Algoritmo Baseado na Meta-heurística Particle Swarm Optimization para Configuração de Pools de Threads em Motores de Execução de Plataformas de Integração

Maira S. Brigo, Rafael Z. Frantz, Daniela L. Freire

Universidade Regional do Noroeste do Estado do Rio Grande do Sul
Departamento de Ciências Exatas e Engenharias
Rua Lulu Ilgenfritz, 480 – 98700-000 – Ijuí/RS – Brasil

mairа.brigo@sou.unijui.edu.br, {rzfrantz,dsellaro}@unijui.edu.br

Abstract. *This paper reports on the experience of using an optimization algorithm to find the optimal configuration of thread pools to run integration processes that receive a large volume of data. Thread pools are computational resources in the runtime system of integration platforms. Execution models based on a single pool of threads have performance problems when used in contexts in which there is a large volume of data. An execution model based on multiple local pools of threads can avoid performance problems, as long as these pools are configured with the ideal number of threads. However, finding that ideal number of threads is not a trivial task, as there is no automatic way to perform this calculation. In this paper we explore the use of the Particle Swarm Optimisation metaheuristic to find the ideal number of threads for each local pool.*

Resumo. *Este artigo relata a experiência de usar um algoritmo de otimização para encontrar a configuração ideal de pools de threads para executar processos de integração que recebem um grande volume de dados. Threads são recursos computacionais no sistema de tempo de execução de plataformas de integração. Os modelos de execução baseados em um único pool de threads apresentam problemas de desempenho quando usados em contextos nos quais há um grande volume de dados. Um modelo de execução baseado em vários pools locais de threads pode evitar problemas de desempenho, desde que esses pools sejam configurados com o número ideal de threads. No entanto, encontrar esse número ideal de threads não é uma tarefa trivial, pois não há uma maneira automática de realizar esse cálculo. Neste artigo, exploramos o uso da meta-heurística Particle Swarm Optimization para encontrar o número ideal de threads para cada pool local.*

1. Introdução

Os processos de negócio das empresas estão em constante transformação, requerendo a atualização e/ou inclusão de novas aplicações para dar suporte à execução dos processos. O conjunto destas aplicações formam o ecossistema de *software* da empresa [Manikas 2016]. Geralmente, as aplicações foram desenvolvidas em distintas linguagens, são executadas em diferentes sistemas operacionais e possuem modelos de dados que geralmente, não é o mesmo, tornando o ecossistema de *software* heterogêneo. Esta heterogeneidade faz a troca de dados e o compartilhamento de funcionalidades das

aplicações ser um desafio nos setores de Tecnologia de Informação (TI) das empresas. A área de Integração de Aplicações Empresariais *(Enterprise Application Integration - EAI)* proporciona metodologias, técnicas e ferramentas para desenvolver processos de integração [Romero and Vernadat 2016]. Definimos processo de integração como um programa computacional que integra aplicações envolvidas em um processo de negócio.

Plataformas de integração são ferramentas que oferecem recursos para modelar, implementar e executar processos de integração [Freire et al. 2019a]. A modelagem costuma ser realizada utilizando uma linguagem de domínio específico (DSL) e o resultado é um modelo conceitual que descreve o *workflow* com as tarefas e as aplicações integradas, isso permite as aplicações compartilharem dados e funcionalidades, com baixo nível de acoplamento. A implementação é feita com um conjunto de APIs que permitem transformar o modelo conceitual em código executável. A execução do código é responsabilidade do motor de execução, onde as *threads* são os recursos computacionais que executam as tarefas dos processos de integração e estão organizadas de duas diferentes formas: *pool* global ou *pool* local. A estratégia global consiste em *threads* armazenadas e disponibilizadas em um único *pool* associado àquele processo de integração, enquanto a estratégia local consiste em utilizar diversos *pools* de *threads* associados diretamente às tarefas.

Na literatura há dois modelos que podem ser seguidos para implementação dos motores de execução; *process-based* e *task-based* [Blythe et al. 2005, Alkhanak et al. 2016, Boehm et al. 2011, Frantz et al. 2012]. No modelo *process-based*, uma mesma *thread* é responsável pela execução de todas as tarefas sobre uma mesma mensagem que percorre o processo de integração, e no modelo *task-based*, distintas *threads* podem ser usadas para executar tarefas que processam uma mesma mensagem ao longo do processo. Este artigo centra-se em motores de execução que adotam o modelo *task-based*. Neste modelo, quando o motor de execução possui um *pool* global e a taxa de chegada de mensagens é muito alta, ocorre priorização da execução das tarefas do início do processo de integração, prejudicando a execução das demais tarefas [Freire et al. 2019b]. Esta priorização degrada a execução dos processos de integração, pois o tempo de espera das tarefas finais por *threads* tende a ficar elevado. Quando o motor de execução possui *pools* locais, esta degradação é minimizada, porque cada tarefa possui um *pool* de *threads* dedicado a execução de suas instâncias e isso faz com que as mensagens sejam executadas com celeridade, melhorando seu tempo de processamento, desde que seja uma configuração ótima ou próxima de ótima.

Uma medida usual de desempenho da execução de processos de integração é o *makespan*, que é o tempo médio de processamento de uma mensagem desde sua entrada até sua saída do processo [Chirkin et al. 2017]. Um *makespan* baixo é indicativo de mais mensagens processadas por unidade de tempo. Enquanto um *makespan* elevado é indicativo de menos mensagens processadas por unidade de tempo. Geralmente os engenheiros de *software* aumentam o número de *threads* para aumentar o desempenho do motor de execução, especialmente em contextos de grande volume de dados. Porém, um grande número de *threads* aumenta a concorrência entre elas pelos recursos computacionais que compartilham, levando a degradação de desempenho na execução [Pusukuri et al. 2011]. Portanto, o desafio encontrado é dimensionar a quantidade ideal de *threads* em cada *pool* local, independente do número de *threads* disponíveis, para que essa configuração de *pools* locais possa melhorar o desempenho dos motores de execução. Este artigo explora

uma alternativa para obter a configuração ideal para *pools* de *threads* locais por meio de um algoritmo de otimização que minimiza o *makespan* do processamento de mensagens, melhorando o desempenho dos motores de execução. Foi realizado um experimento com o algoritmo baseado na meta-heurística *Particle Swarm Optimization* (PSO) em um processo de integração, onde se variam o número de *threads* distribuídos em *pools* de *threads* locais. As principais contribuições com este artigo são de que é possível encontrar uma configuração ideal para os *pools* locais de *threads*, é possível encontrá-la por meio do algoritmo e obtém um *makespan* otimizado dado a um número total de *threads* fixo. O experimento pode ser ampliado para outras taxas de entrada de mensagens, processos de integração, números de *threads* e números de configuração testados. O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta um embasamento teórico. A Seção 3 discute trabalhos relacionados. A Seção 4 formula o problema e apresenta a proposta. A Seção 5 apresenta o experimento realizado e analisa os resultados alcançados. A Seção 6 apresenta as conclusões do trabalho.

2. Fundamentação Teórica

Um processo de integração consiste em um *workflow* de estrutura *pipes-and-filters* [Alexander 1977], na qual os *pipes* são os canais e os *filters* são tarefas que processam os dados que são encapsulados no formato de mensagens. [Hohpe and Woolf 2003] documentaram um conjunto de padrões de integração que inspiram a concepção dessas tarefas proporcionadas pelas linguagens de domínio específico das plataformas de integração. Em um processo de integração, as tarefas estão dispostas em uma determinada ordem, de maneira que uma mensagem só pode ser processada por uma tarefa depois de já ter sido processada por todas as tarefas que a antecedem. Uma mensagem é completamente processada quando executada por cada uma das tarefas que compõem o *workflow* que ela percorre desde que entra no processo até quando ela sai.

O processo de integração é realizado pelo motor de execução. Os principais elementos de um motor de execução são: planificador, filas de instâncias, e *pools* de *threads*. O planificador gerencia a fila de instâncias, e as *threads*. Os agendamentos de cada uma das instâncias são feitos em uma fila de instâncias relacionada a sua tarefa e consequentemente ao seu *pool*, onde permanecem enquanto aguardam *threads* disponíveis para serem executadas. A fila de instâncias mantém o agendamento da execução das tarefas prontas para serem executadas, sendo que, uma tarefa é considerada apta, quando há mensagens aguardando em todas as suas entradas. Uma tarefa possui obrigatoriamente entradas e saídas. Para a tarefa ser executada, é necessário que haja *thread* disponível para sua execução. As *threads* estão recorrentemente verificando a sua respectiva fila de instâncias, na qual as primeiras instâncias anotadas na fila serão as primeiras a serem executadas. As *threads* do *pool*, quando estão disponíveis, buscam na sua fila as instâncias para executar. Chamamos de uma configuração de um *pool* local um vetor, no qual cada elemento corresponde ao número de *threads* de um *pool* local.

O *task-based* é um modelo que a literatura define como tendo uma granularidade mais baixa, e portanto, é um modelo que busca incrementar o processo paralelo de tarefas e o compartilhamento de *threads* entre tarefas de processos. A literatura também acrescenta que esse modelo costuma ser mais eficiente que o *process-based* [Blythe et al. 2005, Frantz et al. 2012]. A meta-heurística PSO foi utilizada já que segundo seu criador, cada partícula do enxame controla melhor sua posição no hiperespaço,

sendo uma meta-heurística eficaz em diversos tipos de problemas, além de ter um conceito simples e ser de fácil implementação [Eberhart and Kennedy 1995].

3. Trabalhos Relacionados

Na revisão de literatura, identificamos um conjunto de trabalhos que analisam e buscam reduzir o *makespan* utilizando técnicas de otimização. Nos trabalhos analisados, as propostas são algoritmos ou aperfeiçoamento de algoritmos tendo como base o PSO, ou outras meta-heurísticas. Em concordância, este artigo busca a minimização do *makespan*, mas através de um algoritmo baseado apenas no PSO, não são feitas comparações com outras meta-heurísticas. [Irman et al. 2019] propuseram um algoritmo para reduzir o *makespan* envolvido no tempo de produção de uma indústria na Indonésia. Os autores compararam a eficiência de seu algoritmo na solução do problema também com o uso dos métodos heurísticos *Camppbell Dudek Smith* (CDS) e PSO. Neste trabalho eles demonstraram que o *makespan* utilizando o CDS foi o mais alto, o com o PSO foi o de desempenho médio, e com o algoritmo proposto no artigo se obteve o menor *makespan*. O trabalho de [Zarrouk and Jemai 2018] teve o intuito de minimizar a produção e carga de trabalho total. Para isso, foram avaliados e comparados o desempenho de duas variantes de PSO com diferentes representações de partículas, uma delas foi o PSO com esquema de codificação *Job-Manchine* (PSO-JMS) e a outra o PSO com o esquema de codificação *Only-Manchine* (PSO-OMS). Como resultado, o PSO-OMS ofereceu o melhor desempenho.

[Bozorgnezhad et al. 2019] propuseram minimizar o *makespan* para solucionar dois problemas simultâneos: a) encontrar a melhor atribuição do trabalhador; e, b) resolver o problema de agendamento correspondente. Para isso, foram utilizadas duas abordagens de aproximação baseadas no PSO: o PSO e o SPSO. Os resultados mostraram que os dois algoritmos minimizaram eficientemente, mas o SPSO teve soluções melhores, principalmente para problemas maiores. [Teekeng et al. 2016] tinham como função objetivo minimizar o *makespan* para resolver problemas de agendamento. Para isso, propuseram um novo algoritmo, denominado EPSO, que foi baseado no PSO. No EPSO foram incluídos dois conjuntos de recursos, com o intuito de expandir o espaço de solução e evitar a convergência prematura. [Zhang and Wu 2014] investigaram um problema de agendamento de permutas de produção, com o objetivo de minimizar o tempo total de fluxo. Para isso, propõem uma meta-heurística híbrida baseada no PSO. Os resultados mostram que a proposta pode competir com outras meta-heurísticas poderosas da literatura.

[Khosiawan et al. 2018] tinham objetivo de minimizar o *makespan* e o consumo total de bateria em um cronograma de sistema de agendamento. Fizeram a comparação da otimização diferencial do enxame de partículas fundidas na evolução com a evolução diferencial e a otimização do enxame de partículas. A primeira mostrou resultados eficazes. Também para um problema de agendamento de tarefas, [Sarathambekai and Umamaheswari 2017] apresentaram este em sistemas distribuídos, e utilizaram a otimização de enxame de partículas discretas (DPSO) com várias topologias de vizinhança. Esta é uma meta-heurística recente para algoritmo populacional.

Estes trabalhos relacionados buscam otimizar o *makespan* em processos de integração, mas nenhum deles faz esta otimização utilizando o modelo de execução *task-based* e seguindo a estratégia de *pools* locais, sendo este o diferencial do nosso trabalho.

4. Formulação do problema

Um processo de integração é composto por um fluxo de trabalho que pode ser representado por um Grafo Acíclico Direcionado, representado por $DAG = (Z, H)$, onde Z é o conjunto de *n* tarefas e H são as arestas que representam a relação de dependência entre as tarefas, ou seja, as tarefas dependem umas das outras e se relacionam entre si. Assumimos que um conjunto de recursos utilizados para o processamento é representado por *r* e o seu armazenamento é indicado por *Dr*, sendo que todos estes recursos se localizam em *P* locais diferentes. Em um DAG existem nós que são conectados por arestas. Em nosso contexto, os nós são tarefas, as arestas são canais de comunicação em que são organizadas mensagens, e as instâncias são ocorrências concretas das tarefas que são executadas por *threads*. As *threads* estão disponíveis em *pools* locais, sendo que cada tarefa tem um *pool* de *threads*. Um caminho é um conjunto sequencial de tarefas que estão organizadas em um fluxo de trabalho, e o caminho mais longo no processo de integração, se chama *caminho crítico*. Em cada nó há um tempo de processamento associado, e dos nós pode haver um conjunto de instâncias que permitem uma execução paralela do nó.

Se busca encontrar o número correto de instâncias para cada nó, para que o tempo de processamento seja o menor possível. Para resolver o problema da degradação, consideramos que existe uma restrição que limita o número total de *threads* a serem distribuídas nos *pools*, e que é necessário ter pelo menos uma *thread* para que a tarefa seja executada. O tempo de processamento de uma tarefa é a soma do tempo de execução e do tempo de espera na fila de instâncias. O *makespan* é obtido realizando a soma de tempo de execução de cada tarefa e a soma do tempo de transferência de dados de uma tarefa para sua sucessora [Bilgaiyan et al. 2014]. Assim, o *makespan* de todo o fluxo do processo T_t (r_i) é o somatório do tempo incorrido da execução de cada tarefa T_i para o r_i de recursos e o somatório dos tempos de transmissão (T_m) entre um conjunto de tarefas que sejam dependentes entre si. O tempo total de processamento (TT_p) é calculado pela Equação 1:

$$TT_p = \sum T_i + \sum T_m \tag{1}$$

Queremos que o tempo de processamento que uma mensagem leva para atravessar todo o grafo seja o mínimo possível, então, a função objetivo é: *Minimizar o makespan do processamento de mensagens pelo caminho crítico de um processo de integração, encontrando a melhor configuração dos pools de threads locais, utilizando uma quantidade pré-definida de threads para ser distribuída entre os pools locais*. Para isso, podemos utilizar a função objetivo em uma meta-heurística, que é uma técnica que tem o intuito de buscar a otimização. Se baseia em uma população, em que a função objetivo é testada para os indivíduos dessa população inicial, e são feitas operações para melhorar a convergência do resultado, buscando minimizar a função objetivo. Como proposta, utilizamos a técnica meta-heurística PSO, que tem motivação no comportamento social [Shi and Eberhart 1999]. Inicialmente é gerada uma população com possíveis configurações de *threads* para os *pools* locais, que é a população inicial. Então, é calculado o tempo total médio de processamento das mensagens no fluxo de trabalho, o *makespan*, para cada configuração. Sempre que o *makespan* atual for menor que o anterior, é atualizado o *makespan* mínimo acompanhado de sua configuração. Ao fim, é encontrada a melhor configuração, que é a que possui o *makespan* mínimo de todo o processo.

5. Experimento

Esta seção descreve o experimento realizado com uma adaptação do PSO, para encontrar a configuração ideal dos *pools* de *threads* local para *makespan* mínimo em um processo de integração. O protocolo aplicado para conduzir o estudo experimental tem bases nos trabalhos de [Jedlitschka and Pfahl 2005, Perry et al. 2000] e [Wohlin et al. 2012].

5.1. Questão de pesquisa e hipótese

Foi selecionada uma questão de pesquisa para ser respondida a partir deste estudo experimental: *É possível encontrar uma configuração ideal de pools de threads para que motores de execução de plataformas de integração possam executar um processo de integração no menor makespan possível no contexto de grandes volumes de dados?*

Para esta questão de pesquisa, foi fornecida uma hipótese, que deve ser confirmada ou refutada pelo experimento: *Podemos encontrar a configuração ideal dos pools de threads locais por meio de um algoritmo baseado na técnica de otimização PSO, cujo objetivo é minimizar o makespan da execução no processo de integração*.

5.2. Ambiente do experimento

O experimento foi realizado em uma máquina equipada com 32 processadores *Intel Xeon CPU* E5-4610 1,80 *GHz*, 32 *GB* de *RAM* e sistema operacional *Microsoft Windows 10 Education* 64 *bits*. O *software Matlab* [Leonard and Levine 1995], versão R2018, foi usado executar os algoritmos. O código fonte do algoritmo de otimização está disponível para *download*[1].

5.3. Variáveis

Nesta seção, descrevem-se as variáveis dependentes e independentes que consideramos em nosso experimento. As variáveis independentes são:

Número de soluções: a população de configurações iniciais dos *pools* de *threads*. O valor desta variável foi de 20 soluções.

Número de *threads*: número total de *threads* que são distribuídos nos *pools* locais. O valor desta variável foi de 100 *threads*.

Número de mensagens: carga de trabalho do processo de integração, em que o valor testado foi 1.000.000 mensagens.

A variável dependente é:

***Makespan*:** o tempo médio total de processamento que uma mensagem leva para ser processada por todas as tarefas que fazem parte do *caminho crítico* do processo de integração.

5.4. Execução e coleta de dados

Nesta seção, descreve-se o estudo de caso e a coleta de dados. O estudo de caso adotado é o processo de integração para processamento de pedidos proposto por [Hohpe and Woolf 2003], conforme a Figura 1. O processo de integração tem cinco aplicações: *Ordering System*, *Widget Inventory*, *Gadget Inventory*, *Invalid Items Log* e *Inventory System*. O *Ordering System* representa o aplicativo de origem entregando os

[1] `www.gca.unijui.edu.br/publication/data/eres-pso-sources.zip`

dados dos novos pedidos ao processo de integração. Os dados de um pedido compõem uma mensagem que flui pelo processo de integração. Cada mensagem com um novo pedido é dividida em distintas mensagens, cada uma contendo apenas um item. O destino da mensagem para a aplicação *Widget Inventory* ou para a aplicação *Gadget Inventory*, está contido no conteúdo da mesma. Mensagens com itens não pertencentes a nenhum desses inventários são roteadas para *Invalid Items Log*. O aplicativo *Inventory System* representa a aplicação de destino final, que responde registrando a disponibilidade dos itens.

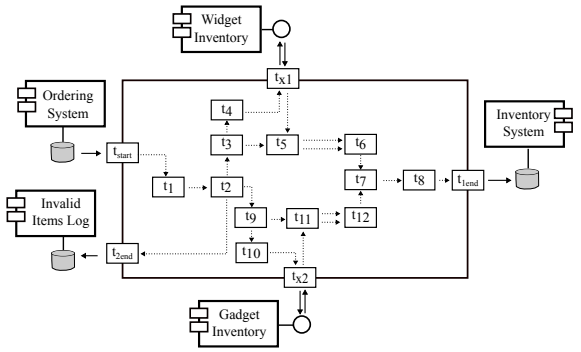

Figura 1. Modelo conceitual do processo de integração.

Este processo de integração possui três caminhos diferentes para mensagens que são representados no DAG da Figura 2.O *caminho crítico* é formado pelas tarefas t_{start}, t_1, t_2, t_3, t_4, t_{x1}, t_5, t_6, t_7, t_8, t_{1end}. As tarefas do *caminho crítico* consomem os seguintes tempos de processamento tomados do trabalho de [Haugg et al. 2019], em milissegundos: 2.005ms, 2.005ms, 3.005ms, 3.005ms, 2.005ms, 2.005ms, 4.553ms, 2.005ms, 4.553ms, 2.005ms, 2.005ms. A simulação da execução deste processo foi realizada na máquina virtual descrita anteriormente, com o *software Matlab*, utilizando o caminho crítico com um processamento de 1.000.000 de mensagens. Foram experimentadas de maneira aleatória pelo algoritmo, 20 configurações diferentes para os *pools* de *threads* locais, sendo um *pool* por tarefa, ou seja, 11 *pools*, com a restrição de usar no total 50 *threads*. A simulação resultou em 20 configurações, e cada uma delas gerou um *makespan*. Cada configuração e seu *makespan* foram coletados e armazenados para posterior análise.

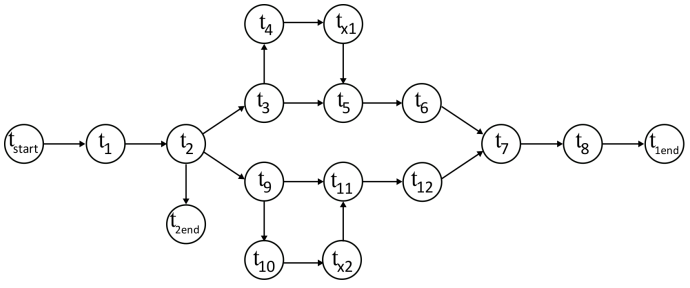

Figura 2. Grafo Acíclico Direcionado que representa o processo de integração.

5.5. Resultados

O menor *makespan* encontrado foi de 1002528,15ms com as configurações: [16,7,2,8,3,3,3,1,3,3,1] e [7,5,2,3,5,8,5,1,7,1,6]. A Figura 3 apresenta o gráfico dos *makespan* de todas as configurações de *threads* geradas pelo algoritmo. Neste estudo de

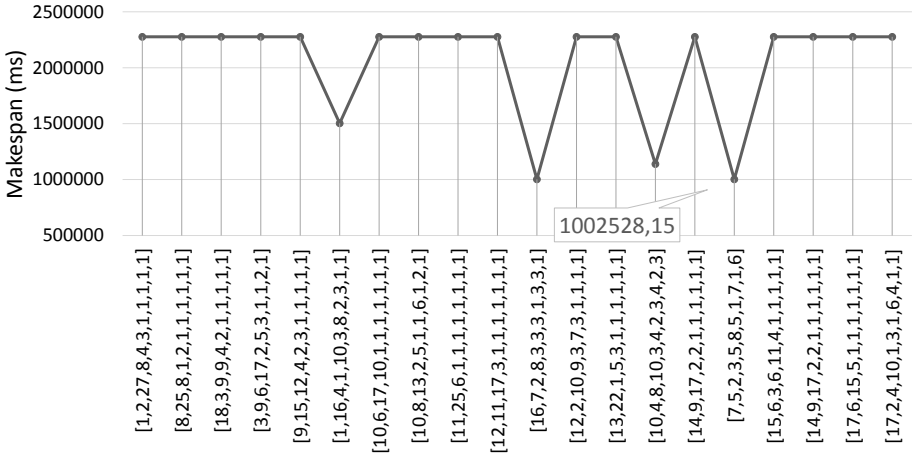

Figura 3. Makespan para diferentes configurações de pools de threads locais.

caso foi possível encontrar as configurações que obtiveram o menor tempo de processamento. Estas configurações contém o número ideal de *threads* nos *pools* locais do motor de execução, e podemos nos referir a elas como as configurações ideais, pois são as que encontraram o menor *makespan* no processo de integração.

5.6. Ameaças à validade

Instrumentação e fonte de ruído são possíveis ameaças. Para mitigá-las, antes da realização da simulação, definiu-se a questão de pesquisa, formulou-se a hipótese e definiram-se as variáveis independentes e dependentes. Depois, foram fornecidas informações sobre o ambiente de execução, ferramentas de suporte, execução e coleta de dados. Para minimizar a interferência no tempo de execução do algoritmo, toda a simulação foi realizada na mesma máquina, utilizando recursos mínimos e desconectada da *Internet* durante as execuções.

6. Conclusão

Neste artigo foi apresentada uma proposta para encontrar uma configuração ideal de *threads* para cada *pool* local, com o objetivo de reduzir o *makespan* em um processo de integração. Como o sistema de execução é o componente da plataforma responsável pela execução dos processos de integração, ele se torna o elemento mais importante quando o objetivo é o desempenho, pois ele está diretamente relacionado aos recursos computacionais. Propomos um algoritmo de otimização baseado na meta-heurística PSO, ele foi implementado e experimentado, encontramos múltiplas configurações dentre as quais uma configuração que minimizou o *makespan*, que denominamos de ideal no experimento do estudo de caso proposto. O experimento permitiu confirmar a hipótese de que era possível encontrar a configuração ideal dos *pools* de *threads* locais por meio de um algoritmo baseado em PSO, cuja função objetivo era minimizar o *makespan* do processo de integração. O algoritmo ainda, pode ser utilizado em diferentes contextos, com diferentes números de mensagens, outros processos de integração, números de *threads* e das configurações testadas. Como trabalho futuro, propomos de estender os resultados desta pesquisa, analisar o experimento estatisticamente e, utilizar o mesmo estudo de caso e protocolo de experimentação, utilizando outra meta-heurística. Também realizar uma comparação en-

tre os dois experimentos para posterior análise, para entender qual meta-heurística é mais adequada para otimizar o desempenho de processos de integração.

Agradecimentos

Este trabalho tem o apoio da Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul (FAPERGS) no contexto do Programa Pesquisador Gaúcho, com termo de outorga número 17/2551-0001206-2 e da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela bolsa de pós-graduação. Gostaríamos de agradecer ao colega Fernando Parahyba por seus comentários em versões anteriores deste artigo.

Referências

Alexander, C. (1977). *A pattern language: towns, buildings, construction*. Oxford University Press.

Alkhanak, E. N., Lee, S. P., Rezaei, R., and Parizi, R. M. (2016). Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. *Journal of Systems and Software*, 113:1–26.

Bilgaiyan, S., Sagnika, S., and Das, M. (2014). Workflow scheduling in cloud computing environment using cat swarm optimization. In *2014 IEEE International Advance Computing Conference (IACC)*, pages 680–685. IEEE.

Blythe, J., Jain, S., Deelman, E., Gil, Y., Vahi, K., Mandal, A., and Kennedy, K. (2005). Task scheduling strategies for workflow-based applications in grids. In *IEEE International Symposium on Cluster Computing and the Grid, 2005.*, volume 2, pages 759–767.

Boehm, M., Habich, D., Preissler, S., Lehner, W., and Wloka, U. (2011). Cost-based vectorization of instance-based integration processes. *Information Syst.*, 36(1):3–29.

Bozorgnezhad, F., Asadi-Gangraj, E., and Mahdi, M. (2019). A simultaneous worker assignment and scheduling problem for minimizing makespan in flexible flow shop via metaheuristic approaches.

Chirkin, A. M., Belloum, A. S., Kovalchuk, S. V., Makkes, M. X., Melnik, M. A., Visheratin, A. A., and Nasonov, D. A. (2017). Execution time estimation for workflow scheduling. *Future Generation Computer Systems*, 75:376–387.

Eberhart, R. and Kennedy, J. (1995). Particle swarm optimization. In *Proceedings of the IEEE intl. conference on neural networks*, volume 4, pages 1942–1948. Citeseer.

Frantz, R. Z., Corchuelo, R., and Molina-Jiménez, C. (2012). A proposal to detect errors in enterprise application integration solutions. *Journ. of Syst. and Soft.*, 85(3):480–497.

Freire, D. L., Frantz, R. Z., and Roos-Frantz, F. (2019a). Ranking enterprise application integration platforms from a performance perspective: An experience report. *Software: Practice and Experience*, 49(5):921–941.

Freire, D. L., Frantz, R. Z., Roos-Frantz, F., and Sawicki, S. (2019b). Survey on the runtime systems of enterprise application integration platforms focusing on performance. *Software: Practice and Experience*, 49(3):341–360.

Haugg, I. G., Frantz, R. Z., Roos-Frantz, F., Sawicki, S., and Zucolotto, B. (2019). Towards optimisation of the number of threads in the integration platform engines using simulation models based on queueing theory. *Rv. Br. de Cmp. Ap.*, 11(1):48–58.

Hohpe, G. and Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley.

Irman, A., Febianti, E., and Khasanah, U. (2019). Minimizing makespan on flow shop scheduling using campbel dudek and smith, particle swarm optimization, and proposed heuristic algorithm. In *IOP Conference Series: Materials Science and Engineering*, volume 673, page 012099. IOP Publishing.

Jedlitschka, A. and Pfahl, D. (2005). Reporting guidelines for controlled experiments in software engineering. In *Intl. Symposium on Empirical Software Eng.*, pages 95–104.

Khosiawan, Y., Khalfay, A., and Nielsen, I. (2018). Scheduling unmanned aerial vehicle and automated guided vehicle operations in an indoor manufacturing environment using differential evolution-fused particle swarm optimization. *International Journal of Advanced Robotic Systems*, 15(1):1729881417754145.

Leonard, N. E. and Levine, W. S. (1995). *Using MATLAB to analyze and design Control Systems*. Benjamin-Cummings Publishing Company.

Manikas, K. (2016). Revisiting software ecosystems research: A longitudinal literature study. *Journal of Systems and Software*, 117:84–103.

Perry, D. E., Porter, A. A., and Votta, L. G. (2000). Empirical studies of software engineering: A roadmap. In *Proc. of the Conf. on The Future of Soft. Eng.*, pages 345–355.

Pusukuri, K. K., Gupta, R., and Bhuyan, L. N. (2011). Thread reinforcer: Dynamically determining number of threads via os level monitoring. In *IEEE Iinternational Simposium on Workload Characterization (IISWC)*, pages 116–125.

Romero, D. and Vernadat, F. (2016). Enterprise information systems state of the art: Past, present and future trends. *Computers in Industry*, 79:3–13.

Sarathambekai, S. and Umamaheswari, K. (2017). Task scheduling in distributed systems using heap intelligent discrete particle swarm optimization. *Computational Intelligence*, 33(4):737–770.

Shi, Y. and Eberhart, R. C. (1999). Empirical study of particle swarm optimization. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1945–1950. IEEE.

Teekeng, W., Thammano, A., Unkaw, P., and Kiatwuthiamorn, J. (2016). A new algorithm for flexible job-shop scheduling problem based on particle swarm optimization. *Artificial Life and Robotics*, 21(1):18–23.

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., and Wessln, A. (2012). *Experimentation in Software Engineering*. Springer.

Zarrouk, R. and Jemai, A. (2018). Performance evaluation of particles coding in particle swarm optimization with self-adaptive parameters for flexible job shop scheduling problem. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 396–407. Springer.

Zhang, L. and Wu, J. (2014). A pso-based hybrid metaheuristic for permutation flowshop scheduling problems. *The Scientific World Journal*, 2014.