Fast LCA - MapReduce-based Algorithm for solving the Lowest Common Ancestor

Fast LCA - Algoritmo baseado em MapReduce para solução do Menor Ancestral Comum

Jorgi Luiz Bolonhezi Dias[1], **Francisco Sanches Banhos Filho**[1]

[1]Departamento de Informática – Universidade do Estado de Mato Grosso (Unemat)
Av. dos Ingas, 3001 - Jardim Imperial, Sinop – MT, 78555-000 – Brazil

{jorgi.luiz,fsanches}@unemat.br

Abstract. *In current times, where the volume of information is rapidly expanding, the analysis of problems that can be represented by graphs emerges as an intriguing approach. One of these problems is the Lowest Common Ancestor, widely used in phylogenetics and various other research areas. In this study, we introduce an algorithm that addresses the Lowest Common Ancestor problem, employing the MapReduce paradigm and implemented with the assistance of the Apache Hadoop Framework. We executed the algorithm on a computer cluster and assessed its performance through measures of speedup and efficiency.*
Keywords: *MapReduce. Hadoop. Cluster. Lowest Common Ancestor. Parallel Processing.*

Resumo. *Nos tempos atuais, onde a quantidade de informações está crescendo rapidamente, a análise de problemas que podem ser representados por grafos emerge como uma abordagem intrigante. Um desses problemas é o Menor Ancestral Comum, amplamente utilizado na filogenética e em diversas outras áreas de pesquisa. Neste estudo, apresentamos um algoritmo que aborda o problema do Menor Ancestral Comum, empregando o paradigma MapReduce e implementado com o auxílio do Framework Apache Hadoop. Executamos o algoritmo em um cluster de computadores e avaliamos seu desempenho por meio de medidas de speedup e eficiência.*
Palavras-chave: *MapReduce. Hadoop. Cluster. Menor Ancestral Comum. Computação Paralela.*

1. Introdução

O presente trabalho introduz um algoritmo que aborda eficazmente o problema do *Lowest Common Ancestor* (LCA), utilizando o modelo *MapReduce* e, por consequência, o *framework* Apache Hadoop. A aplicação desse algoritmo permitiu a resolução do desafio envolvendo árvores de bilhões de vértices e arestas.

O problema do LCA possui relevância em diversas áreas de pesquisa e se concentra na determinação do ancestral comum mais próximo entre dois vértices especificados. Em outras palavras, dado um grafo em forma de árvore e dois vértices arbitrários, o desafio é identificar o ancestral compartilhado mais próximo entre esses vértices. Um campo

de pesquisa onde esse problema desempenha um papel crucial é a filogenia. A filogenia, baseada na teoria da evolução, postula que grupos de organismos com características semelhantes têm um ancestral comum.

O algoritmo desenvolvido, juntamente com o ambiente de teste composto por 16 computadores, demonstrou sua capacidade de solucionar eficientemente problemas relacionados a árvores com tamanhos variando de 1 milhão a 1 bilhão de vértices, com graus de ramificação de 2, 3 e 4.

2. Grafos e o Menor Ancestral Comum

O desafio central em questão se resume à identificação do *Lowest Common Ancestor* (LCA), ou seja, o ancestral comum mais baixo, entre dois vértices arbitrários selecionados a partir de uma árvore T [Aho et al. 1973]. Essencialmente, trata-se de localizar o vértice ancestral mais distante da raiz que conecta esses dois vértices. Por exemplo, ao analisar a Figura 1, ao escolher os vértices B e C, o LCA será o vértice A. De forma análoga, ao selecionar os vértices I e G, o LCA será o vértice D.

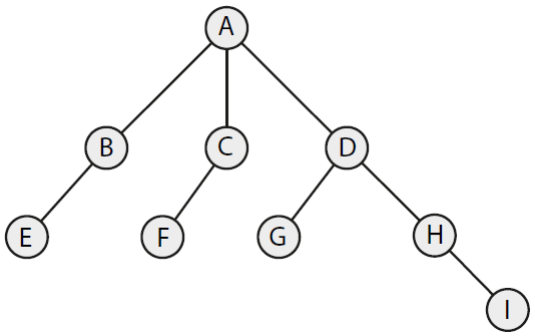

Figura 1. Grafo Árvore T

Apesar de aparentar simplicidade, esse problema adquire complexidade considerável quando aplicado a árvores de grande porte, compostas por bilhões de vértices e arestas. A magnitude dos dados o transforma em um desafio caracterizado pelo *Big Data*. Algoritmos capazes de resolver o problema do LCA têm desempenhado um papel fundamental em *softwares* dedicados à análise de metagenomas, que visam determinar a composição taxonômica. No entanto, os sequenciadores de última geração atualmente geram volumes massivos de dados, o que torna a análise computacionalmente desafiadora, dada a limitação de recursos de *hardware* disponíveis [Zhao et al. 2012].

3. Modelo MapReduce e o framework Apache Hadoop

A computação paralela é caracterizada pela capacidade de dividir um processo computacional em subprocessos que podem ser executados simultaneamente, de forma independente, em diferentes processadores [Kumar et al. 1994]. É uma abordagem que se tornou fundamental em diversas áreas da computação.

O modelo *MapReduce* é um paradigma de programação e processamento de dados amplamente utilizado em computação distribuída e *Big Data*. Ele divide uma tarefa em duas etapas principais: *map*, na qual os dados são fragmentados e processados em

paralelo, resultando em pares chave-valor, e *reduce*, onde os dados são agrupados e processados com base em suas chaves. Isso permite o processamento eficiente de grandes volumes de dados, aproveitando a escalabilidade e a capacidade de paralelização, sendo fundamental em sistemas que lidam com análise e transformação de dados em larga escala [Dean and Ghemawat 2008].

No desenvolvimento de soluções distribuídas, um desafio significativo é garantir a distribuição equitativa do processamento entre os vários processadores de um *cluster*, bem como gerenciar eficazmente as tarefas. Uma abordagem para mitigar essa complexidade é o modelo *MapReduce* e o *framework* Apache Hadoop, que simplificam o desenvolvimento de aplicações distribuídas.

O ecossistema Hadoop é uma plataforma Java de código aberto para armazenamento e processamento de dados em larga escala. A configuração de *clusters* Hadoop é simples, enquanto a integridade dos dados, a disponibilidade de nós, a escalabilidade de aplicativos e a recuperação de falhas são tratadas automaticamente de forma transparente para o usuário. Além disso, o Hadoop oferece um processamento superior em comparação com outras tecnologias da área [White 2015].

As atualizações no ecossistema Hadoop, indo da versão 1.x para a 2.x, trouxeram mudanças notáveis, incluindo a introdução do YARN para gerenciamento de recursos, um único *Namenode* ativo e *standby*, um balanceador HDFS e a capacidade de escalabilidade para até 10.000 nós. O Hadoop 3.x trouxe melhorias importantes, como o uso do *Erasure Coding* (EC) para economia de espaço, múltiplos *Namenodes* standby, um balanceador de dados intra-nó e um aumento de desempenho de mais de 40% em operações *TeraSort* em comparação com versões anteriores, especialmente em operações de leitura e gravação [Masur and Mcintosh 2017].

4. O Algoritmo

O algoritmo adota uma abordagem bifásica composta por etapas distintas: a fase *Map* e a fase *Reduce*. A Figura 2 ilustra um diagrama de fluxo da fase *Map*, na qual, após o arquivo contendo a estrutura da árvore ser devidamente alocado no *Hadoop Distributed File System* (HDFS), é realizado um processo de leitura linha por linha. Nesta etapa, apenas os dados relativos a cada vértice e seu respetivo ancestral são emitidos para a fase *Reduce*.

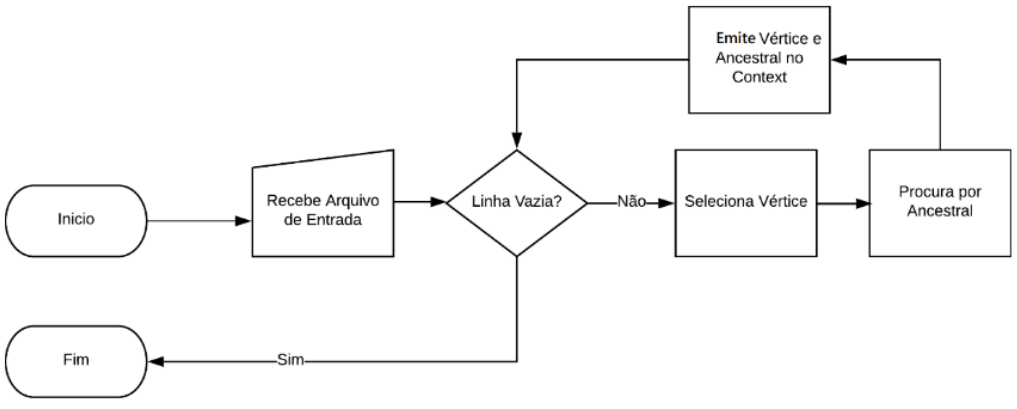

Figura 2. Fluxograma Map

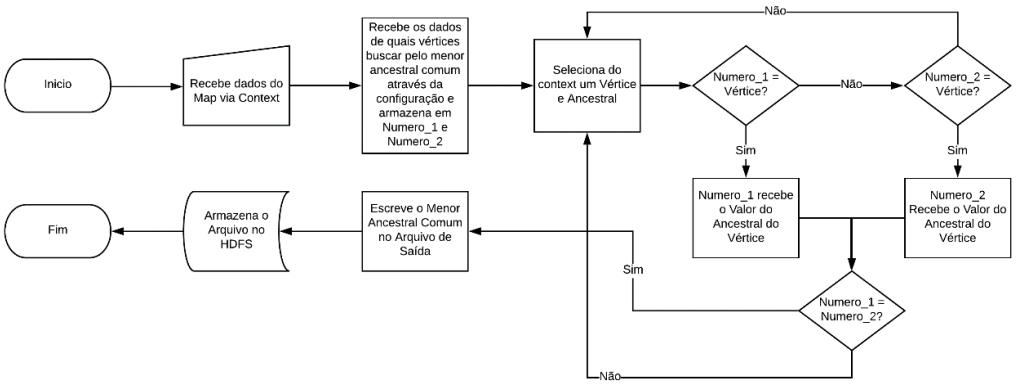

Figura 3. Fluxograma Reduce

A fase *Reduce*, representada na Figura 3, recebe os dados referentes aos vértices e seus ancestrais, transmitidos pela fase *Map* através do contexto de execução. A partir desse ponto, inicia-se uma sequência de ações de natureza comparativa para cada linha do arquivo. Durante esse processo, é traçado o percurso de cada vértice indicado até a raiz da árvore, com o objetivo de identificar o menor ancestral comum.

5. Metodologia

Tabela 1. Ambiente de Testes

Categoria	Parâmetro	Valor
Hardware		
Cluster	Número de nodos	16
	Número de racks	2
Nodo	CPU	Intel® Core i7-2600 CPU 3.4GHz
	Número de cores	4
	Número de processos	8
	RAM	8GB
Rede	Juniper EX2200-24T-4G	1GbE
Cargas de Trabalho		
Config.	Jobtracker	1
	Namenodes	1
	Número de trabalhadores	14
	Maps por nodo	1
	Reduces por nodo	1
TCP buffer	Tamanho padrão	64 KB por conexão
TCP packet	Tamanho padrão	1500 bytes

A tabela 1 detalha o cenário dos testes. Configuramos um *cluster* com 16 nós, cada nó equipado com um processador Intel® Core™ i7-2600 quad-core de 3.4GHz e uma única

Tabela 2. Árvores Utilizadas

Vértices	Ramificação	Tamanho	Vértices	Ramificação	Tamanho
1 Milhão	2, 3 e 4	27,4 MB	200 Milhões	2	7,4 GB
10 Milhões	2, 3 e 4	314,4 MB	300 Milhões	2	11,4 GB
30 Milhões	2, 3 e 4	1 GB	400 Milhões	2	15,4 GB
50 Milhões	2, 3 e 4	1,7 GB	500 Milhões	2	19,4 GB
100 Milhões	2	3,5 GB	1 Bilhão	2	39,4 GB

conexão na parte superior do rack (ToR) de 1GbE. O *oversubscription ratio* em nossos testes foram de 1.6:1, o que corresponde à recomendação da Cisco para implantações de *MapReduce* com um *oversubscription ratio* de 4:1 ou menor na camada de acesso [Systems 2011]. Conforme mencionado por [Hortonworks 2016], ter um baixo *oversubscription ratio* é uma maneira de melhorar o desempenho da rede com investimentos em equipamentos. Todas as configurações dos *switches*, como *buffers* e consumo de energia, são baseadas na série Juniper EX2200-24T-4G [Networks 2013], um produto padrão da indústria.

As cargas de trabalho também podem ser vistas na tabela 1. Reservamos um nó para a execução do *JobTracker*, enquanto 14 nós estão reservados para a execução das tarefas de *Map* e *Reduce*. Além disso, reservamos outro nó para executar um *Namenode*, o que é recomendado pelo Apache, oferecendo assim um equilíbrio entre escalabilidade e sobrecarga de comunicação [Foundation 2017]. Nos testes foram utilizadas 18 árvores com grau de ramificação 2, 3 e 4, e então executadas 9 buscas (3 no início, 3 no meio e 3 no fim) para cada árvore da tabela 2.

6. Resultados

A figura 4 exibe os resultados da execução do algoritmo *Fast LCA* em doze árvores distintas. Estas árvores variam em tamanho, com 1, 10, 30, e 50 milhões de vértices, e também em seus fatores de ramificação, que são 2, 3 e 4. Notavelmente, na Figura 4, observa-se que, para a maioria das árvores, mesmo quando o fator de ramificação é modificado, não se verifica uma variação significativa no tempo de processamento. Entretanto, em árvores com 50 milhões de vértices, observa-se uma leve redução no tempo de processamento com um fator de ramificação de 4. Isso sugere que, para árvores desse tamanho, o uso de 16 computadores pode começar a compensar, ao passo que para árvores menores, não é necessária a utilização de todos os computadores disponíveis no *cluster*.

A figura 5 apresenta a execução do algoritmo em uma árvore com 50 milhões de vértices e fator de ramificação 2. O teste foi realizado com 2, 4, 8 e 16 computadores, considerando vértices no início, meio e fim da árvore para calcular o LCA. O objetivo principal deste gráfico é demonstrar que, apesar das variações, a proximidade ou distância em relação à raiz não influencia significativamente o tempo de execução do algoritmo.

Na figura 6, o algoritmo foi executado em árvores que variam de 1 a 100 milhões de vértices, todas com fator de ramificação 2 (árvores binárias). Para este teste, utilizou-se de 1 a 16 computadores. Observou-se que, mesmo dispondo de mais recursos para uso, o gerenciador do Hadoop YARN não empregou todos os recursos disponíveis. Isso pode ser atribuído, principalmente, ao tamanho das árvores, que não era suficientemente

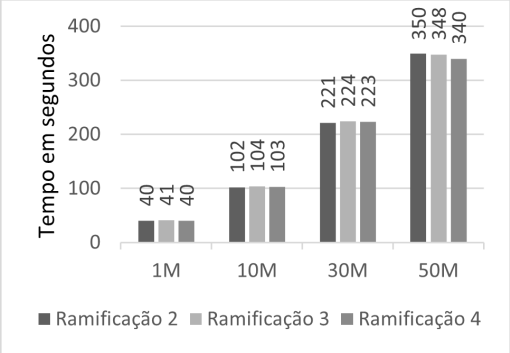

Figura 4. Buscas em Árvores com Ramificação 2, 3 e 4

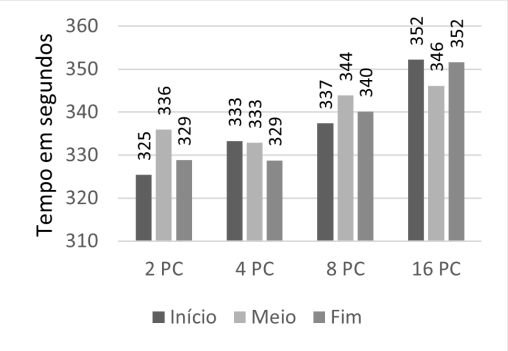

Figura 5. Árvore com 50 Milhões de Vértices de Ramificação 2

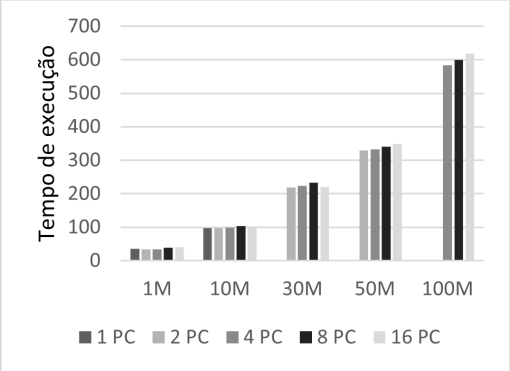

Figura 6. Execuções de 1 a 16 máquinas

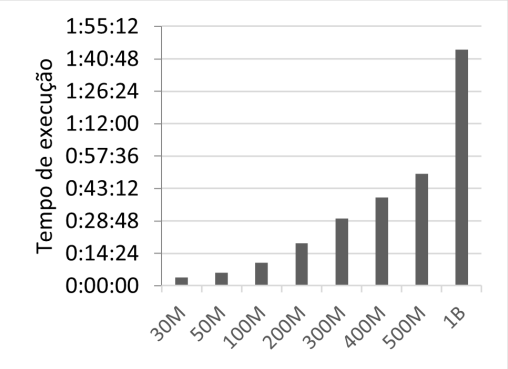

Figura 7. Execuções em 16 máquinas

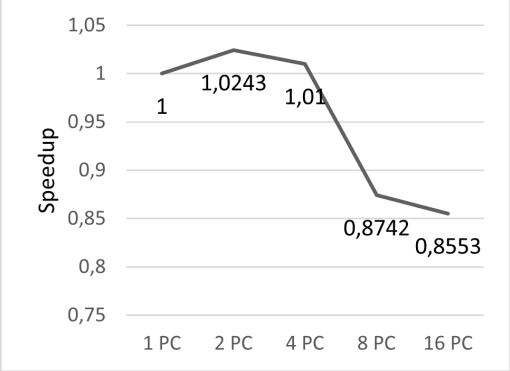

Figura 8. MapReduce Speedup Árvore 1 milhão de Vértices e Ramificação 2

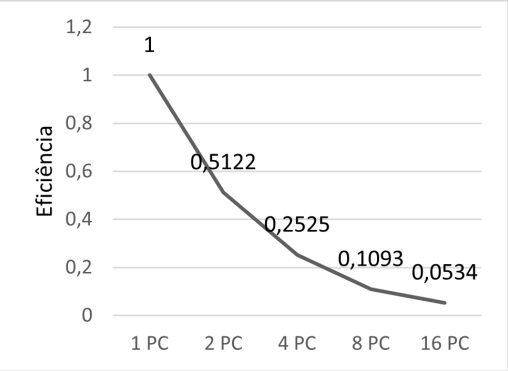

Figura 9. MapReduce Speedup Árvore 1 milhão de Vértices e Ramificação 2

grande para utilizar todos os recursos do *cluster*. A evidência disso é a impossibilidade de executar a árvore com 100 milhões de vértices com apenas 2 computadores.

Na figura 7, as árvores variaram de 30 milhões a 1 bilhão de vértices e foram executadas com 16 computadores. Este teste teve como objetivo verificar a capacidade do algoritmo de processar árvores com bilhões de vértices. Com a infraestrutura utilizada, foi possível executar o algoritmo em árvores com até 1 bilhão de vértices, com um tempo

de execução de 1 hora e 44 minutos.

Analisando as imagens 8 e 9, é possível concluir que houve uma utilização eficiente dos recursos disponíveis ao usar 1, 2 e 4 computadores para processar a árvore de 1 milhão de vértices. No entanto, houve um desperdício de recursos computacionais ao usar 8 e 16 computadores, uma vez que o Hadoop não alocou todo o recurso disponível para processamento, resultando apenas na alocação do necessário. Além disso, o gerenciamento dos computadores adicionais no *cluster* enquanto estavam ociosos causou perda de *Speedup* e Eficiência quando o número de computadores ultrapassou 8.

7. Conclusão

Este trabalho apresentou um algoritmo desenvolvido para resolver o desafio do *Lowest Common Ancestor* (LCA), junto com os tempos de execução obtidos por meio de uma infraestrutura de *cluster*. Apesar de seu caráter exploratório inicial, é evidente que a aplicação do Modelo *MapReduce* na abordagem de problemas relacionados a grafos de grande porte se mostra uma alternativa promissora.

É relevante mencionar que, de acordo com as pesquisas conduzidas por [Banhos Filho and Yero 2014] e [Banhos Filho and Yero 2016], o modelo *Bulk Synchronous Parallel* (BSP) demonstrou um desempenho superior na solução do problema de centralidade em grafos de grande escala. Nesse sentido, a próxima etapa deste estudo visa a elaboração de uma solução para o problema LCA, fazendo uso do modelo BSP, e, posteriormente, comparar seus resultados com os obtidos por meio do modelo *MapReduce*.

Essa comparação entre diferentes abordagens de processamento paralelo contribuirá para uma compreensão mais aprofundada das vantagens e desafios de cada uma delas no contexto da solução de problemas em grafos de grande magnitude.

Referências

Aho, A. V., Hopcroft, J. E., and Ullman, J. D. (1973). On finding lowest common ancestors in trees. In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 253–265.

Banhos Filho, F. and Yero, E. (2014). Mapreduce vs bsp para o cálculo de medidas de centralidade em grafos grandes. In *Anais do XIII Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, pages 183–187. SBC.

Banhos Filho, F. S. and Yero, E. J. H. (2016). Exact vs. approximated diameter calculation in large graphs. In *2016 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, pages 256–263. IEEE.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Foundation, T. A. S. (2017). Hdfs high availability. `https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HDFSHighAvailabilityWithNFS.html`. Acessado em: 24 de julho de 2022.

Hortonworks (2016). Cluster planning guide. `https://docs.cloudera.com/HDPDocuments/HDP1/HDP-1.3.7/bk_cluster-planning-guide/`

content/typical-hadoop-cluster-hardware.html. Acessado em: 22 de julho de 2021.

Kumar, V., Grama, A., Gupta, A., and Karypis, G. (1994). *Introduction to parallel computing*, volume 110. Benjamin/Cummings Redwood City, CA.

Masur, R. G. and Mcintosh, S. K. (2017). Preliminary performance analysis of hadoop 3.0. 0-alpha3. In *2017 New York Scientific Data Summit (NYSDS)*, pages 1–3. IEEE.

Networks, J. (2013). Ex2200-24t-4g switch hardware guide. https://www.networkscreen.com/datasheets/1000307-en.pdf. Acessado em: 24 de abril de 2022.

Systems, C. (2011). Big data in the enterprise - network design considerations white paper. Technical report, Cisco Systems.

White, T. (2015). Hadoop: The definitive guide, storage and analysis at internet scale. *White–London: O'Reilly Media*.

Zhao, G., Bu, D., Liu, C., Li, J., Yang, J., Liu, Z., Zhao, Y., and Chen, R. (2012). Cloudlca: finding the lowest common ancestor in metagenome analysis using cloud computing. *Protein & cell*, 3(2):148–152.