Evento: XXV Jornada de Pesquisa
ODS: 9 - Indústria, Inovação e Infra-estrutura

ESTUDO EXPERIMENTAL DE UM ALGORITMO INSPIRADO EM CAT SWARM OPTIMIZATION PARA MINIMIZAR O MAKESPAN EM UM SISTEMA VAREJISTA ON-LINE[1]

EXPERIMENTAL STUDY OF AN INSPIRED ALGORITHM IN CAT SWARM OPTIMIZATION TO MINIMIZE MAKESPAN IN AN ONLINE RETAIL SYSTEM

Maira Simoni Brigo[2], Fernando Parahyba[3], Rafael Zancan Frantz[4]

[1] Pesquisa desenvolvida no Programa de Pós-Graduação em Modelagem Matemática e Computacional, pertencente ao Grupo de Pesquisa em Computação Aplicada
[2] Aluna do Curso de Mestrado em Modelagem Matemática e Computacional, bolsista CAPES, maira.brigo@sou.unijui.edu.br
[3] Aluno do Curso de Doutorado em Modelagem Matemática e Computacional, bolsista CAPES, fernando.parahyba@sou.unijui.edu.br
[4] Professor Doutor do Programa de Pós-Graduação em Modelagem Matemática e Computacional, Orientador, rzfrantz@unijui.edu.br

Resumo: As aplicações dão suporte nas empresas aos processos de negócio e, geralmente, é necessário realizar a integração entre elas. Um processo de integração é o resultado de uma integração entre as aplicações e, é, geralmente, construído utilizando uma plataforma de integração. O motor de execução é quem realiza a execução do processo de integração e existem dois modelos teóricos de execução são eles: *task-based* e *process-based.* Os recursos computacionais usados para a execução são as *threads* e elas podem ser alocadas em duas estratégias de distribuição de *pool*: global e local. Nosso trabalho utiliza o modelo teórico *task-based* e a estratégia de *pool l*ocal. É importante melhorar o desempenho dos processos de integração e a métrica comumente utilizada para medir este desempenho é o tempo médio de processamento de mensagens, o *makespan*. O problema consiste em minimizar o *makespan*, possibilitando otimizar a performance, para isso, os *pools* locais precisam estar configurados com o número ideal de *threads* em cada um, pois uma má distribuição afeta a performance. Como não há uma maneira automática de se fazer isso, realizar este cálculo torna-se um desafio. O presente artigo busca explorar o uso da meta-heurística *Cat Swarm Optimization* para encontrar o número ideal de *threads* para cada *pool* local de um processo de integração. Este artigo descreve um experimento que usa um caso de estudo clássico de Integração de Aplicações Empresariais e encontra a configuração ótima para *pools* de *threads* locais na execução do processo de integração para um grande volume de dados.

Abstract: Applications support business processes in companies and it is generally necessary to integrate them. The integration process is the result of an integration between applications and is generally, built using an integration platform. The runtime system is the peace of software responsible for executing the integration process and in the literature, there are two theoretical models of execution, namely: task-based and process-based. The computational resources used for execution are threads and they can be allocated following two pool distribution strategies: global and local. Our research focuses on the task-based theoretical model and the local pool strategy. It is important to improve the performance of the integration processes and the metric commonly used to measure is the average message processing time, known as makespan. The problem consists in minimizing the makespan, making it possible to optimize the performance, for that, local pools need to be configured with the ideal number of threads in each one, because a bad distribution affects the performance. Since there is no automatic way to do this, a challenge is to perform this calculation. The paper seeks to explore the use of CSO meta-heuristics to find the ideal number of threads for

Evento: XXV Jornada de Pesquisa
ODS: 9 - Indústria, Inovação e Infra-estrutura

each local pool in an integration process. Our paper describes an experiment that uses a classic Enterprise Application Integration case study and finds the optimal configuration for pools of local threads in the execution of the integration process for a large volume of data.

Palavras-chave: Integração de Aplicações Empresariais, Processo de Integração, *Makespan*, Meta-heurística, *Cat Swarm Optimization.*

Keywords: Enterprise Application Integration, Integration Process, Makespan, Meta-heuristics, Cat Swarm Optimization.

1 INTRODUÇÃO

Os processos de negócio são atividades que uma empresa possui, que resulta em um serviço ou produto que ela oferece ao mercado. Existem diferentes aplicações de *software* que asseguram o funcionamento dos processos de negócio. É normal os processos sofrerem transformações e mudanças com o passar dos anos e isso implica também na melhoria das aplicações, ou com a compra/desenvolvimento de aplicações novas. O conjunto de todas as aplicações que fazem parte de uma empresa é chamado de ecossistema de *softwares* (MANIKAS, 2016). Geralmente com o passar do tempo o ecossistema de *softwares* tende a ficar heterogêneo, isso por que, as diferentes aplicações que fazem parte dele podem ter sido desenvolvidas em diferentes sistemas operacionais e com linguagens de programações distintas. Essa heterogeneidade faz com que as trocas de dados e funcionalidades das aplicações fiquem comprometidas, e encontrar uma forma para que haja colaboração entre as aplicações, é um desafio para o setor de Tecnologia da Informação (TI) das empresas.

Uma maneira de fazer com que as aplicações atuem em colaboração é por meio da área de estudo Integração de Aplicações Empresariais (EAI), que fornece métodos, técnicas e ferramentas com o fim de criar processos de integração capazes de realizar uma interação (ROMERO E VERDANAT, 2016). Um processo de integração consiste em tarefas dispostas em um fluxo de trabalho, por onde passam mensagens que precisam ser processadas, fazendo com que assim sejam integradas aplicações de um determinado processo de negócio. As mensagens são estruturas que contém dados, que podem ser roteadas e transformadas pelas tarefas no fluxo de trabalho, e uma mensagem só é considerada completamente processada quando já tiver percorrido todo o fluxo de tarefas do início do processo até o fim. As tarefas de um processo de integração obedecem uma ordem de que uma mensagem só pode ser executada na tarefa posterior, se já tiver passado pela sua antecessora. Para que seja possível a modelagem, implementação e execução dos processos de integração são necessárias as plataformas de integração (FREIRE et al., 2019). Nelas estão dispostas algumas ferramentas que geralmente são: uma linguagem de domínio especifico (DSL); uma Biblioteca de programação em PT (API); um motor de execução. Com uso da DSL é possível realizar a modelagem conceitual do processo de integração, e a API permite realizar a implementação que consiste em transformar o modelo conceitual em código executável. O motor é a ferramenta que realiza a execução do código gerado, ele utiliza para isso recursos computacionais, denominadas *threads*.

Para realizar a implementação dos motores de execução, observamos na literatura que podem ser seguidos dois modelos teóricos, que são: *process-based* e *task-based* (BLYTHE et al. 2005, ALKHANAK et al. 2016, BOEHM et al. 2011, FRANTZ et al. 2012). No modelo teórico

Evento: XXV Jornada de Pesquisa
ODS: 9 - Indústria, Inovação e Infra-estrutura

process-based, a execução do processo de integração é feita em nível de processo, ou seja, as *threads* realizam a execução das tarefas seguindo a ordem estabelecida entre elas, uma *thread* acompanha a execução de uma mensagem em todas as tarefas do processo até o fim. Enquanto no modelo *task-based* a execução é feita em nível de tarefa, isto é, as *threads* podem realizar a execução de qualquer tarefa do processo, isto possibilita que não existam *threads* ociosas na execução das tarefas e, desse modo, diferentes *threads* podem executar a mesma mensagem em tarefas distintas. Em ambos os modelos teóricos de execução as *threads* podem estar dispostas em dois diferentes tipos de *pool*: global e local. A diferença é que o *pool* global disponibiliza todas as *threads* em um único *pool*, enquanto no *pool* local elas são disponibilizadas de forma a ser um *pool* para cada tarefa do processo de integração. Neste artigo, será abordada a estratégia de *pools* locais e o modelo teórico de execução *task-based*.

Para avaliar o desempenho do motor de execução em processos de integração, é possível calcular o tempo médio de processamento de uma mensagem durante o processo, esse cálculo é definido como o *makespan* de um processo de integração (CHIRKIN et al. 2017). A relação do *makespan* com a performance dos processos de integração é de que quanto maior for o *makespan*, menor será a quantidade de mensagens processados em um determinado tempo, gerando dessa forma, degradação da performance. Em contrapartida, quando o *makespan* de um processo de integração tende a ser menor, podemos dizer que mais mensagens serão processadas em um determinado tempo, gerando assim otimização da performance. Levando em consideração que não é possível ter um número ilimitado de *threads* e ainda que ter *threads* ociosas pode levar a uma degradação da performance do motor de integração (FREIRE, 2020), existe um desafio que é dimensionar o número de *threads* por *pool* local para um processo de integração. Uma configuração equivocada pode ocasionar em baixo desempenho, portanto, é preciso encontrar uma configuração que resulte no menor *makespan* dado um número total de *threads* disponíveis a serem utilizadas, a fim de aprimorar assim o desempenho do motor de execução.

A área de estudos conhecida como *Search-based Software Engineering* (SBSE), aplica modelos matemáticos e técnicas de otimização em problemas relacionados à Engenharia de Software, sendo que Freitas et al. (2009) obtiveram resultados positivos na resolução de alguns problemas, e Harman, Mansouri e Zhang (2012) revisaram esta mesma área, e evidenciaram um grande interesse por pesquisadores na utilização de técnicas de otimização aplicadas a Engenharia de Software. Desta forma, como é necessário distribuir e testar diferentes configurações de *threads,* este artigo propõe que por meio de um algoritmo de otimização, seja possível encontrar a configuração ideal de *threads* para *pools* locais, que resulta no menor *makespan* para o processo de integração tendo como referência uma quantidade total de *threads* disponíveis, e assim melhorar a performance do motor de execução. Para isso, foi realizado um experimento com um algoritmo baseado na meta-heurística *Cat Swarm Optimization* (CSO) em um estudo clássico da literatura denominado processamento de pedidos que pode ser comparado analogamente com um sistema de varejo *on-line*. Este processo de integração foi proposto por Hohpe e Woolf (2003), e com o desenvolvimento do experimento, encontramos a melhor configuração que resultou no menor *makespan*, e elencamos as principais contribuições a partir do experimento e das análises. Esse estudo pode ser ampliado e novos resultados poderão ser encontrados para o mesmo experimento e poderá ser mudado o número total da carga de mensagens, o número de *threads*, o número de configurações testadas e o número de gatos.

O restante deste artigo está organizado da seguinte forma: a Seção 2 apresenta um conjunto

Evento: XXV Jornada de Pesquisa
ODS: 9 - Indústria, Inovação e Infra-estrutura

de trabalhados que foram analisados por terem relação com o objetivo do seguinte trabalho. A Seção 3 é a descrição da formulação do problema e da nossa proposta. A Seção 4 descreve o experimento realizado, a questão de pesquisa e a hipótese, as ferramentas, as variáveis, a execução e a coleta de dados, os resultados alcançados e as ameaças à validade. A Seção 5 apresenta as conclusões deste trabalho.

2 TRABALHOS RELACIONADOS

Para realizar uma revisão de literatura e encontrar trabalhos que tivessem relação com o tema, foram identificados um conjunto de trabalhos que por meio de experimentos encontram o *makespan* mínimo utilizando técnicas de otimização como meta-heurísticas em diferentes tipos de processos. Nos trabalhos relacionados encontrados, foram analisados e comparados algoritmos, em parte dos trabalhos foram realizadas melhoras nos algoritmos já existentes, e em outros, foram propostos algoritmos novos, sendo que em todos os trabalhos consta como parte ou como a única meta-heurística proposta, o CSO. Na mesma linha, este artigo busca a minimização do *makespan* por meio de um algoritmo baseado em CSO, não é intenção deste artigo realizar comparação entre técnicas de otimização.

O trabalho de Du, Wang e Lei (2019) fizeram uma tentativa de minimização do *makespan* e do custo, e de equilíbrio da taxa de carga do agendamento de processos de recursos de fabricação em nuvens. Os autores criaram um modelo de programação multiobjetivo com o intuito de atingir os objetivos citados. Em seguida, foram combinados o algoritmo da otimização por enxame de gatos (CSO) e o algoritmo de vagalume (FA) em um único algoritmo de programação multiobjetivo híbrido, para que fosse verificado através de uma simulação *CloudSim*. Como resultado dessa simulação, obtiveram um algoritmo gera um plano de programação ideal em pouco tempo. Sendo assim, tiveram como resultado também, que a aplicação de algoritmos de inteligência de enxame em problemas de agendamento é promissora. Gabi, Ismail e Dankolo (2019) trabalharam em uma linha semelhante, pois buscaram minimizar o tempo de produção para o agendamento eficiente de tarefas de um *datacenter* em nuvem. Para isso, introduziram uma técnica convencional de programação de tarefas baseada em CSO, incorporando uma equação *Linear Descending Inertia Weight* (LDIW) na busca local do CSO. Essa incorporação levou a uma melhor velocidade de convergência e diminuiu o tempo de execução. Este experimento foi implementado pelo *simulator CloudSim* com cinco máquinas virtuais heterogêneas, e encontraram uma solução ideal para o problema .

Maurya e Tripathi (2018) relatam que há um problema de agendamento de tarefas em ambientes de computação em nuvem para aplicativos *Bag-of-task* (BoT). Este problema consiste em que os aplicativos podem ter inúmeras tarefas, e isso pode aumentar os custos da execução, caso o agendamento seja realizado de forma inadequada. Este problema foi resolvido através de algoritmos de agendamento, e os autores testaram dois algoritmos heurísticos e dois algoritmos meta-heurísticos para resolver esse problema. Os algoritmos heurísticos são o IC-PCP e o SCS e os algoritmos meta-heurísticos são o PSO e o CSO. Todos os algoritmos citados tiveram o intuito de minimizar o custo de fabricação, já colocado pelos autores como um problema, e ainda atender a restrições de prazo observadas. Foram utilizadas três categorias de BoT para que fosse possível serem comparados os algoritmos a partir de seus desempenhos. Por fim, o resultado dos experimentos realizados foi de que o algoritmo CSO tem melhor desempenho do que os demais.

Semlali, Riffi e Chebihi (2018) apresentaram uma adaptação de um algoritmo meta-

heurístico, que chamaram de otimização de enxame de galinha com recozimento simulado (CSOSA) para resolver o problema de agendamento de oficina (JSSP). O objetivo dos autores foi otimizar a produção, envolvendo este novo algoritmo. O desempenho do algoritmo proposto foi aprimorado a partir de uma hibridação do algoritmo da otimização por enxame de gatos (CSO) com o algoritmo do recozimento simulado (SA). A obtenção dos resultados empíricos foram por meio da aplicação do novo algoritmo com instâncias da *OR Library*. Nos resultados computacionais concluíram que a eficácia do CSOSA em termos de qualidade da solução e tempo de execução, em comparação com outras meta-heurísticas existentes na literatura. Grupta, Singh e Anand (2018) relataram que o agendamento de tarefas na computação em nuvem é um aspecto muito importante. Dessa forma, o seu objetivo foi de minimizar custo e o tempo de execução para atingir um mapeamento de custo otimizado, a partir de um algoritmo. Para isso, estudaram que existem váriados algoritmos que podiam ser executados para o agendamento de tarefas na computação em nuvem, com base em enxames inteligentes. Os autores escolheram dois deles, que são o *Particle Swarm Optimization* (PSO) e o *Cat Swarm Optimization* (CSO). Foi proposto um CSO mesclado (MCSO), onde foram apresentadas combinações dos méritos do CSO e do PSO com o intuito de obterem melhores resultados. O algoritmo MCSO foi implementado no simulator *CloudSim* e o experimento obteve bons resultados, concluindo que o algoritmo MCSO tem um desempenho melhor em comparação com os algoritmos PSO e CSO, em termos de tempo de execução e convergência.

3 FORMULAÇÃO DO PROBLEMA E PROPOSTA

Um processo de integração geralmente é comparado a um fluxo de trabalho. Conforme a necessidade, este fluxo pode ser representado por um Gráfico Acíclico Direcionado, que pode ser representado por $DAG = (Z,H)$, onde Z é um conjunto de n tarefas e H são arestas que representam uma relação de dependência entre as tarefas. Em um DAG é possível assumir também que um conjunto de recursos que serão utilizados no processo é representado por r e o armazenamento destes recursos é representado por Dr, sendo que os diferentes locais em que esses recursos podem ser localizados é chamado de P. No DAG existem nós que são conectados por arestas. Para nós, os nós são as tarefas do processo de integração, as arestas são canais de comunicação onde as mensagens são transmitidas, e podem chamar as instâncias e ocorrências concretas das tarefas que são executadas pelas *threads*.

Há ainda o caminho, que pode ser caracterizado como um conjunto de sequências de tarefas organizadas no fluxo de trabalho, sendo que o caminho mais longo em um processo de integração, é sempre chamado de caminho crítico. Ocorre ainda que existem um tempo de processamento associado a cada nó, e de cada nó, pode haver um conjunto de instâncias que permitem execuções paralelas deste nó. Sendo assim, buscamos encontrar o número correto de instâncias para cada nó, para que o tempo de processamento do fluxo de trabalho seja o minimizado. Para que o objetivo seja concretizado, ainda é necessário levar em consideração que existe uma restrição no número de *threads*. Estas, são limitadas, e devem ser distribuídas nos pools locais obedecendo ao limite e ainda é preciso enfatizar que cada *pool* local precisa necessariamente ter pelo menos uma *thread* para que exista a execução da tarefa. O tempo de processamento de uma tarefa é igual a soma do tempo de execução propriamente dita e do tempo de espera na fila de instâncias.

O *makespan*, que é o tempo médio de processamento, é obtido ao realizar a soma do tempo de execução de cada tarefa processada e a soma do tempo de transferência dos dados de uma tarefa para sua sucessora (BILGAIYAN et al., 2014). Sendo assim, para conhecermos o *makespan*

de toda a quantidade de mensagens do processo podemos dizer que o tempo incorrido da execução de cada uma das tarefas para o conjunto de recursos de cada uma delas, e o somatório dos tempos de transmissão, entre um conjunto de tarefas que sejam dependentes entre si. Por fim, o tempo total de processamento (*TTp*) pode ser calculado pela equação:

$$TT_p = \sum T_i + \sum T_m$$

O objetivo é que o *TTp* seja o mínimo, portanto 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.

A função objetivo pode ser utilizada em uma meta-heurística, para que o problema seja resolvido. A meta-heurística é uma técnica que busca a otimização e tem base em uma população inicial, é testada a função objetivo para cada indivíduo da população e realiza operações para melhorar a convergência do resultado para minimizar a função objetivo. Nossa proposta é na utilização da técnica *Cat Swarm Optimization* (CSO), na qual se baseia no comportamento dos gatos. Se observarmos um conjunto de gatos, conseguiremos perceber que eles possuem um padrão determinado e que todos internamente tendem a funcionar de dois modos: modo de busca e modo de rastreamento. Os gatos que estão em modo de busca, permanecem em repouso fazendo inúmeras análises sobre o ambiente e/ou presa e, enquanto achamos que estão dormindo, estão na verdade com memória e senso de dimensão e de posicionamento ativos, já os gatos que estão em modo de rastreamento podem atacar a qualquer momento com movimentos rápidos, por mais que pareçam estar avançando lentamente. No processo, os gatos podem mudar de modo, e o ciclo só termina quando alguma condição que foi imposta, seja de mudar de posição ou atacar uma presa, for satisfeita.

A estruturação da proposta está em inicialmente gerar qual o número de gatos que farão parte do experimento e em seguida é preciso definir as demais variáveis independentes, que são o número de configurações que queremos, o número de *threads* utilizadas e o número de mensagens processadas. O algoritmo faz um ciclo de calcular o *makespan* para cada configuração de *pools* de *threads*, e, cada vez que esse *makespan* for menor que o anterior, é atualizado o *makespan* mínimo, e a melhor configuração dos *pools* de *threads*. A cada novo ciclo, é definido o modo dos gatos novamente, com base em parâmetros do algoritmo. Ao fim, é definido qual a configuração que possui o menor makespan para o processo de integração, desta maneira é resolvida a função objetivo minimizando o *makespan*.

4 EXPERIMENTO

Nesta seção, consta a descrição do experimento realizado por meio de uma adaptação do algoritmo do CSO, a fim de encontrar a configuração ideal dos *pools* locais de *threads* para obter um *makespan* mínimo em um processo de integração. O protocolo aplicado pra conduzir este estudo experimental tem suas bases nos trabalhos de Jedlitschka e Pfahl (2005), Perry et al. (2000) e Wohlin et al. (2012).

4.1 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* locais de *threads* para que, os motores de execução de plataformas de integração possam executar um processo de integração no menor *makespan* possível, no contexto de grandes volumes de dados?

Foi selecionada para esta questão de pesquisa, uma hipótese, que será confirmada ou refutada pelo experimento desenvolvido: Podemos encontrar a configuração ideal dos *pools* de *threads* locais por meio de um algoritmo baseado na meta-heurística CSO, cuja função objetivo é minimizar o *makespan* da execução no processo de integração.

4.2 Ambiente de 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*. Para executar os algoritmos, foi utilizado o *software Matlab* (Leonard e Levine, 1995), versão R2018. O código fonte do algoritmo de otimização está disponível para *download* através do link: *www.gca.unijui.edu.br/publication/data/sc-cso-sources.zip*.

4.3 Variáveis

Aqui descrevemos 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 30 soluções.
- **Número de *threads*:** o número total de *threads* que são distribuídos nos *pools* locais. O valor desta variável foi de 50 *threads*.
- **Número de mensagens:** a carga de trabalho do processo de integração. O valor testado foi de 1.000.000 mensagens.
- **Número de gatos:** o número de gatos utilizados no experimento. O valor utilizado foi de 25 gatos.

A variável dependente é:

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

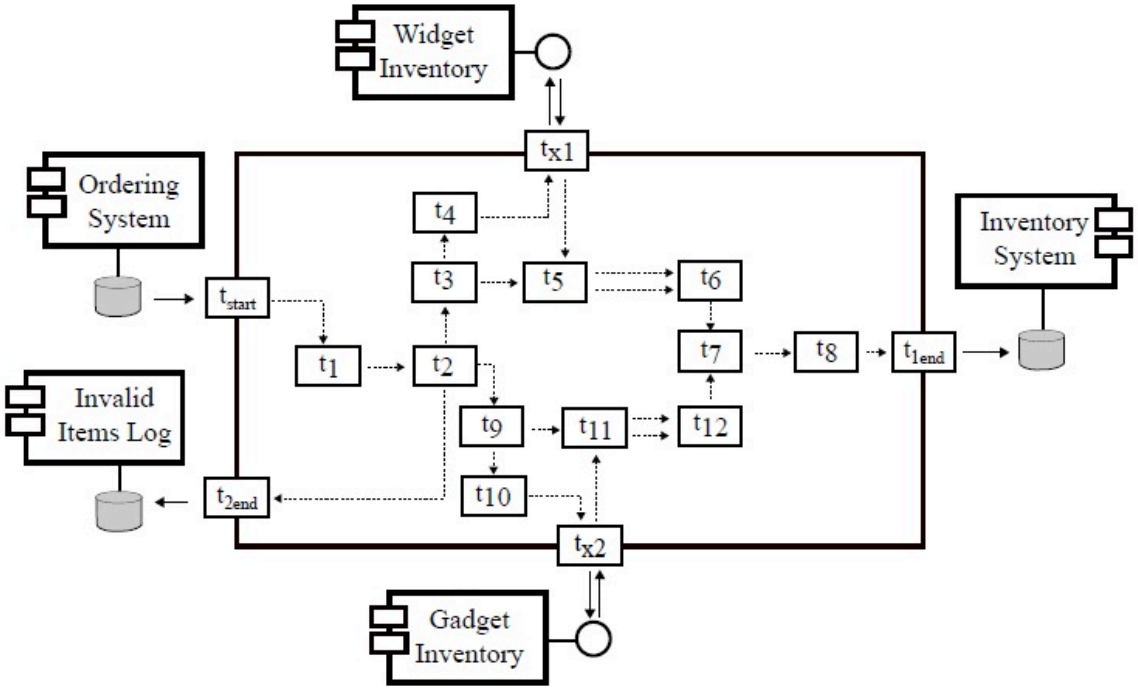
4.4 Descrição do caso de estudo e da execução e coleta dos dados

Nesta seção, descrevemos o estudo de caso utilizado como processo de integração e como ocorreu a execução e a coleta dos dados. Como estudo de caso, optamos pelo processo de integração do processamento de pedidos que pode representar um sistema varejista *on-line*. Este processo de integração foi proposto por Hohpe e Woolf (2003), como mostra a Figura 1. O processamento de pedidos é composto por cinco aplicações: *Ordering System, Widget Inventory, Gadget Inventory, Invalid Items Log* e *Inventory System.* A primeira aplicação, que é a *Ordering*

Evento: XXV Jornada de Pesquisa
ODS: 9 - Indústria, Inovação e Infra-estrutura

System, é o aplicativo de origem que basicamente entrega os dados dos novos pedidos que chegam no processo de integração, sendo que esses dados compõem uma mensagem, e é essa mensagem flui pelo processo. Desta maneira, após criadas as mensagens com dados dos pedidos, cada mensagem é dividida em diferentes mensagens novas, cada uma contendo apenas um item do pedido. Neste momento, a mensagem precisa seguir destino para a aplicação *Widget Inventory* ou para a aplicação *Gadget Inventory*, e a decisão é tomada a partir do conteúdo da mensagem, que diz para qual das aplicações ela irá. Caso a mensagem contenha itens que não pertencem a nenhum desses dois inventários, então elas são roteadas para a aplicação *Invalid Items Log*. Por fim, a aplicação *Inventory System* é o destino final das mensagens, sendo que essa aplicação é a que dá as respostas registrando a disponibilidade dos itens.

Figura 1: Modelo conceitual do processo de integração para processamento de pedidos.

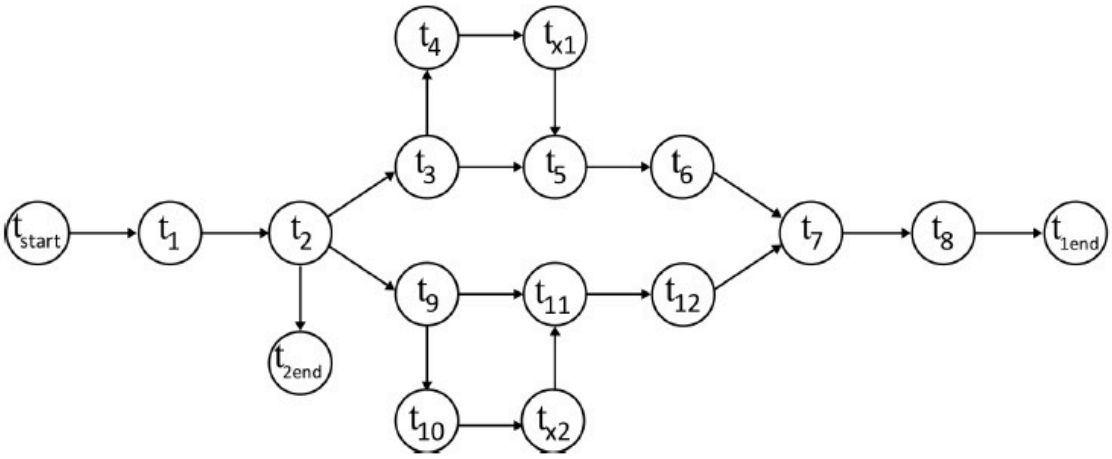


Fonte: Adaptado de FREIRE (2020).

O processo de integração em questão possui três caminhos diferentes em que as mensagens podem seguir, e estes caminhos estão representados no *DAG* da figura 2. O caminho crítico é formado pelas tarefas *tstart*, *t1*, *t2*, *t3*, *t4*, *tx1*, *t5*, *t6*, *t7*, *t8*, *tend*. As tarefas deste caminho possuem os seguintes tempos de processamento: 2.005 ms, 2.005 ms, 3.005 ms, 3.005 ms, 2.005 ms, 2.005 ms, 4.553 ms, 2.005 ms, 4.553 ms, 2..005 ms, 2.005 ms. Estes tempos estão em milissegundos e foram retirados do trabalho de Haugg et al. (2019). A simulação da execução do processo de integração foi realizada utilizando o caminho crítico, com um processamento de 1.000.000 de mensagens, sendo experimentadas de maneira aleatória pelo algoritmo baseado no CSO, 30 configurações diferentes de *pools* de *threads* locais, sendo 11 *pools*, pois é um *pool* para cada tarefa, mantendo a restrição de utilizar um total de 50 *threads*, e com 25 gatos. Cada uma das configurações resultantes gerou um *makespan*, e estes foram coletados e armazenados para análise.

Evento: XXV Jornada de Pesquisa
ODS: 9 - Indústria, Inovação e Infra-estrutura

Figura 2: Gráfico Acíclico Direcionado que representa o processo de integração.

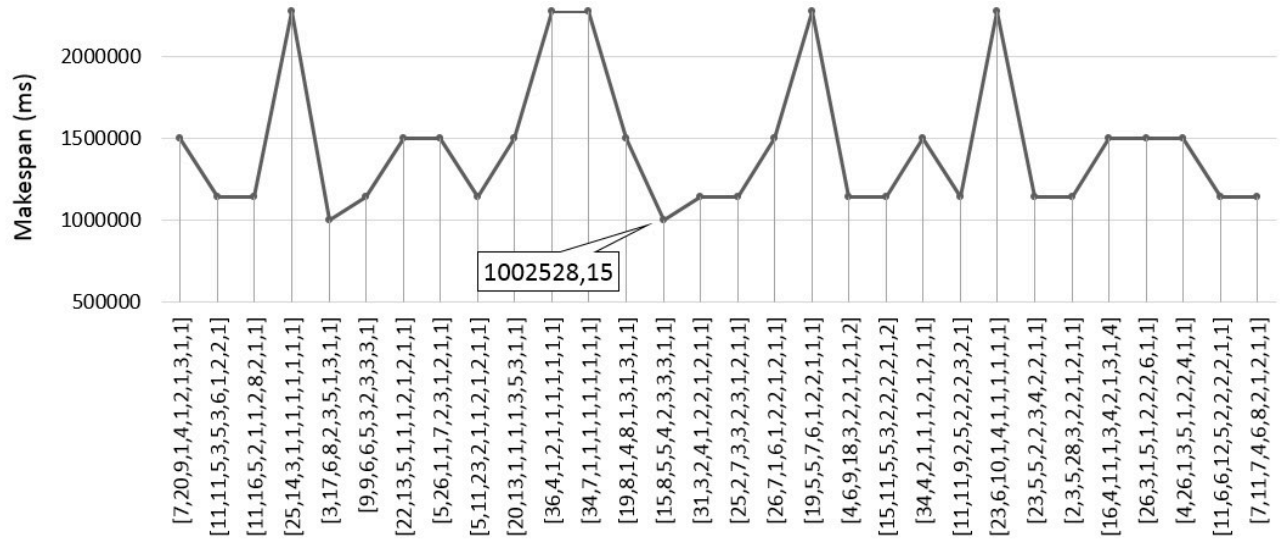


Fonte: Adaptado de FREIRE (2020).

4.5 Resultados

O menor *makespan* encontrado foi de 1002528,15 ms com as configurações [3,17,6,8,2,3,5,1,3,1,1] e [15,8,5,5,4,2,3,3,3,1,1]. A Figura 3 demonstra o gráfico com o *makespan* de todas as configurações de *threads* que resultaram da execução do algoritmo. A partir deste estudo de caso foi possível encontrar as configurações para os *pools* de *threads* locais, que alcançam o menor *makespan*. A nossa reflexão está no fato de que cada configuração que tende a apresentar um comportamento e uma alocação equivocada de *threads* em um *pool* acaba gerando um efeito nos demais *pools*, já que terá uma quantidade menor de *threads* a serem consideradas. Observamos que o tempo de processamento das tarefas que obtivemos com base no sistema real influi na correta distribuição das *threads*, para isso esse tempo precisa ser melhor compreendido e concluímos que tarefas mais lentas precisam de mais recursos e isso explica algumas discrepâncias entre as configurações. Podemos encarar as configurações aqui citadas como as duas configurações ideais de *threads* nos *pools* locais para o estudo de caso apresentado, pois foram elas que encontraram o menor *makespan* para este processo de integração. Precisamos destacar também que a comparação entre a população de configurações obtidas, mostra que existe diferença entre as muitas combinações possíveis e que existe uma configuração ótima ou próxima de ótima a ser obtida, tendo em vista que uma distribuição equivocada piora comprovadamente o *makespan*, mesmo com a mesma quantidade total de *threads*. No experimento que realizamos, uma distribuição equivocada pode ser duas vezes pior que uma configuração que classificamos como ideal.

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



Fonte: Os autores.

4.6 Ameaças à validade

Possíveis ameaças neste trabalho são a instrumentação e a fonte de ruído. Para suavizar essa possibilidade, antes de ser realizada a simulação da execução, definimos a questão de pesquisa, e a partir dela formulamos a hipótese e definimos as variáveis, sendo as dependentes e as independentes. Foram fornecidas na sequência, todas as informações necessárias sobre o ambiente da execução da experimentação, as ferramentas que deram suporte, e a descrição da execução e da coleta de dados. Toda a execução do experimento foi realizada na mesma máquina, em que foram desabilitados os processos desnecessários, e ainda a máquina foi desconectada da internet durante todo o experimento, tudo isso para minimizar a interferência do tempo de execução do algoritmo e possíveis ruídos. Este estudo pode ser generalizado para plataformas de integração que adotam os padrões de integração por Hohpe e Woolf (2003), pelo estilo arquitetural *pipes-and-filters* (ALEXANDER, 1977) e modelo de execução *task-based* utilizando a estratégia de *pool* local. O experimento é válido para testar outros parâmetros, como outros processos de integração, com outro número de mensagens, *threads*, soluções e gatos.

5. CONCLUSÃO

As empresas sempre estão em constante avanço e melhorando seus processos de negócio. Para isso é necessário muitas vezes integrar diversas aplicações do ecossistema de *software*. Um processo de integração realiza essa integração com o objetivo de atender a um processo de negócio. Neste artigo, foi apresentada uma proposta para que pudesse ser encontrada uma configuração ideal para o *pool* de *threads* seguindo a estratégia de *pools* locais, e com isso, minimizar o *makespan* em um processo de integração. Conseguimos encontrar a configuração ideal dos *pools* de threads a partir do algoritmo de otimização que foi baseado na meta-heurística CSO, e que foi implementado e executado. Essa configuração ideal gerou o menor *makespan* coletado no experimento, fazendo com que a hipótese seja confirmada. Com o objetivo de minimizar o *makespan*, era possível encontrar a configuração ideal dos *pools* de *threads* locais e refletir sobre o impacto que uma

distribuição equivocada pode causar dado o mesmo número de *threads*. O experimento descrito, pode ser utilizado em outros contextos e modificando as variáveis independentes. Pretendemos como trabalho futuro, ampliar os conhecimentos obtidos, realizando novos experimentos.

REFERÊNCIAS

ALEXANDER, Christopher. **A pattern language: towns, buildings, construction.** Oxford university press, 1977.

ALKHANAK, Ehab Nabiel et al. Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: A review, classifications, and open issues. **Journal of Systems and Software**, v. 113, p. 1-26, 2016.

BILGAIYAN, Saurabh; SAGNIKA, Santwana; DAS, Madhabananda. Workflow scheduling in cloud computing environment using cat swarm optimization. In: **2014 IEEE International Advance Computing Conference (IACC)**. IEEE, 2014. p. 680-685.

BLYTHE, James et al. Task scheduling strategies for workflow-based applications in grids. In: **CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid**, 2005. IEEE, 2005. p. 759-767.

BOEHM, Matthias et al. **Cost-based vectorization of instance-based integration processes.** Information Systems, v. 36, n. 1, p. 3-29, 2011.

CHIRKIN, Artem M. et al. Execution time estimation for workflow scheduling. **Future generation computer systems**, v. 75, p. 376-387, 2017.

DU, Y.; WANG, J. L.; LEI, L. Multi-objective scheduling of cloud manufacturing resources through the integration of Cat swarm optimization and Firefly algorithm. **Advances in Production Engineering & Management**, v. 14, n. 3, 2019.

FRANTZ, Rafael Z.; CORCHUELO, Rafael; MOLINA-JIMÉNEZ, Carlos. A proposal to detect errors in Enterprise Application Integration solutions. **Journal of Systems and Software**, v. 85, n. 3, p. 480-497, 2012.

FREIRE, Daniela L.; FRANTZ, Rafael Z.; ROOSâ€ÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂÂ?FRANTZ, Fabricia. Ranking enterprise application integration platforms from a performance perspective: An experience report. **Software: Practice and Experience**, v. 49, n. 5, p. 921-941, 2019.

FREIRE, Daniela L.; FRANTZ, Rafael Z.; ROOS-FRANTZ, Fabricia. Towards optimal thread pool configuration for run-time systems of integration platforms. **International Journal of Computer Applications in Technology**, v. 62, n. 2, p. 129-147, 2020.

FREITAS, FG de et al. Aplicação de metaheurísticas em problemas da engenharia de software: Revisão de literatura. In: **II Congresso Tecnológico Infobrasil.** 2009.

GABI, Danlami; ISMAIL, Abdul Samad; DANKOLO, Nasiru Muhammad. Minimized Makespan

Based Improved Cat Swarm Optimization for Efficient Task Scheduling in Cloud Datacenter. In: **Proceedings of the 2019 3rd High Performance Computing and Cluster Technologies Conference.** 2019. p. 16-20.

GUPTA, Swati; SINGH, Ravi Shankar; ANAND, Aniket. Cloudlet Scheduling using Merged CSO algorithm. In: **2018 Fifth International Conference on Parallel, Distributed and Grid Computing (PDGC).** IEEE, 2018. p. 278-283.

HARMAN, Mark; MANSOURI, S. Afshin; ZHANG, Yuanyuan. Search-based software engineering: Trends, techniques and applications. **ACM Computing Surveys (CSUR),** v. 45, n. 1, p. 1-61, 2012.

HAUGG, Igor G. et al. Towards optimisation of the number of threads in the integration platform engines using simulation models based on queueing theory. **Revista Brasileira de Computação Aplicada,** v. 11, n. 1, p. 48-58, 2019.

HOHPE, Gregor; WOOLF, Bobby. **Enterprise integration patterns: Designing, building, and deploying messaging solutions.** Addison-Wesley Professional, 2004.

JEDLITSCHKA, Andreas; PFAHL, Dietmar. Reporting guidelines for controlled experiments in software engineering. In: **2005 International Symposium on Empirical Software Engineering,** 2005. IEEE, 2005. p. 10 pp.

LEONARD, Naomi Ehrich; LEVINE, William S. **Using MATLAB to analyze and design Control Systems.** Benjamin-Cummings Publishing Co., Inc., 1995.

MANIKAS, Konstantinos. Revisiting software ecosystems research: A longitudinal literature study. **Journal of Systems and Software,** v. 117, p. 84-103, 2016.

MAURYA, Ashish Kumar; TRIPATHI, Anil Kumar. Deadline-constrained algorithms for scheduling of bag-of-tasks and workflows in cloud computing environments. In: **Proceedings of the 2nd International Conference on High Performance Compilation, Computing and Communications.** 2018. p. 6-10.

PERRY, Dewayne E.; PORTER, Adam A.; VOTTA, Lawrence G. Empirical studies of software engineering: a roadmap. In: **Proceedings of the conference on The future of Software engineering.** 2000. p. 345-355.

ROMERO, David; VERNADAT, François. Enterprise information systems state of the art: Past, present and future trends. **Computers in Industry,** v. 79, p. 3-13, 2016.

SEMLALI, Soukaina Cherif Bourki; RIFFI, Mohammed Essaid; CHEBIHI, Fayçal. Optimization of Makespan in Job Shop Scheduling Problem by Hybrid Chicken Swarm Algorithm. In: **International Conference on Advanced Intelligent Systems for Sustainable Development.** Springer, Cham, 2018. p. 358-369.

WOHLIN, Claes et al. **Experimentation in software engineering.** Springer Science & Business

Media, 2012.

Parecer CEUA: 640.285