenta em direção a esses valores, onde diferentes números aleatórios são gerados em direção à aceleração para *pbest* e *gbest* locais [19]. O algoritmo continuará a iterar até que um critério de parada seja alcançado. Geralmente, esse critério de parada é um número máximo de iterações especificado ou um valor aptidão

pré-definido considerado bom o suficiente. A equação de velocidade contém vários parâmetros que afetam o desempenho do algoritmo e alguns afetam significativamente na convergência do algoritmo. A inércia $w$, por exemplo, é fundamental para a convergência do algoritmo. Ela determina o quanto as velocidades anteriores afetarão a velocidade atual e definirá um equilíbrio entre o componente cognitivo local e o global social das partículas. Se o valor da inércia for alto, a velocidade aumentará, favorecendo a busca global. Mas, se o valor for baixo, as partículas sofrerão uma desaceleração, favorecendo a busca local. Dessa forma, um valor $w$ que equilibra a pesquisa global e local implica menos iterações para que o algoritmo possa convergir. Apesar de $c_1$ e $c_2$ não influenciarem diretamente na convergência do PSO, o ajuste desses parâmetros agiliza e evita que o algoritmo seja pego em mínimos locais. O parâmetro $c_1$ é referido como parâmetro cognitivo e o valor $c_1r_1$, na Equação 6, define a relevância da melhor posição anterior. $c_2$ é referido como o parâmetro social e $c_2r_2$ e determina o comportamento da partícula em relação a outros vizinhos.

**Algorithm 1** *Particle Swarm Optimization*

1: $d \leftarrow n$ ▷ Inicializa a dimensão das partículas para $d$
2: $x[i] \leftarrow x_{aleatorio}$ ▷ Inicializa a população de partículas com posições e velocidades aleatórias $v[i] \leftarrow v_{aleatorio}$
3: **while** $criterioparada = falso$ **do** ▷ Repete enquanto o critério de parada não tiver sido alcançado
4: **for** i **do**1n ▷ Para cada partícula calcula o seu valor aptidão, compara o valor aptidão da partícula com o valor *pbest* e com o valor de *gbest*
5: **if** $x_{atual} \leftrightarrow pbest$ **then** ▷ Se o valor atual da partícula é melhor do que *pbest*
6: $x[i] \leftarrow x_{atual}$ $pbest \leftarrow x_{atual}$
7: **end if**
8: **if** $x_{atual} \leftrightarrow gbest$ **then** ▷ Se o valor atual da partícula é melhor do que *gbest*
9: $x[i] \leftarrow x_{atual}$ $gbest \leftarrow x_{atual}$
10: **end if**
11: $x[i] \leftarrow x_{calculada}$ ▷ Atualiza a posição e velocidade da partícula Equações 5 e 6
$v[i] \leftarrow v_{calcculada}$
12: **end for**
13: **end while**

O número, a dimensão das partículas, o alcance e a velocidade máxima das partículas são parâmetros utilizados como entrada para o algoritmo, embora não componham a definição de velocidade. Quanto ao número de partículas, um valor alto costuma aumentar a probabilidade de encontrar o ótimo global. O valor deste número depende da complexidade do problema de otimização, mas um intervalo típico é entre 20 e 40 partículas [21]. Os valores para a dimensão das partículas e o alcance, no qual sua movimentação é permitida, são determinados unicamente pela natureza do problema que está sendo resolvido e de como ele é modelado para se adequar ao PSO. A velocidade máxima define a mudança máxima que uma partícula pode ter em uma iteração e normalmente seu valor é aproximadamente a metade do alcance de posição da partícula [21].

# 4 ALGORITMO

A modelagem de um problema PSO é dividida em duas fases: definição do problema e definição da função aptidão. A primeira consiste em definir como o problema será codificado, ou seja, definir como a solução será representada. A segunda, em definir o quão *boa* uma partícula será medida, ou seja, definir a função de aptidão. Já para transformar o problema PSO em um algoritmo, é preciso definir a partícula e sua dimensão. Na abordagem adotada neste artigo, uma partícula representa um fluxo de trabalho e suas tarefas, e dimensão da partícula representa o número de tarefas no fluxo de trabalho. A dimensão de uma partícula serve para localizar sua posição no espaço, definindo o sistema de coordenadas. No exemplo apresentado na Figura 2, a dimensão da partícula é dez, sendo sua posição especificada por um sistema com quatro coordenadas.

Uma partícula movimenta-se num espaço limitado, denomina-se alcance. Na abordagem adotada neste artigo, o alcance é determinado pelo número de *pools* de *threads* disponíveis para executar a tarefa. Assim, o valor de uma coordenada no sistema de coordenadas, que define o espaço de movimentação da partícula, tem um alcance de 0 até o número máximo de *pools* de *threads* disponíveis. A parte inteira do valor de uma coordenada na posição de uma partícula corresponde ao número de *pools* de *threads* e representa o recurso computacional atribuído a uma tarefa definida por essa coordenada específica. Assim, a posição da partícula corresponde a um mapeamento da tarefa em recursos.

A Figura 5 apresenta um exemplo de valores para as dez coordenadas do nosso sistema de coordenadas, no qual existem três recursos computacionais (*pools* de *threads* disponíveis), de forma que o valor de cada coordenada poderá variar entre 0 e 3. Há 4 tarefas para serem mapeadas, o que corresponde a dimensão 4 da partícula, portanto a posição da partícula terá 4 coordenadas. O índice da coordenada (de 1 a 4) corresponde a uma tarefa (de $t_1$ a $t_4$). O valor da coordenada é um número real de 0 a 3 , correspondendo ao número de recursos disponíveis para cada tarefa, com o máximo de 3 no nosso exemplo. Quando esse número inteiro é arredondado, é determinado o número de recursos disponíveis para cada tarefa. Conforme o exemplo da Figura 5, a coordenada 1 da tarefa 1 tem valor igual a 1, 3 significando que para essa tarefa foi alocado 1 recurso, pois 1 é a parte inteira desse valor; a coordenada 2, corresponde a tarefa 2, tem valor de 2, 0 indica que para a tarefa 2 foram alocados 2 recursos; e assim, sucessivamente até a coordenada 4. A função

| Coordenada | ***Coord.1*** | ***Coord.2*** | ***Coord.3*** | ***Coord.4*** |
|---|---|---|---|---|
| Valor | 1,3 | 2,0 | 3,0 | 1,4 |
| Tarefa | $t_1$ | $t_2$ | $t_3$ | $t_4$ |
| Quant. de recursos | 1,3 | 2,0 | 3,0 | 1,4 |

**Figura 5: Coordenadas da posição das partículas no espaço.**

de aptidão deve refletir os objetivos do problema de agendamento, pois ela é usada para determinar o quão boa uma determinada solução é. Nessa abordagem, ela é minimizada e seu valor será o tempo total de execução $TTE$ contido no agendamento $A$ derivado da posição da partícula. Considerando que o motor de execução é capaz de aumentar elástica e dinamicamente a quantidade de

recursos, o modelo de aquisição oferecido pela computação em nuvem parece ilimitado, ou seja, não há um conjunto de recursos disponíveis que pode ser usado no algoritmo. A estratégia é definir um conjunto inicial de recursos que o algoritmo pode usar para explorar diferentes soluções e alcançar o agendamento. O tamanho desse conjunto será nossa restrição $TR \leq \delta_r$, conforme a Equação 4. Tal estratégia tem de refletir a heterogeneidade dos *pools* de *threads* (diferentes números de *threads*) e oferecer opções suficientes de PSO, a fim de que seja produzida uma partícula adequada, ou seja, a solução. O conjunto de recursos inicial e limitado conterá os recursos que podem ser usados. Se esse conjunto é muito grande, o número de agendamento possíveis aumentará, assim como, o espaço de pesquisa explorado pelo PSO, tornando difícil para o algoritmo convergir e encontrar uma solução adequada [21].

Para reduzir o tamanho do espaço de pesquisa, o qual será usado pelo PSO para encontrar um agendamento próximo do ótimo, considera-se um conjunto de recursos inicial $R_{inicial}$, composto por um *Pool* de cada tipo, para cada tarefa em P; onde P é o conjunto que contém o número máximo de tarefas que podem ser executadas em paralelo para um dado fluxo de trabalho. O algoritmo selecionará o número e o tipo apropriados de *Pools* para o motor de execução, dentro das opções contidas em $R_{inicial}$. Assim, reflete-se a heterogeneidade dos recursos computacionais e reduz-se o tamanho do espaço de busca, além de permitir mapear todas as tarefas que podem ser executadas em paralelo. O tamanho de $R_{inicial}$ seria igual a $|P| * n$, onde $n$ é o número de tipos de *Pools* disponíveis, sendo ainda possível, que o PSO selecione mais de um recurso $|P|$, se necessário (a menos que $n = 1$).

O problema abordado possui a restrição $TR \leq \delta_r$, utiliza-se uma versão do PSO que incorpora uma estratégia para tratar restrições [6]. Nela, sempre que duas soluções estão sendo comparadas e (*i*) ambas as soluções forem viáveis, então a solução com melhor aptidão é selecionada; (*ii*) se uma solução é viável e a outra não é, então a viável é selecionada; e finalmente, (*iii*) se ambas as soluções são inviáveis, aquela que violar menos a restrição é selecionada. No último caso, implica que uma medida de quanto uma solução viola a restrição precisa ser encontrada. O problema define o valor de violação de restrição de uma solução $\delta_k$, a restrição pode ser atribuída a limitação de recursos que se deseja contratar na computação em nuvem ou simplesmente e a quantidade de *threads* físicas das máquinas que irão ser utilizadas na execução da solução de integração. Uma solução do PSO que utilize recursos próximos de $\delta_k$ será preterida em relação a uma solução menos recursos.

O Algoritmo 2 apresenta o pseudo-código para mapear a posição de uma partícula em um agendamento. O conjunto de recursos $R$ para serem alocados e o conjunto de mapeamentos $M$ de tarefas para recursos são inicializados sem elementos, ou seja, vazios, e tempo total de execução $TTE$ é inicializado com o valor zero. Na sequência, o algoritmo estima o tempo de execução de cada tarefa do fluxo de trabalho para todo o recurso $r_i \in R_{inicial}$. A representação é uma matriz em que as linhas representam as tarefas, as colunas representam os recursos e a entrada $TempoExec[i, j]$ corresponde ao tempo gasto para executar a tarefa $t_i$ no recurso $r_j$, calculado conforme a Equação 1. O próximo passo é o cálculo ou atribuição da matriz de tempo de transferência de dados, ou tempo de espera na fila de tarefas, o qual assume-se que está sendo obtido por um procedimento, não tratado nesse trabalho, tal como por uma ferramenta de monitoramento. Essa matriz é representada como uma matriz de adjacência ponderada do workflow DAG, onde a entrada $TempoTransfer[i, j]$ contém o tempo que leva para transferir os dados de saída da tarefa $t_i$ para a tarefa $t_j$ e esse valor é zero sempre que $i = j$ ou não há aresta direcionada conectando $t_i$ (tarefa pai) e $t_j$ (tarefa filha). A Figura 6 exemplifica essas matrizes para o workflow apresentado na Figura 2. De posse dessas informações, o

$$tempoExec = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{array} \begin{array}{c} \begin{array}{ccc} r_1 & r_2 & r_3 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \\ 3 & 3 & 3 \end{bmatrix} \end{array} \quad tempoTransfer = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \end{array} \begin{array}{c} \begin{array}{cccc} t_1 & t_2 & t_3 & t_4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

**Figura 6: Matrizes Tempo de Execução e de Transferência.**

algoritmo inicia determinando a posição da partícula e construindo o agendamento. Para isso, itera para toda *i* do *array* de posição *pos* e atualiza $R$ e $M$. Primeiro, determina a tarefa e o recurso que está associado à coordenada atual e seu valor. A estratégia usada para isso é a descrita anteriormente, a qual indica que a coordenada $i$ corresponde à tarefa $t_i$ e seu valor $pos[i]$ corresponde ao recurso $r_{pos[i]} \in R_{inicial}$. Encontrados os componentes, $t_i$ e $r_j$, de uma tupla de mapeamento $m_{t_i}^{r_j}$, o algoritmo calcula os demais, o tempo de inicio $TIni_{t_i}$ e tempo de finalização $TFin_{t_i}$ da tarefa.

O valor de tempo de início $TIni_{t_i}$ diferencia-se em duas situações. Na primeira, a tarefa não tem tarefa pai e, portanto, pode começar a ser executada, assim que o recurso alocado para ela $r_{pos[i]}$ estiver disponível, o que ocorrerá quando o referido recurso terminar a execução que estiver em andamento. Na segunda situação, a tarefa tem um ou mais pais e, nesse caso, além de esperar que o recurso para ela alocado esteja disponível, também terá que aguardar pelo término da execução das suas tarefas pai, além do tempo de transferência dos dados.

O valor de $TFin_{t_i}$ é calculado baseado no tempo total de processamento e no tempo de início da tarefa. Para determinar o tempo de processamento $TP_{t_i}^{r_{pos[j]}}$, é necessário calcular o tempo de execução e o tempo de transferência. O primeiro é $TempoExec[i, pos[i]]$, enquanto o último é calculado pela soma dos valores do tempo de transferência $TempoTransf[i, filha(i)]$ para toda tarefa filha $t_{filha(i)}$ de $t_i$, que está mapeada para rodar em um recurso diferente de $r_{pos[i]}$. Esses dois valores são então somados para obter $TP_{t_i}^{r_{pos[j]}}$, como definido na Equação 2. Por fim, o valor de $TFin_{t_i}$ é obtido pela subtração de $TIni_{t_i}$ de $TP_{t_i}^{r_{pos[j]}}$. Calculados os elementos de $m$, adiciona-se recurso para $R$, se necessário. Quando o algoritmo termina de processar, cada coordenada do vetor posição, $R$ conterá todos os recursos necessários e os tempos de início e finalização. Além disso, o mapeamento completo das tarefas para os recursos estará em $M$ e cada tarefa terá um recurso atribuído a ela e o tempo estimado de início e o de término. Com essas informações o algoritmo pode calcular o $TTE$ associado a solução atual, conforme Equação 2. Nesse ponto, o algoritmo calculou $R$, $M$ e $TTE$ e poderá construir e apresentar o agendamento associado para essa posição da partícula. Finalmente, os Algoritmos 1 e 2 são combinados para o agendamento próximo de ótimo. Na linha 4 do

**Algorithm 2** Geração de agendamento

**Entrada:** Conjunto de fluxo de trabalho de tarefas $T$
Conjunto Inicial de recursos $R_{inicial}$
Um array $pos[|T|]$ representando a posição da partícula
**Saída:** Um agendamento $A$

1: $\mathsf{R} = \emptyset, \mathsf{M} = \emptyset, TTE = 0$ ▷ Inicializa componentes
2: Calcula $TempoExec[\,|T| \times |R_{inicial}|]$
3: Calcula $TempoTransf[\,|T| \times |T|]$
4: **for** $i$ **do**$0|T| - 1$
5: $t_i = T[i], r_{pos[i]} = \mathsf{R}_{inicial}[pos[i]]$
6: **if** $t_i$ não tem pai **then**
7: $TIni_{t_i} = TFin_{r_{pos[i]}}$
8: **else**
9: $TIni_{t_i} = (\max\{TFin_{t_{pai}} : t_{pai}[\in pais(t_i)\}, TFin_{r_{pos[i]}})$
10: **end if**
11: $exe = tempoExec\,[i]\,[pos\,[i]]$
12: **for** cada filha $t_{filha}$ de $t_i$ **do**
13: **if** $t_{filha}$ é mapeada para um recurso diferente de $r_{pos[i]}$ **then**
14: $trasfer+ = TempoTransf[i][c]$
15: **end if**
16: **end for**
17: $TP_{t_i}^{r_{pos[j]}} = exe + transf$
18: $TFin_{t_i} = TP_{t_i}^{r_{pos[j]}} - TIni_{t_i}$
19: $m_{t_i}^{r_{pos[i]}} = (t_i, r_{pos[i]}, TIni_{t_i}, TFin_{t_i})$
20: $\mathsf{M} = \mathsf{M} \cup \{m_{t_i}^{r_{pos[i]}}\}$
21: **if** $r_{pos[i]} \notin \mathsf{R}$ **then**
22: $\mathsf{R} = \mathsf{R} \cup r_{pos[i]}$
23: **end if**
24: **end for**
25: Calcula $TTE$ conforme Equação 3
26: $A = (R, M, TR, TTE)$

Algoritmo 1, em vez de calcular o valor de aptidão da partícula, gera-se o agendamento, conforme Algoritmo 2. Em seguida, usa-se o $TTE$ como valor de aptidão nas etapas seguintes e é introduzido o mecanismo de manipulação de restrição, tal que $TR \leq \delta_r$.

## 5 CONCLUSÃO

A eficiência dos motores de execução das plataformas de integração está diretamente relacionado com o algoritmo de agendamento das tarefas e a alocação de *threads* para executá-las. Um algoritmo ineficiente leva a um aumento do tempo de execução, degradando assim o desempenho das soluções de integração. Este artigo propõe um algoritmo baseado na meta-heurística *Particle Swarm Optimization* para a alocação de *threads* em motores de execução baseados em filas FIFO. O algoritmo proposto atribui *threads* para as tarefas considerando a complexidade computacional da tarefa e a heterogeneidade na capacidade computacional das *threads*. Como trabalho futuro, pretende-se implementar esse algoritmo em um motor de execução da plataforma Guaraná para avaliar o ganho de desempenho com distintos casos de uso.

## REFERÊNCIAS


[1] A. A. Abraham, G. M. King, D. V. Rosa, and D. W. Schmidt. Runtime capacity planning in a simultaneous multithreading (smt) environment, Aug. 16 2016. US Patent 9,417,927.

[2] A. Al-maamari and F. A. Omara. Task scheduling using pso algorithm in cloud computing environments. *International Journal of Grid and Distributed Computing*, 8(5):245–256, 2015.

[3] C. Alexander, S. Ishikawa, and M. Silvertein. *A pattern language: towns, buildings, construction.* Oxford University Press, 1977.

[4] M. R. AlRashidi and M. E. El-Hawary. A survey of particle swarm optimization applications in electric power systems. *IEEE Transactions on Evolutionary Computation*, 13(4):913–918, 2009.

[5] D. Bratton and J. Kennedy. Defining a standard for particle swarm optimization. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 120–127. IEEE, 2007.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[7] P. Dietel. *Java how to program.* PHI, 2009.

[8] D. Dossot, J. D'Emic, and V. Romero. *Mule in action.* Manning, 2014.

[9] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE, 1995.

[10] A. P. Engelbrecht. *Computational intelligence: an introduction.* John Wiley & Sons, 2007.

[11] M. Fisher, J. Partner, M. Bogoevice, and I. Fuld. *Spring integration in action.* Manning Publications Co., 2012.

[12] M. Fisher, M. Bogoevici, et al. *Spring Integration Reference Manual, 4.3.7*, 2017.

[13] R. Z. Frantz, R. Corchuelo, and F. Roos-Frantz. On the design of a maintainable software development kit to implement integration solutions. *Journal of Systems and Software*, 111:89–104, 2016.

[14] Y. Fukuyama and Y. Nakanishi. A particle swarm optimization for reactive power and voltage control considering voltage stability. In *Proc. 11th IEEE Int. Conf. Intell. Syst. Appl. Power Syst*, pages 117–121, 1999.

[15] B. Hohpe, Gregor; Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions.* Addison-Wesley Professional, 2004.

[16] G. Hohpe. Your coffee shop doesn't use two-phase commit [asynchronous messaging architecture]. *IEEE software*, 22(2):64–66, 2005.

[17] J. Ibsen, Claus & Anstey. *Camel in action.* Manning Publications Co., 2010.

[18] K. Indrasiri. *Introduction to WSO2 ESB.* Springer, 2016.

[19] A. Lazinica. *Particle swarm optimization.* InTech Kirchengasse, 2009.

[20] C. O. Ourique, E. C. Biscaia, and J. C. Pinto. The use of particle swarm optimization for dynamical analysis in chemical processes. *Computers & Chemical Engineering*, 26(12):1783–1793, 2002.

[21] M. A. Rodriguez and R. Buyya. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. *IEEE Transactions on Cloud Computing*, 2(2):222–235, 2014.

[22] J. Russell and R. Cohn. *Fuse Esb.* Book on Demand, 2012. ISBN 9785510817041.

[23] T. Sousa, A. Silva, and A. Neves. Particle swarm based data mining algorithms for classification tasks. *Parallel Computing*, 30(5):767–783, 2004.

[24] A. Tanenbaum. *Modern operating systems.* Pearson Education, Inc., 2009.

[25] F. Van Den Bergh and A. P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information sciences*, 176(8):937–971, 2006.