SALÃO DO CONHECIMENTO UNIJUÍ 2018
CIÊNCIA PARA A REDUÇÃO DAS DESIGUALDADES

XXVI Seminário de Iniciação Científica
XXIII Jornada de Pesquisa
XIX Jornada de Extensão
VIII Mostra de Iniciação Científica Júnior
VIII Seminário de Inovação e Tecnologia

01 a 04 de outubro de 2018

Evento: Bolsistas de Iniciação Científica e Iniciação Tecnológica da Unijuí

ESTUDO E IMPLEMENTAÇÃO DA TÉCNICA PARTICLE SWARM OPTIMIZATION UTILIZANDO A LINGUAGEM R[1]
STUDY AND IMPLEMENTATION OF PARTICLE SWARM OPTIMIZATION TECHNIQUE USING R LANGUAGE

Gabriel Uberti Vieira Messina[2], Rafael Zancan Frantz[3]

[1] Projeto de pesquisa realizado no programa de Iniciação Científica
[2] Bolsista PIBIC/cNPQ
[3] Orientador do projeto de Iniciação Científica

1. INTRODUÇÃO

O campo conhecido como *Enterprise Application Integration* (EAI) busca a integração de várias aplicações de software que foram desenvolvidos de forma independente ou que não possui compatibilidade sendo gerenciado de forma independente.

Esta integração do sistema é realizada a partir de uma plataforma de integração, uma ferramenta voltada para a modelagem, implementação e execução de um ecossistema de software. Estas plataformas de integração na maioria dos casos oferecem um *kit* de desenvolvimento de software, uma linguagem de domínio específico, uma biblioteca para implementação das soluções modeladas e um motor de execução para as soluções de integração.

O motor de execução dessas plataformas que realizam a solução de integração modelada do usuário permitindo que ela seja executada, logo, o desempenho dos motores de execução é relacionado com o desempenho das soluções de integração, quanto melhor for o desempenho do motor de execução, melhor o desempenho da solução de integração. O motor de execução realiza o gerenciamento e manipulação dos mensagens, tarefas, *slots* e portas da integração.

Mensagem é uma informação que é lida e modificada por intermédio de uma integração solução. *Slots* são *buffers* conectando da saída de uma tarefa à entrada de outra tarefa, com isso, as mensagens são processadas de forma assíncrona pelas tarefas, um slot pode seguir diferentes políticas para servir as mensagens à tarefas, como uma saída baseada em prioridade onde a primeira a chegar, primeira a ser atendida. Já as tarefas tratam das operações a serem realizadas pelas mensagens como, ler, dividir, agregar, filtrar, entre outras. Uma tarefa pode ter uma ou mais entradas da qual recebe mensagens e uma ou mais saídas pelas quais as mensagens partem. Por meio de uma porta é possível estabelecer a leitura, escrever, solicitação e resposta às operações de comunicação com os recursos que estão sendo integrados. Processo de Integração contém lógica de integração que executa a transformação, roteamento, modificação e operações relacionadas ao tempo através de mensagens. Um processo de integração é composto por portas que permitem a comunicação com os recursos sendo integrados, *slots* e um conjunto de tarefas para especificar a lógica de integração conforme descrito por Fernando [4].

As plataformas atuais, trabalham com uma estrutura de dados do tipo fila, resumidamente, trata-se de uma estrutura da qual a primeira mensagem a chegar será a primeira a sair, esta técnica que conhecida pela sigla FIFO (*First In First Out*). Devido ao alto consumo e demanda cada vez maior por parte de usuários, a tendência é de que uma plataforma possa receber um maior número de mensagem para serem processadas. Logo, esta técnica apresenta problemas em seu desempenho, por que em uma integração que receba um enorme fluxo de mensagens, as quais serão processadas por diversas tarefas, e serão reintegradas para o final da fila, com o intuito de serem processadas por uma próxima tarefa, ocorrendo um gargalo de tempo para que uma única mensagem será finalizada por este processo de integração.

Com a utilização do método desenvolvido por Eberhart e Kennedy [3], o *Particle Swarm Optimization* oferece uma metodologia que pode servir como estrutura para realizar o escalonamento de plataformas de integração, com a principal ideia de fornecer um maior desempenho na troca de recursos. Oferecendo assim, uma maneira de evitar este gargalo de mensagens gerado pelo método *First In First Out*.

É neste contexto, que o artigo busca a implementação de um algoritmo, que possa realizar uma simulação de um agendamento de tarefas para a plataforma de integração. Desta maneira, visa explorar e demonstrar por que a utilização do método *Particle Swarm Optimization* possui um maior desempenho em sistemas que possuem um alto fluxo de mensagem. O projeto possui a pesquisa bibliográfica com base na teoria de Eberhart e Kennedy [3].

2. PARTICLE SWARM OPTIMIZATION

O *Particle Swarm Optimization* é inspirado pela observação de aves. Segundo Eberhart and Kennedy [3], a sincronia e o comportamento dos pássaros ocorriam por intermédio das distâncias entre membros individuais e seus vizinhos. Assim, a velocidade desempenha um importante papel ao se ajustar mutuamente para a distância ideal. Além disso, os cientistas simularam um cenário, em que as aves buscam por alimentos e observam seu comportamento social. Eles perceberam que, para encontrar comida, os membros individuais determinaram suas velocidades por dois fatores, sua melhor experiência anterior e a melhor experiência de todos os outros membros. Isso é semelhante ao comportamento humano na tomada de decisões, em que as pessoas consideram a melhor experiência anterior e a melhor experiência da forma em que as outras pessoas ao seu redor obtiveram.

A. Algoritmo

Na área computacional, o primeiro desenvolvimento de um algoritmo PSO para o problema do atraso ponderado total de uma única máquina, utilizou-se da regra de valor de posição menor, um mecanismo de ordem não decrescente. Além disso, um método de busca local bem conhecido, chamado de busca de vizinhança variável, foi aplicado para melhorar a solução. Desta forma, com a mesma abordagem, resolveu dois problemas de fluxo de permutação com os critérios de minimização de tempo em que todos as mensagens tiverem terminado o processamento e

minimização de atrasos máximos, respectivamente.

Conforme descrito por Daniela [2], 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 a função de aptidão. Neste caso, uma partícula representa um fluxo de trabalho e suas tarefas e a dimensão da partícula é igual ao número de tarefas no fluxo de trabalho. A dimensão de uma partícula serve para localizar sua posição no espaço, definindo o sistema de coordenadas. Confome figura a seguir.

B. Estudo de Caso

O estudo foi realizado com a utilização da linguagem R [7], linguagem e ambiente para computação estatística, e executado sobre a plataforma RStudio [6].

A idéia definida, conforme descrita por Daniela [2], parte do princípio de gerar uma tabela com *n* testes, definindo assim o número de threads para cada um de seus pools de tarefas, onde cada teste apresenta uma população distinta. Após gerada a população é realizado um laço de repetição com a finalidade de calcular o tempo de execução de cada teste, até que atinja o número máximo de iterações. O tempo de execução consiste em realizar o somatório de tempo de cada tarefa, tantas vezes quanto o somatório dos pools da população atual e por fim, realizar a soma de seu tempo ocioso (tempo no qual uma ou mais *thread(s)* ficam esperando a entrada de uma nova mensagem). E sua etapa final consiste em realizar a varredura dos tempos e verificar sua configuração com o melhor tempo de execução (*gBest*). Conforme o pseudo-código a seguir.

```
for cada numero de teste do
  gerar a populacao de threads em cada tarefa
end for
define gBest igual ao pBest
repeat
  for cada teste do
      calcule o tempo de execucao, salve o valor
          obtido
        atualize as variaveis gBest e pBest
    end for
until maximo de iteracoes atingidas
```

Para a realização da execução foi criado um ambiente com a configuração a seguir:

Total de *threads*: 6;

Total de mensagens: 500.000;

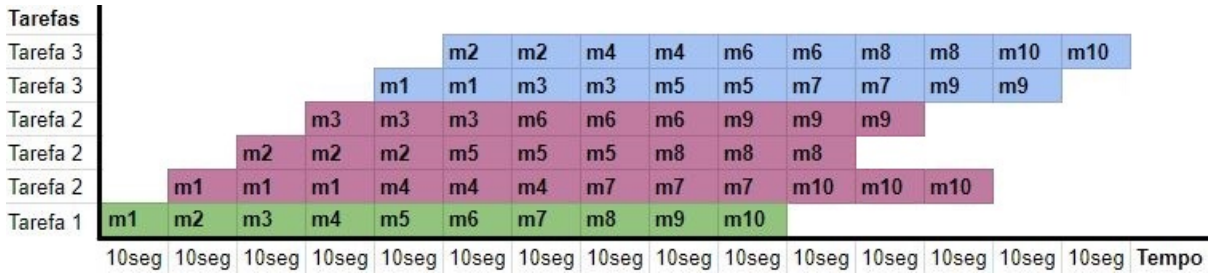
Total de tarefas: 3;

Tempo da tarefa 1: 10 segundos;

Tempo da tarefa 2: 30 segundos;

Tempo da tarefa 3: 20 segundos.

Com base neste ambiente é possível chegar a uma configuração de *threads* na qual há um desperdício mínimo em termos de tempo ocioso, da qual resultou em 1, 3 e 2 *threads* para seus respectivos *pools* nas tarefas 1, 2 e 3. Conforme a figura a seguir. Pode-se demonstrar a entrada das suas dez primeiras mensagens, verifica-se que não há tempo ocioso entre suas tarefas, entretanto, as mensagens que entram no processo de integração, requerem um tempo mínimo de ociosidade de tarefas subsequentes as quais esperam suas anteriores terminar a sua execução.



3. CONCLUSÃO

A partir da pesquisa teórica baseada nos estudos e Eberhart e Kennedy [3] e constatou-se que um algoritmo FIFO possui um gargalo de tempo no processamento de mensagens de acordo com a entrada de mensagens, a grosso modo, em um processo de integração que possui um alto índice de entrada, ocorre também um aumento na taxa de tempo para que uma única mensagem seja processada.

Dessa forma, o objetivo do artigo era a implementação de um agendador de tarefas utilizando a técnica do Particle Swarm Optimization, concluiu-se que por mais que ocorra uma grande entrada de mensagens, esta técnica permite que os *pools* de *threads* bem configurados, tenham um melhor aproveitamento no processamento de mensagens.

Concluiu-se também que a implementação deste algoritmo em um ambiente utilizando linguagem e uma plataforma distinta, levando em comparação a implementação solucionada pela Daniela [2], criando um mesmo ambiente de entrada de mensagens, total de *threads*, tarefas... Chegou à um mesmo resultado, desta forma, podemos ver que a utilização do *Particle Swarm Optimization* depende em si da técnica e não da plataforma ou linguagem em que a mesma está inserida.

Evento: Bolsistas de Iniciação Científica e Iniciação Tecnológica da Unijuí