Relatório Técnico da Terceira Avaliação de Sistemas Distribuídos

Vandirleya Barbosa da Costa[1], **Viviane Barbosa Leal Dias**[1]

[1]Universidade Federal do Piauí (UFPI)
Picos – PI – Brasil

vandirleya.barbosa@ufpi.edu.br, vivianebarbosa@ufpi.edu.br

Resumo. *Este trabalho compara as abordagens sequencial, paralela e distribuída para resolver problemas de otimização utilizando Algoritmos Genéticos (AGs). A implementação foi testada em um cenário de minimização de atrasos em redes distribuídas representadas por grafos. Os resultados mostram que a abordagem sequencial é a mais rápida devido à ausência de sobrecargas de comunicação e gerenciamento de threads, enquanto a paralela e a distribuída apresentaram maior escalabilidade e eficiência em cenários complexos, com tempos de execução e atrasos reduzidos.*

Abstract. *This work compares the sequential, parallel, and distributed approaches for solving optimization problems using Genetic Algorithms (GAs). The implementation was tested in a scenario focused on minimizing delays in distributed networks represented by graphs. The results show that the sequential approach is the fastest due to the absence of communication overhead and thread management, while the parallel and distributed approaches demonstrated greater scalability and efficiency in complex scenarios, with reduced execution times and delays.*

1. Introdução

Os AGs são amplamente reconhecidos como ferramentas poderosas e versáteis para resolver problemas complexos de otimização que desafiam métodos convencionais. Inspirados nos processos evolutivos observados na natureza, esses algoritmos simulam mecanismos como seleção natural, cruzamento genético e mutação para explorar vastos espaços de busca em busca de soluções próximas do ótimo global. Sua habilidade em lidar com funções objetivo não lineares, múltiplos picos de otimização e restrições complexas os torna uma escolha eficiente para uma variedade de aplicações. Em especial, na área de redes distribuídas, os AGs se destacam ao balancear múltiplas variáveis, lidar com interdependências e atender a critérios rigorosos de desempenho, robustez e eficiência, mesmo em cenários dinâmicos e incertos.

A computação moderna oferece um conjunto diversificado de arquiteturas para a execução de algoritmos, sendo as abordagens sequencial, paralela e distribuída as mais amplamente empregadas devido às suas características e capacidades distintas. A abordagem sequencial, tradicional e de implementação direta, processa as tarefas em uma sequência linear, o que é adequado para problemas pequenos ou moderados, mas se torna limitante em cenários de grande escala devido ao tempo necessário para explorar espaços de busca complexos. Já a abordagem paralela, que distribui tarefas entre múltiplos núcleos

de processamento em uma única máquina, representa um avanço significativo, permitindo reduções substanciais no tempo de execução ao explorar o paralelismo intrínseco de muitos algoritmos. Por fim, a abordagem distribuída vai além, aproveitando o poder computacional de diversas máquinas conectadas em rede. Essa arquitetura é particularmente eficaz para problemas que demandam alta escalabilidade, maior capacidade de processamento e resiliência.

No entanto, o uso de sistemas distribuídos também introduz desafios, como a necessidade de coordenação eficiente entre os nós, latências na comunicação e a complexidade de sincronizar estados entre máquinas distribuídas. Essas diferenças tornam a escolha da arquitetura um aspecto essencial para maximizar o desempenho de algoritmos em cenários específicos. No presente trabalho, três abordagens são exploradas para resolver o problema de encontrar o menor conjunto de nós em uma rede distribuída que permita a transmissão eficiente de serviços a clientes, minimizando atrasos. A proposta consiste em aplicar um AG para avaliar diferentes configurações de rede, onde os nós representam componentes com funções específicas, e os enlaces possuem atrasos atribuídos aleatoriamente. A análise comparativa das abordagens sequencial, paralela e distribuída é realizada com foco na identificação de vantagens, desvantagens e possibilidades de melhoria, considerando métricas como desempenho, confiabilidade e escalabilidade.

2. Metodologia

Esta seção apresenta a resolução do problema abordado neste trabalho. O início da implementação consistiu em modelar a minimização do atraso de transmissão em uma rede distribuída, representada por um grafo não direcionado. Nesse grafo, os nós correspondem a componentes distribuídos, cada um associado a uma função específica, enquanto os enlaces representam os atrasos de comunicação. A construção do grafo envolveu a geração de nós aleatórios, aos quais foram atribuídas funções selecionadas de um conjunto pré-definido, e a conexão desses nós por enlaces com atrasos aleatórios. Além disso, serviços foram definidos como conjuntos aleatórios de 3 a 10 funções, representando as operações requeridas pelos clientes. Cada requisição foi caracterizada por um nó de origem, um nó de destino e um serviço a ser atendido. Para garantir a consistência das requisições, implementou-se um validador que verificava, por exemplo, que o nó de origem não coincidia com o destino e que todas as funções do serviço estavam presentes em nós distintos da rede. Três abordagens computacionais foram desenvolvidas para resolver o problema de forma eficiente: sequencial, paralela e distribuída.

Na abordagem sequencial, cada requisição foi processada linearmente. O cálculo do caminho de menor atraso para atender a uma requisição foi realizado iterativamente, partindo do nó de origem e buscando nós vizinhos que possuíssem a função requerida em cada etapa do serviço. A cada transição entre nós, o atraso correspondente era somado ao total, e o nó atual era atualizado até que todas as funções do serviço fossem atendidas. Essa abordagem, embora simples, mostrou limitações em cenários de maior carga, devido à ausência de paralelismo no processamento. A abordagem paralela introduziu a divisão de tarefas entre múltiplas threads, utilizando o modelo de concorrência baseado em threads. Para cada requisição, as buscas pelas funções necessárias foram realizadas em paralelo, com cada thread sendo responsável por encontrar o próximo nó adequado. A coordenação entre threads foi gerenciada por meio de um executor de threads, reduzindo o tempo total de execução ao explorar os recursos multicore da CPU. Ape-

sar disso, essa abordagem apresentou sobrecarga associada ao gerenciamento de threads e à sincronização das operações. A versão distribuída ampliou a escala do processamento ao distribuir as tarefas entre múltiplos processos, potencialmente executados em diferentes núcleos ou máquinas. Utilizando o módulo de multiprocessing, cada processo foi responsável por localizar a próxima função no caminho, com a coordenação sendo feita por meio de um pool de processos. Essa abordagem demonstrou maior escalabilidade em comparação com as anteriores, mas introduziu desafios adicionais relacionados à comunicação interprocessual e ao gerenciamento de latência entre nós.

O AG foi empregado como uma abordagem de otimização baseada na evolução natural. Nele, cada indivíduo representava uma sequência de nós candidata para atender a uma requisição. O processo evolutivo incluiu a geração de uma população inicial, avaliação de aptidão com base no atraso total de cada caminho, e aplicação de operadores de cruzamento e mutação para gerar novas soluções. O algoritmo foi executado por múltiplas gerações, resultando em indivíduos otimizados para minimizar o atraso. Essa abordagem permitiu explorar o espaço de busca de maneira robusta, combinando soluções promissoras para encontrar caminhos eficientes. Para avaliar o desempenho das diferentes abordagens, foram realizados testes sistemáticos com 30 execuções para cada uma das 10 requisições geradas, totalizando 900 execuções. Durante os testes, métricas como atraso mínimo, médio, máximo e desvio padrão foram registradas, assim como os tempos de execução para cada abordagem. O código foi desenvolvido neste Colab: **Terceira avaliação de SD**[1].

3. Resultados

Esta seção apresenta os resultados obtidos na resolução do trabalho. A Tabela 1 resume as métricas de tempo e delay para três categorias de execução: sequencial, paralela e distribuída. Na categoria sequencial, o tempo médio foi extremamente baixo (4.09×10^{-5}), indicando alta rapidez, mas o delay apresentou média de 8.91 e um valor máximo de 29, destacando limitações na capacidade de lidar com múltiplas requisições. Já na execução paralela, o tempo médio aumentou ligeiramente (0.00104), enquanto o delay reduziu para uma média de 7.94 e um máximo de 28, refletindo uma melhora no balanceamento de desempenho. Na categoria distribuída, os valores de tempo foram os mais altos (média de 0.0425), mas os resultados de delay permaneceram idênticos aos da execução paralela, sugerindo que ambas as abordagens são equivalentes no impacto sobre o atraso. Além disso, foram detectados 247 erros durante os testes, o que pode indicar inconsistências nas execuções ou limitações nos algoritmos implementados, especialmente em cenários com maior complexidade operacional.

A Figura 1 apresenta as métricas de tempo (Min, Mean, Max e Std) para as abordagens sequencial, paralela e distribuída. A abordagem sequencial mostrou os menores valores em todas as métricas, de modo que a barra nem os exibe. A paralela teve tempos maiores, oferecendo melhor equilíbrio entre desempenho e estabilidade. Já a distribuída apresentou os maiores tempos, com média de 0.0425 e máximo de 0.0564, refletindo maior complexidade.

A Figura 2 apresenta as métricas de delay (Min, Mean, Max e Std) para as aborda-

[1] `https://colab.research.google.com/drive/1KJ04rXIkYcFGRm8fiTfvAAuz88ypXfmt?usp=sharing`

Table 1. Comparação de métricas de tempo e delay.

Abordagem	Métrica	Min	Mean	Max	Std
Sequencial	Delay	0	8.91	29	6.87
	Tempo	1.31×10^{-5}	4.09×10^{-5}	7.30×10^{-5}	1.30×10^{-5}
Paralela	Delay	0	7.94	28	6.21
	Tempo	0.00077	0.00104	0.00347	0.00040
Distribuída	Delay	0	7.94	28	6.21
	Tempo	0.0377	0.0425	0.0564	0.00366

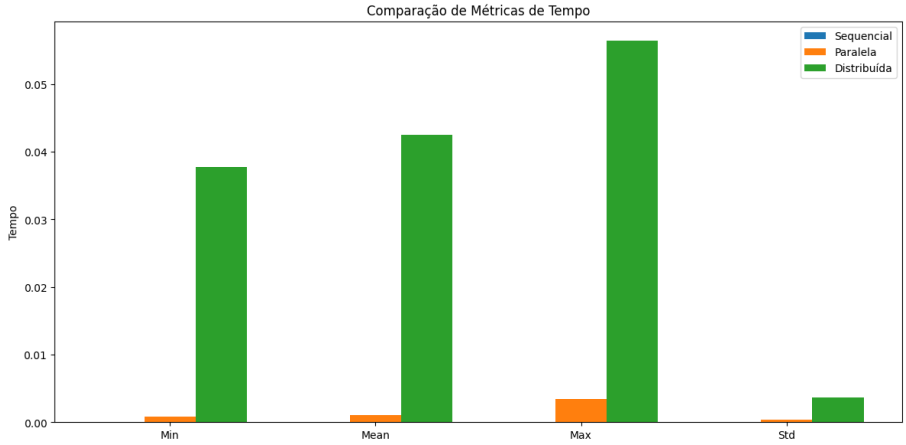

Figure 1. Comparação estatística do tempo de execução de cada abordagem.

gens sequencial, paralela e distribuída. A abordagem sequencial apresentou o maior delay médio (8.91) e máximo (29), indicando uma maior demora na execução em comparação com as outras abordagens. As abordagens paralela e distribuída obtiveram resultados idênticos, com delay médio de 7.94 e máximo de 28, sugerindo que ambas oferecem melhorias no desempenho ao reduzir o atraso em relação ao método sequencial. A variação do delay (Std) foi semelhante para todas as abordagens, com valores próximos a 6.2, evidenciando consistência nos resultados. Esses dados destacam a eficiência das abordagens paralela e distribuída para cenários onde a redução de atrasos é crucial.

4. Conclusão

Este trabalho realizou a análise das abordagens sequencial, paralela e distribuída para execução de AGs revelando diferenças significativas no desempenho. A abordagem sequencial destacou-se pela rapidez, sendo a mais eficiente em termos de tempo devido à sua simplicidade e à ausência de sobrecargas associadas ao gerenciamento de threads e comunicação. No entanto, sua escalabilidade é limitada, tornando-a inadequada para problemas mais complexos. Por outro lado, a abordagem paralela mostrou-se um avanço significativo ao dividir o trabalho entre múltiplos núcleos, reduzindo o tempo de execução para grandes volumes de requisições. Contudo, sua eficiência foi comprometida por sobrecargas de gerenciamento de threads e sincronização. A abordagem distribuída, embora apresente os maiores tempos de execução devido à comunicação entre processos, oferece

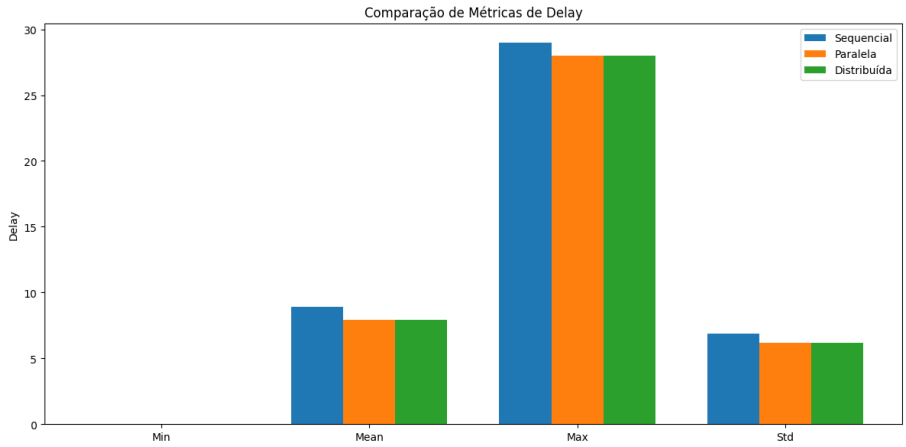

Figure 2. Comparação estatística do delay de cada abordagem.

escalabilidade superior e é ideal para aplicações de grande escala. Melhorias podem ser realizadas na versão distribuída, como o balanceamento de carga entre nós para evitar ociosidade, ajustes no tamanho das subpopulações para otimizar a evolução local e a comunicação, além de implementar conteinerização para garantir maior portabilidade e interoperabilidade. A confiabilidade do sistema poderia ser ampliada com a configuração de réplicas redundantes para processos críticos, assegurando recuperação em caso de falhas. Portanto, a escolha da abordagem depende do contexto e das demandas específicas da aplicação, variando entre a simplicidade da execução sequencial, o equilíbrio oferecido pelo paralelismo e a escalabilidade proporcionada pelo processamento distribuído.