Revista Brasileira de Computação Aplicada, Abril, 2019

DOI: 10.5335/rbca.v11i1.8738
Vol. 11, Nº 1, pp. 36–47
Homepage: seer.upf.br/index.php/rbca/index

ISSN 2176-6649

ARTIGO ORIGINAL

Projeto e análise de desempenho de um algoritmo iterativo para grandes grafos em um ambiente distribuído

João P. B. Nascimento[1], Daniel de O. Capanema[1] and Adriano C. M. Pereira[2]

[1]Centro Federal de Educação Tecnológica de Minas Gerais and [2]Universidade Federal de Minas Gerais

*joaopaulobn2@gmail.com; danielcapanema1@gmail.com; adrianoc@dcc.ufmg.br

Recebido: 12/10/2018. **Revisado:** 08/02/2019. **Aceito:** 15/03/2019.

Resumo

Atualmente grandes volumes de dados são gerados e coletados por meio de sensores, dispositivos e redes sociais. A capacidade de lidar com grandes massas de dados tornou-se um importante fator para o sucesso de muitas organizações, exigindo, cada vez mais, a utilização de processamento paralelo e distribuído. Para auxiliar os desenvolvedores a projetar programas distribuídos, existem várias ferramentas (*frameworks*), como Apache Hadoop e Spark. Esses *frameworks* fornecem diversos parâmetros de configuração (por exemplo, o Hadoop tem mais de 200) e atribuir valores otimizados a todos eles não é uma tarefa simples. Este trabalho investiga a influência desses parâmetros no desempenho do Apache Hadoop, utilizando o algoritmo HEDA, um algoritmo iterativo que calcula métricas de centralidade em grandes grafos. A execução do HEDA em uma rede complexa é extremamente importante, pois existem várias medidas de centralidade que determinam a importância de um vértice dentro do grafo. Observou-se que, em alguns casos, a melhoria no tempo de execução atingiu aproximadamente 80% aplicando os valores propostos por este trabalho aos parâmetros de configuração do Hadoop. Além disso, foi possível aumentar em cinco vezes o uso dos processadores e melhorar consideravelmente a escalabilidade. O trabalho também apresenta os métodos aplicados para preparar, executar e analisar os experimentos, o que poderá auxiliar em novos estudos.

Palavras-Chave: Hadoop, Grafo, Parâmetros, Algoritmo Iterativo.

Abstract

Currently, large volumes of data are generated and collected through sensors, devices, and social networks. The ability to handle large masses of data has become an important factor for the success of many organizations, increasingly requiring the use of parallel and distributed processing. To help developers design distributed programs, there are a number of tools (frameworks), such as Apache Hadoop and Spark. These frameworks provide various configuration parameters (for example, Hadoop has more than 200) and assigning optimized values to all of them is no trivial task. This work investigates the influence of these parameters on Apache Hadoop performance, using the HEDA algorithm, an iterative algorithm that calculates centrality metrics in large graphs. The execution of HEDA in a complex network is extremely important because there are several measures of centrality that determine the importance of a vertex within the graph. It was observed that in some cases the improvement in execution time reached approximately 80 % applying the values proposed by this work to the Hadoop configuration parameters. In addition, it was possible to increase processor utilization by five times and greatly improve scalability. The work also presents the methods applied to prepare, execute and analyze the experiments, which may aid in further studies.

Key words: Hadoop, Graph, Parameters, Iterative Algorithm.

1 Introdução

Atualmente os nossos dispositivos são capazes de gerar dados em uma variedade e velocidade nunca antes vistas. Essa grande massa de dados surge a partir de diversas fontes, como vídeos, e-mails, postagens em redes sociais, notícias, músicas, sensores, experimentos científicos e muitas outras. Devido a essa diversidade de fontes temos um grande volume de dados produzidos e processá-los utilizando nossas ferramentas tradicionais está se tornando uma tarefa inviável. A computação paralela baseada em arquiteturas *multicore* ressurgiu nos últimos anos como uma opção para a solução desse problema, contudo ela requer habilidades e conhecimentos específicos que os atuais desenvolvedores podem não ter. Isso inclui a criação, sincronização e gerenciamento de *threads*, *locks*, concorrência e mecanismos de tolerância a falhas (Asanovic et al.; 2009).

Para auxiliar os desenvolvedores a projetarem algoritmos paralelos e distribuídos e ao mesmo tempo explorar a contínua evolução dos processadores, que aumentam o número de núcleos, existem diversas ferramentas distribuídas, como o Apache Hadoop[1]. O *framework* Hadoop possui mais de 200 parâmetros de configuração e atribuir valores de maneira otimizada para eles não é uma tarefa trivial, devido à grande quantidade de combinações que podem ser formadas (Chen et al.; 2010). Customizar os parâmetros de configuração é uma tarefa que requer bons conhecimentos de cada um deles, pois a mudança do valor de um parâmetro pode impactar em outros, uma vez que eles são interconectados entre si (Mathiya and Desai; 2015). Diversos estudos da literatura sugerem valores para esses parâmetros, contudo existe uma lacuna para recomendações de melhores práticas para configuração de *frameworks* distribuídos (Li et al.; 2014) (Garvit et al.; 2014) (Guo and Fox; 2012) (Krishna et al.; 2014).

O algoritmo HEDA (Nascimento and Murta; 2012) (*Hadoop-based Exact Diameter Algorithm*) é um algoritmo iterativo para MapReduce/Hadoop, baseado no algoritmo para grafos Busca em Largura. O HEDA calcula os menores caminhos de todos os nós para todos os outros nós de um grafo, as excentricidades de todos os nós e o raio e diâmetro de um grafo grande e direcionado. Todos esses cálculos são exatos e essas métricas podem auxiliar em diversos problemas do mundo real, tais como: redes sociais, roteamento de veículos, redes de distribuição, etc. O objetivo dessas métricas é apontar a importância relativa de um nó dentro de todo o grafo, informação esta que tem diversas aplicações importantes em um cenário real.

O objetivo desse estudo é analisar a influência de alguns parâmetros de configuração na eficiência do Hadoop utilizando o algoritmo iterativo HEDA. Para isso iremos executar diversos experimentos com variações de valores dos parâmetros, utilizando três conjuntos de dados (*dataset*) em *clusters* de diferentes tamanhos. Para alcançar esse objetivo nós monitoramos e avaliamos algumas métricas de desempenho do *cluster* e do algoritmo, tais como: tempo de execução, consumo de recursos do sistema (CPU, Memória RAM e Tráfego de Rede) e a escalabilidade.

Os resultados dos experimentos executados mostram que, com os valores sugeridos aos parâmetros de configuração do Hadoop, conseguimos uma redução de aproximadamente 80% no tempo de execução do algoritmo HEDA. Além disso, foi possível aumentar o consumo de CPU em cinco vezes, além de alcançarmos um Speedup de 92% da linearidade.

O restante do trabalho está organização da seguinte forma: a Seção 2 apresenta os trabalhos relacionados, a Seção 3 a metodologia adotada para projetar, configurar, executar e analisar os experimentos. Na Seção 4 são apresentados os resultados dos experimentos utilizando o algoritmo HEDA em um problema real. Por último, na Seção 5 são apresentadas as conclusões e trabalhos futuros seguidas pela Referências.

2 Trabalhos Relacionados

Diversos trabalhos presentes na literatura lidam com a tarefa de recomendar valores otimizados para os parâmetros de configuração presentes no *framework* Hadoop. Esse tipo de recomendação é uma tarefa difícil, uma vez que o Hadoop possui mais de 200 parâmetros que, quando ajustados, podem ter impacto direto no tempo de execução e no consumo de recursos (CPU, memória RAM, I/O e tráfego de rede) do sistema. Além disso, outros fatores tais como o algoritmo, o tamanho e configuração do *cluster* e a base de dados podem impactar na escolha de valores para esses parâmetros.

Uma compilação de diversos estudos sobre *frameworks* que utilizam parâmetros de configuração para melhorar o desempenho do Hadoop foi apresentado no trabalho de Mathiya and Desai (2015). Os autores identificaram vários parâmetros e apontaram aqueles que são mais relevantes, dividindo-os em três grupos, chamados CPU, Memória e I/O. Os autores concluíram que o ajuste correto dos parâmetros de desempenho tem impacto direto no tempo de execução da aplicação e que cada tipo de problema pode exigir um conjunto diferente de valores e parâmetros específicos.

Outros estudos sugerem conjuntos específicos de parâmetros de configuração que envolvem *I/O*. Os autores Shafer and Rixner (2010) e Krishna et al. (2014) avaliaram o sistema de arquivos distribuído de seus *frameworks* utilizando vários conjuntos de dados, com vários formatos diferentes. Foram utilizados também diversos valores para os parâmetros de configuração. Os experimentos foram realizados com arquivos de diferentes granularidades e, ao final, os autores concluíram que os parâmetros usados e referentes ao sistema de arquivos distribuído têm grande impacto no tempo de processamento, especialmente na quantidade de operações de *I/O*.

Os autores Nghiem and Figueira (2016) criaram um modelo matemático e um algoritmo para determinar a quantidade ótima de tarefas para serem usadas nas fases Map e Reduce do Hadoop, com o objetivo de melhorar o consumo de recursos e energia, usando um *cluster* heterogêneo, o modelo MapReduce e diversas cargas de dados para executar o *benchmark*

[1]Apache Hadoop - http://www.hadoop.apache.org

Terasort[2].

Alguns trabalhos presentes na literatura executam o ajuste de valores de parâmetros de configuração do Hadoop baseado nos resultados de execução de um *job* MapReduce anterior (Garvit et al.; 2014). Nesse trabalho, utilizamos três algoritmos iterativos que instanciam diversas vezes o conjunto de funções Map e Reduce e, por meio dos resultados da execução da instância anterior, o sistema automaticamente atribui novos valores para os parâmetros de execução de uma nova instância. O trabalho alcançou bons resultados no tempo de execução para os três algoritmos utilizados e também apresentou os ajustes de parâmetros que foram feitos, onde os mais relevantes foram o tamanho do bloco de dados do HDFS (*dfs.blocksize*) e parâmetros específicos que influenciam no desempenho da fase *Copy* do Hadoop.

Outros parâmetros de configuração do Hadoop que podem trazer grandes benefícios no tempo de execução são os que tratam da compressão dos dados que são trocados entre as fases do modelo MapReduce. O trabalho de Chen et al. (2010) executou experimentos específicos com parâmetros dessa categoria, variando as configurações de *cluster* e as bases de dados utilizadas, analisando os impactos de cada experimento no tempo de execução e no consumo de energia. Ao final os autores concluíram que, ao realizar os ajustes de parâmetros, foi possível reduzir em mais de 60% o consumo de energia do *cluster*.

O trabalho de Wang et al. (2012) criou um otimizador de parâmetros de configuração para o Hadoop chamado Predator. Esse otimizador estima o tempo de execução de um *job* MapReduce e, por meio de uma função objetivo, ordena os parâmetros do Hadoop dentro de diferentes grupos para diminuir a busca por melhores valores. Além disso, o algoritmo de otimização usa o conceito de subdivisão de espaço para evitar problemas de locais ótimos e também reduzir o tempo de busca, assim reduzindo o custo de visitas a pontos não promissores do espaço de busca. Os resultados dos experimentos utilizando o otimizador apresentaram melhoria acima de 80% no tempo de execução, comparado com o Hadoop em sua configuração padrão.

Outro relevante trabalho é o de Bei et al. (2016) que criou o RFHOC uma ferramenta que automaticamente tenta ajustar os parâmetros de configuração do Hadoop utilizando o método *Random Forest* (Breiman; 2001).

Os autores Kumar et al. (2016) usaram um algoritmo *Noisy Gradient* para otimizar valores para os parâmetros de configuração e alcançaram melhorias no tempo de execução próximas a 66% quando comparados com a configuração padrão do Hadoop.

Todos os trabalhos apresentados nesta seção lidam com a atribuição de valores para os parâmetros do Hadoop, porém sem adotar uma metodologia específica para esse fim. Este artigo se difere dos demais estudos por definir uma metodologia para realizar tal tarefa. Para validar essa metodologia, foram executados mais de 1500 horas de experimentos com diversas configurações de *cluster*, bases de dados e valores para os parâmetros de configuração, usando um algoritmo iterativo projetado para o ambiente Hadoop/MapReduce.

Este artigo encontrou uma importante lacuna na literatura e, diante disso, suas principais contribuições são: (i) criação de um modelo matemático para a recomendação de valores para alguns parâmetros da ferramenta Hadoop; (ii) validação do modelo proposto, por meio de uma extensa bateria de experimentos, certificando a sua eficiência; (iii) análise detalhada dos resultados, comparando tempo de execução, consumo de recursos do *cluster* (CPU, memória e rede), além de um estudo dos efeitos da aplicação do modelo proposto na escalabilidade da ferramenta; (iv) demonstração dos riscos da seleção inadequada de valores para os parâmetros de configuração.

3 Metodologia

Nessa seção descreveremos os passos seguidos para a preparação e execução dos nossos experimentos. Esses passos são: definição da arquitetura do sistema, seleção do algoritmo a ser utilizado, configuração do ambiente onde os experimentos serão realizados, definição dos conjuntos de dados (*dataset*), seleção dos valores e parâmetros de configuração, definição da forma como os recursos serão monitorados, maneira como os experimentos serão executados e como a análise dos resultados será feita.

3.1 Arquitetura do Sistema

As necessidades por novos paradigmas e técnicas de processamento eficientes surgem quando as técnicas convencionais, como o processamento sequencial, não se mostram efetivas em cenários onde os dados a serem processados aumentam rapidamente de tamanho.

A computação distribuída é uma das soluções para o processamento de grandes quantidades de dados, contudo ela requer uma camada de software para gerenciá-la e lidar com problemas que não ocorrem ou são menos importantes na programação sequencial, tais como conflitos e tolerância a falhas. Uma boa estratégia para não deixar para o usuário a tarefa de implementar essa estrutura de gerenciamento é a utilização de *frameworks*.

O *framework* utilizado nesse estudo é o Apache Hadoop que é um projeto *open source* baseado no modelo MapReduce e fornece escalabilidade e confiabilidade para processamento de grandes conjuntos de dados. O Hadoop foi projetado para escalar através de centenas de máquinas fornecendo processamento e armazenamento distribuído. Mais do que apenas gerenciar o processamento, o Hadoop garante disponibilidade através de um avançado sistema de tolerância a falhas (White; 2009). O MapReduce é um modelo de programação criado pelos engenheiros do Google em 2004 (Dean and Ghemawat; 2008).

3.2 Projeto e Implementação do Algoritmo

O algoritmo escolhido para a execução dos experimentos no ambiente distribuído do Hadoop

[2]Benchmark Terasort – https://mapr.com/resources/terasort-benchmark-comparison-yarn/

com diferentes valores para os parâmetros de configuração é o HEDA (*Hadoop-based Exact Diameter Algorithm*), um algoritmo iterativo, baseado no Apache Hadoop, o qual calcula de maneira exata os menores caminhos entre todos os pares de vértices, a Excentricidade de cada vértice, o Diâmetro e o Raio do grafo.

O algoritmo é dividido em três fases. A primeira tem como objetivo calcular o menor caminho entre todos os pares de vértices. Para isso, são implementadas uma função Map e uma função Reduce. Na segunda fase do algoritmo (*EncontrarExcentricidades*) são calculadas as medidas de excentricidade de cada um dos vértices. Por último, na terceira fase, são calculados o diâmetro e o raio do grafo, e as demais medidas de centralidade. Todas as fases são compostas por uma função Map e uma função Reduce.

O fluxo principal do algoritmo HEDA recebe como parâmetros de entrada o endereço do arquivo de arestas, o arquivo de distâncias e o endereço no HDFS para gravação dos dados de saída. Inicialmente, o arquivo de arestas é carregado em um vetor, cujo índice i indica o vértice de origem da aresta e o conteúdo da posição i do vetor armazena uma lista dos vértices de destino da aresta. A seguir, o algoritmo realiza um laço para processar todos os vértices do grafo. Ao finalizar esse laço, será chamada uma função para o cálculo das métricas de centralidade.

A execução do HEDA em uma rede complexa é extremamente importante, uma vez que existem diversas medidas de centralidade que determinam a importância de um vértice dentro de um grafo. A centralidade de um vértice ou a identificação de quais vértices são mais centrais que outros são problemas chaves na análise de redes complexas, especialmente aquelas que modelam as redes sociais. O algoritmo HEDA é dividido em iterações onde as primeiras iterações são relacionadas ao cálculo dos menores caminhos, tendo cada uma delas uma fase Map e outra Reduce. As iterações seguintes são usadas para encontrar as medidas de centralidade, que são: Excentricidade, Raio e Diâmetro. O Algoritmo 1 apresenta as iterações do HEDA.

Na teoria de grafos e na análise de redes complexas, existem diversas medidas de centralidade que determinam a importância de um vértice dentro do grafo. A centralidade de um vértice, ou a identificação de quais vértices são mais centrais que outros, é um problema chave na análise de redes complexas (Opsahl et al.; 2010). Como exemplo de medidas de centralidade podemos citar a excentricidade de um vértice, o diâmetro e o raio de um grafo (Fred Buckley; 1990).

Para identificar o diâmetro de um grafo é necessário primeiramente descobrir a excentricidade de cada vértice. A excentricidade de um vértice v em um grafo G, denotada como $e(v)$, é a distância de v para o vértice mais distante de v, utilizando o menor caminho (Gross and Yellen; 1999), (Fred Buckley; 1990). Assim, $e(v) = max(d(v, x)) \; \forall \, x \in V$.

Descoberta a excentricidade de todos os vértices do grafo G, podemos denotar o diâmetro $D(G)$ como o maior valor dentre as excentricidades dos vértices contidos no grafo G ou, de forma equivalente, a distância máxima entre dois vértices no grafo G (Gross and Yellen; 1999). Assim, $D(G) = max(e(x)) \; \forall \, x \in V$.

O raio de um grafo G, denotado por $R(G)$, é o menor valor dentre as excentricidades dos vértices contidos no grafo G (Gross and Yellen; 1999). Assim, $R(G) = min(e(x)), \; \forall \, x \in V$.

Um vértice central em um grafo G é o vértice que tem a menor excentricidade, tal que $e(v) = R(G)$. O centro de um grafo G, denotado por $Z(G)$ é o subgrafo formado pelo conjunto de vértices centrais do grafo G (Gross and Yellen; 1999). A periferia de um grafo é o oposto ao centro. Um vértice periférico v de um grafo G é um vértice com maior excentricidade, tal que $e(v) = D(G)$. A periferia de um grafo G, denotado como P(G) é o subgrafo formado pelo conjunto de vértices periféricos (Gross and Yellen; 1999). O grau $g(v)$ de um vértice v em um grafo G é o número de arestas de G que incidem sobre v (Bondy; 1976).

Algoritmo 1: Algoritmo HEDA - Fluxo Principal

```
Entrada: Arquivo de Arestas, Arquivo de Distâncias,
         Endereço de saída no HDFS
Saída: Excentricidade de cada nó, Diâmetro e Raio do
       Grafo
iteração ← 1;
Enquanto PossuiVerticesAProcessar() faça
  EncontrarMenorCaminho(vetorArestas,
  arquivoDistancias);
    iteração ← iteração + 1;
fim Enquanto
EncontrarExcentricidades();
EncontrarDiametroRaio();
```

Grande parte dos sistemas que são usados em nosso dia-a-dia são partes de grandes redes dinâmicas, chamadas redes complexas, tais como redes sociais (p. ex., Twitter e Facebook) e redes físicas (p. ex., energia elétrica e Internet). Essas redes podem ser modeladas por grafos com importantes características e encontrá-las é a principal tarefa do algoritmo HEDA.

3.3 Ambiente Experimental

Os experimentos deste trabalho foram executados em um *cluster* homogêneo composto por dez nós, sendo um nó designado para ser o mestre (Namenode), oito nós escravos (Datanodes) e um nó para monitorar e executar a coleta dos dados dos experimentos para posteriores análises. Todos os nós são gerenciados pela ferramenta OpenStack[3] e possuem o sistema operacional Linux Ubuntu 14.04.3LTS, com quatro processadores Intel Westmere (Intel 64 Family 6 Model 44 Stepping 1) com 2.5Ghz, duas threads, 8GB de memória RAM e 80GB de HD. Além disso, nós utilizamos dois volumes em um *storage* com 400GB para os nós mestre e escravos e 1.75TB para o nó que monitora o *cluster*. Os nós mestre e escravos possuem o Hadoop em sua versão 2.7.1.

[3]OpenStack - http://www.openstack.org

3.4 Conjunto de Dados

Para a avaliação da seleção dos parâmetros de configuração do Hadoop com o algoritmo HEDA, nós escolhemos dois conjuntos de dados que representam uma rede complexa. Um deles, chamado IRL, é um grafo da topologia dos Sistemas Autônomos da Internet, coletado pelo Internet Research Lab da Universidade da Califórnia (UCLA)[4]. Esse grafo possui 42.089 vértices e 570.570 arestas. O segundo conjunto de dados, chamado Twitter é um grafo da rede social Twitter, coletado pela Stanford Network Analysis Plataform[5], com 81.306 nós e 1.768.149 vértices. A escolha dessas bases de dados deve-se às diferentes características que elas possuem entre si. A base do Twitter é mais densa que a IRL, permitindo uma análise mais compreensiva.

Finalizado os testes com as bases IRL e Twitter para a seleção dos melhores valores dos parâmetros de configuração, nós iremos utilizar uma terceira base de dados em formato de grafo para validar as equações que foram criadas nos testes anteriores e que são sugeridas por este trabalho. A terceira base de dados escolhida foi o grafo da rede Epinions, a qual representa as relações de confiança entre consumidores do site Epinions.com, coletado pela Stanford Network Analysis Plataform, e que contém 75.879 vértices e 508.837 arestas.

3.5 Parâmetros de Configuração

Para tornar mais simples a organização e o entendimento dos nossos experimentos, criamos quatro categorias para rotulá-los. Essas categorias são: Padrão, HDFS, MapReduce e Memória. Os valores de parâmetros com melhores resultados em uma categoria de experimentos são usados em conjunto com outros parâmetros na próxima categoria. A escolha dos parâmetros de configuração que utilizamos no trabalho foi baseada na literatura, na documentação do *framework* Hadoop, no conhecimento técnico dos autores e também incrementando e decrementando os valores a partir da configuração padrão até o limite onde as execuções não retornavam em constantes erros. Por exemplo, o parâmetro *mapreduce.map.sort.spill.percent* foi testado com valores partindo do padrão, 0.8, e variando em 0.05 para cima e para baixo, e parando quando os valores retornavam constantes erros.

Para a categoria Padrão, nós utilizamos o *framework* Apache Hadoop em sua configuração original, sem qualquer alteração nos valores de seus parâmetros de configuração. Para essa categoria foram utilizadas as bases de dados Twitter e IRL para o *cluster* configurado com números de nós: {1, 2, 4 e 8} , totalizando 8 experimentos. Essa categoria será a base de comparação para as outras e permitirá avaliar a efetividade das parametrizações realizadas.

Na categoria HDFS, nós utilizamos apenas a base de dados Twitter e alteramos os parâmetros *dfs.blocksize*: {64, 128 e 256MB} e o parâmetro *dfs.replication*: {1 e 3}. Nessa categoria nós utilizamos o *cluster* com quantidade de nós: {1, 2, 4 e 8}, totalizando 24 experimentos. O parâmetro *dfs.blocksize* refere-se ao tamanho de cada bloco depois que o dado é particionado e o *dfs.replication* é o número de cópias do bloco de dados que é gerado e armazenado em nós diferentes do *cluster*.

Para a Categoria MapReduce nós utilizamos os valores de parâmetros dos experimentos que apresentaram os melhores tempos de execução na categoria HDFS combinados com os parâmetros *mapreduce.job.map*: {1, 2, 5, 6, 8, 16, 19, 32, 48 e 64} e *mapreduce.job.reduce*: {1, 2, 5, 6, 8, 16, 19, 32, 48 e 64}. Ainda na categoria MapReduce, foram utilizados também os parâmetros *mapred.tasktracker.map.tasks.maximum*: {2, 4, 6, 8, 12 e 16} e *mapred.tasktracker.reduce.tasks.maximum*: {2, 4, 6, 8, 12 e 16}. Os parâmetros *mapreduce.job.map* e *mapreduce.job.reduce* referem-se respectivamente ao número de tarefas Map e Reduce por *job* e os parâmetros *mapred.tasktracker.reduce.tasks.maximum* e *mapred.tasktracker.map.tasks.maximum* definem o número máximo de tarefas Map e Reduce que irão executar simultaneamente em um mesmo *tasktracker*.

Os experimentos da Categoria MapReduce foram executados para a base de dados Twitter e com o *cluster* com número de máquinas: {1, 2, 4 e 8}, totalizando aproximadamente 600 experimentos. Foi tentado realizar os experimentos dessa categoria também para a base de dados IRL, porém durante a execução foram encontrados alguns erros de estouro de memória, o que nos motivou a criar a quarta categoria de parâmetros denominada Memória.

A Categoria Memória avaliará os parâmetros *mapred.child.java.opts*: {-Xmx200, -Xmx400, -Xmx500, -Xmx600, -Xmx800, -Xmx1600 e -Xmx2400}, *mapreduce.task.io.sort.mb*: {150, 200, 300, 400, 800 e 1200}, e *mapreduce.map.sort.spill.percent*: {0.70, 0.75, 0.80, 0.85 e 0.90}. Esses parâmetros correspondem respectivamente ao tamanho do *heap* da JVM (Java Virtual Machine) para as tarefas Map e Reduce, a quantidade de memória usada para ordenar os arquivos (em MB) e, do total de memória disponível, qual será o percentual utilizado para armazenar as saídas da fase Map. Essa categoria foi avaliada usando as duas bases de dados (IRL e Twitter) e utilizando o *cluster* configurado com número de nós: {1, 2, 4 e 8}, totalizando aproximadamente 500 experimentos.

3.6 Monitoramento de Desempenho

As ferramentas para monitoramento são usadas para rastrear o uso de recursos e o desempenho dos sistemas. Sugere-se a escolha de uma ferramenta de monitoramento robusta, uma vez que, realizar esse monitoramento em ambientes sofisticados como *clusters*, é uma tarefa que demanda grande esforço (Fatema et al.; 2014).

Nesse trabalho nós usamos o Ganglia (Massie et al.; 2004) que é um sistema de monitoramento distribuído, escalável e projetado para sistemas de alto desempenho, tais como *grids* e *clusters*. A ferramenta é *open source* e foi desenvolvido na Universidade da Califórnia (UCLA).

[4]UCLA – Internet Research Lab. http://irl.cs.ucla.edu/index.html
[5]Stanford SNAP. http://snap.stanford.edu/

3.7 Execução dos Experimentos

Para a execução dos experimentos, iniciamos pela arquitetura do sistema distribuído (Hadoop). O *framework* foi instalado no ambiente de execução (*cluster*) e ficou disponível para comunicação com o algoritmo e os conjuntos de dados. O sistema de monitoramento também ficou disponível para coletar os dados de execução e gerar as métricas. Os experimentos que planejamos exigem que o algoritmo seja executado diversas vezes, devido a variação dos parâmetros de configuração, base de dados e configuração do *cluster*. Por isso, foi importante planejar e criar uma estrutura que automatizasse os testes por meio de *scripts*.

Para agilizar o trabalho de execução dos experimentos, uma rotina automatizada foi criada, onde os valores a serem usados eram lidos de arquivos XML e os resultados salvos em documentos de texto. Esse procedimento possibilitou a execução de mais de 1100 experimentos, totalizando mais de 1500 horas de execução.

Esse capítulo de metodologia apresentou o passo-a-passo do planejamento dos experimentos para avaliar os parâmetros de configuração do Hadoop utilizando o algoritmo HEDA.

4 Resultados

A última fase da metodologia trata da análise dos resultados dos experimentos que foram realizados. Para essa análise serão utilizados os dados que foram coletados pelo monitoramento, avaliando todas as métricas propostas. Será analisado o impacto dos valores de parâmetros de configuração propostos no tempo de execução, o consumo de recursos (CPU e memória e rede), *Speedup* e Escalabilidade.

4.1 Análise de Resultados

Após a execução dos experimentos e análise dos resultados, podemos dizer que os valores atribuídos para os parâmetros de configuração do Hadoop foram bastante efetivos na redução do tempo de execução, mas essa atribuição precisa ser feita corretamente, pois em alguns casos uma escolha incorreta de valores pode levar a erros de execução ou podem levar a resultados piores aos conseguidos na configuração padrão da ferramenta.

Os experimentos com melhores tempos de execução foram analisados e observamos que os parâmetros escolhidos tiveram grande influência no desempenho em termos de tempo de execução. Além disso, percebemos que a quantidade de recursos do *cluster*, tais como memória e CPU, são diretamente relacionados aos melhores valores encontrados para cada configuração, assim, a parametrização do ambiente deve ser proporcional à quantidade destes recursos. A partir dessa constatação, foi possível criar um modelo matemático baseado em todos os experimentos que foram executados nas duas bases de dados (IRL e Twitter) e em todos as configurações de *cluster* (1, 2, 4 e 8 nós), obtendo equações que sugerem valores para os parâmetros de configuração.

Na Categoria HDFS realizamos testes que envolveram parâmetros de configuração relacionados ao particionamento e à replicação dos dados entre os *datanodes*. Para todas as bases de dados e configurações de *cluster*, os melhores valores obtidos foram: 64MB para o parâmetro *dfs.blocksize* e o valor 3 para o parâmetro *dfs.replication*. A melhoria no tempo de execução alcançada com a alteração dos parâmetros dessa categoria ficaram próximos aos 12% quando comparado com a Categoria Padrão, em todos as configurações de *cluster*, conforme apresentado na Tabela 1. O valor de 64MB para o tamanho do bloco é menor que o indicado em alguns trabalhos presentes na literatura, entretanto no caso de algoritmos iterativos onde a base de dados aumenta conforme as iterações avançam, é melhor dividir mais os dados. Diante disso, essa será a configuração utilizada nas categorias posteriores.

Tabela 1: Categoria HDFS - Melhoria em relação a Categoria Padrão - Grafo Twitter

Melhorias obtidas na Categoria HDFS			
1 Nó	2 Nós	4 Nós	8 Nós
8,01 %	9,86 %	11,49 %	11,63 %

Tabela 2: Categoria MAPREDUCE - Melhoria em relação a Categoria Padrão - Grafo Twitter

Melhorias obtidas na Categoria MAPREDUCE			
1 Nó	2 Nós	4 Nós	8 Nós
15,34 %	42,66 %	58,98 %	64,25 %

Os parâmetros da Categoria MapReduce estão relacionados diretamente com o trabalho a ser realizado pelo *cluster*, pois influi diretamente no comportamento do modelo MapReduce. Várias combinações foram realizadas e os parâmetros *mapred.tasktracker.map[reduce].tasks.maximum* não apresentaram impactos relevantes no tempo de execução e, portanto, podem ser mantidos com seus valores padrão fornecidos pelo *framework* Hadoop.

Os parâmetros *mapreduce.job.map* e *mapreduce.job.reduce* foram extremamente significantes no tempo de execução e estão diretamente relacionados à quantidade de núcleos presentes no *cluster*, assim foi encontrada a proporção adequada e recomendado que eles sejam definidos utilizando as Equações 1 e 2. As melhorias alcançadas no tempo de execução são apresentadas na Tabela 2. Podemos observar que essa categoria e a utilização das equações propostas contribuíram para que a melhoria do tempo de execução atingisse aproximadamente 65%, para o *cluster* configurado com 8 nós.

$$mapreduce.job.map = TotalNucleosCluster \times 1.5 \quad (1)$$

$$mapreduce.job.reduce = TotalNucleosCluster \times 0.6 \quad (2)$$

Embora tenhamos alcançado bons resultados na Categoria MapReduce, durante os experimentos o conjunto de dados IRL apresentou erros de *HeapSize* e foi necessário criarmos a categoria de parâmetros Memória, para lidar com esse problema e também para avaliar outros parâmetros. Na Categoria Memória, três novos parâmetros de

configuração foram testados em conjuntos com os parâmetros das categorias anteriores. Os parâmetros *mapreduce.task.io.sort.mb* e *mapreduce.map.spill.percent* não apresentaram redução significante no tempo de execução e, devido a isso, tiveram seus valores padrões mantidos.

O parâmetro *mapred.child.java.opts* resolveu o problema de *HeapSize*, ocorrido na categoria anterior e, além disso, trouxe melhoria no tempo de execução da aplicação. Após todos os experimentos, observamos uma forte relação deste parâmetro com a quantidade de núcleos e memória disponível no *cluster* e, após encontrar a proporção adequada, sugere-se configurar esse parâmetro conforme descrito na Equação 3. A melhoria no tempo de execução proporcionada pela Categoria Memória pode ser observada na Tabela 3.

$$mapred.child.java.opts = \frac{TotalNucleosCluster}{TotalMemoria(GB)} \times 0.83 \quad (3)$$

Tabela 3: Categoria Memória - Melhoria do tempo de execução em relação a Categoria Padrão - Grafos Twitter e IRL

Melhorias obtidas na categoria Memória				
Base	1 Nó	2 Nós	4 Nós	8 Nós
Twitter	31,58 %	61,21 %	61,24 %	58,19 %
IRL	31,63 %	54,80 %	63,41 %	74,88 %

Os parâmetros de configuração que foram considerados irrelevantes em nossos experimentos podem tornar-se relevantes quando combinados com outros parâmetros em outras classes de algoritmos. As recomendações de valores para os parâmetros feitos nessa seção são específicas para problemas que envolvem algoritmos iterativos de busca em grafos.

4.2 Melhoria no Tempo de Execução

Os valores utilizados para cada parâmetro de configuração, em cada configuração de *cluster* utilizando as Equações 1, 2 e 3 são apresentados pela Tabela 4. A melhoria menos significativa foi de 35,14% na base de dados Twitter utilizando uma máquina e os resultados foram melhorando a medida que mais máquinas foram sendo adicionadas, alcançando 79,10% para a base de dados IRL com o *cluster* configurado com 8 nós.

A comparação do tempo de execução do grafo IRL é apresentada na Figura 1 e a do grafo Twitter na Figura 2. É possível observar que o tempo de execução para as duas bases são muito diferentes. O grafo IRL é maior e assim leva um longo tempo

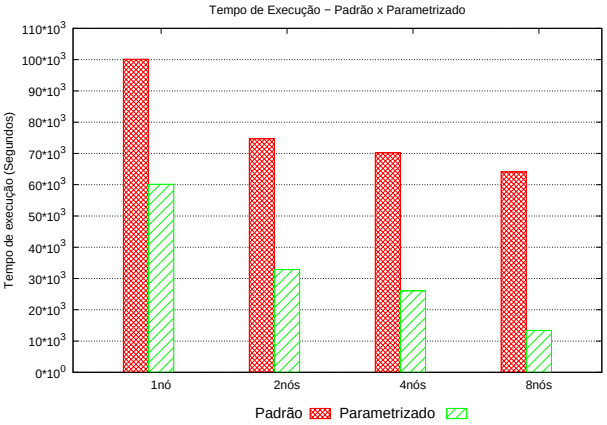

Figura 1: Tempo de Execução - Grafo IRL

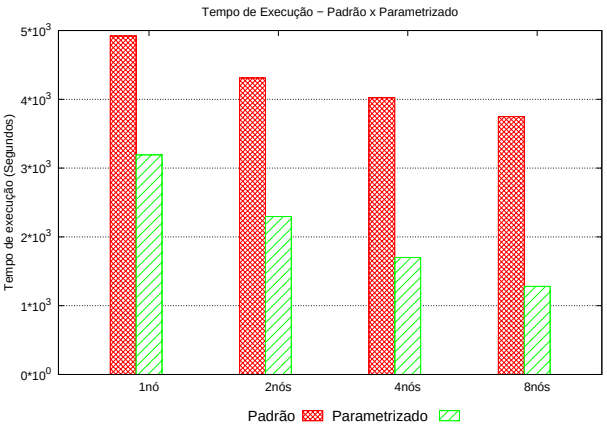

Figura 2: Tempo de Execução - Grafo Twitter

para executar, porém a melhoria alcançada com a parametrização é bastante similar em ambas as bases de dados, apresentando maiores taxas de melhoria a medida que mais máquinas são adicionadas ao *cluster*.

4.3 Escalabilidade

A escalabilidade é um atributo desejado em um sistema, processo ou rede. Esse conceito está relacionado à habilidade de um sistema em acomodar o aumento da quantidade de recursos e utilizá-los adequadamente a medida que a carga de trabalho também aumenta (Bondi; 2000).

Para calcular a escalabilidade do algoritmo HEDA no ambiente Hadoop nós utilizamos a métrica Speedup. O Speedup é definido como a taxa de tempo gasto quando executamos o programa com um simples processador pelo tempo de execução gasto

Tabela 4: Configuração dos parâmetros propostos

	Configuração dos parâmetros					**Melhoria no tempo**	
	Categoria HDFS		**Categoria MAPREDUCE**		**Categoria Memória**		
Nós	**blocksize**	**replication**	**job.map**	**job.reduce**	**mapred.child.java.opts**	**Twitter**	**IRL**
1	64MB	3	6	2	-Xmx400m	**35.14%**	**39.93%**
2	64MB	3	12	5	-Xmx400m	**46.70%**	**56.00%**
4	64MB	3	24	10	-Xmx400m	**57.71%**	**62.86%**
8	64MB	3	48	19	-Xmx400m	**65.80%**	**79.10%**

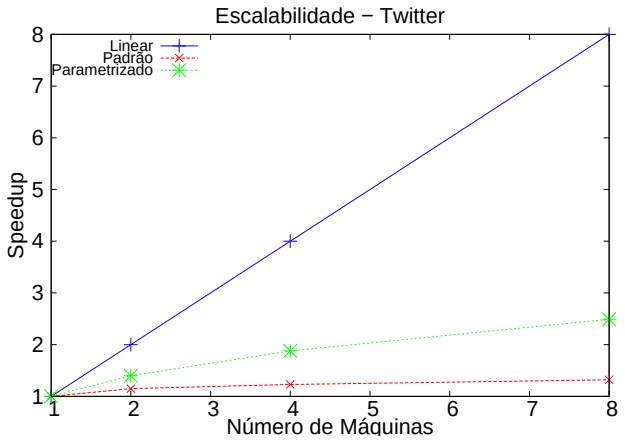

Figura 3: Escalabilidade- Grafo Twitter

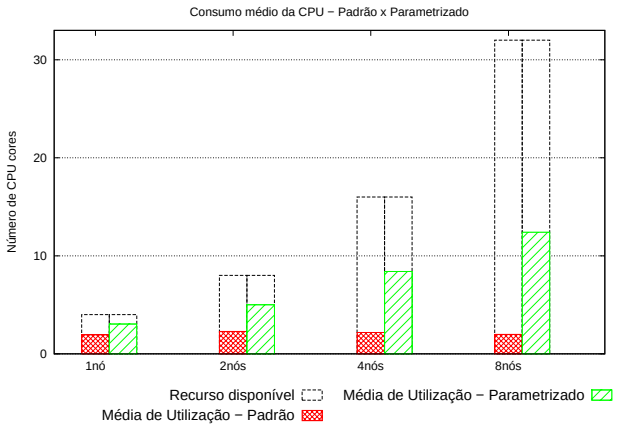

Figura 5: Consumo de CPU - Grafo Twitter

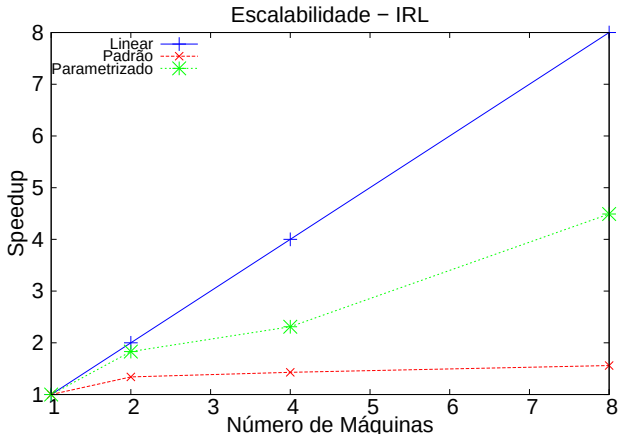

Figura 4: Escalabilidade- Grafo IRL

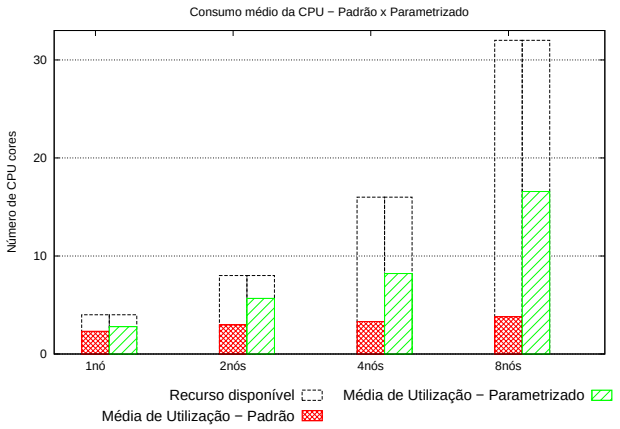

Figura 6: Consumo de CPU - Grafo IRL

com *n* processadores (Eager et al.; 1989).

A Figura 3 apresenta a escalabilidade do algoritmo HEDA para o grafo Twitter considerando as categorias Padrão e Parametrizada. No caso de dois nós, o Speedup foi de aproximadamente 70% da linearidade para os experimentos parametrizados. O grafo IRL apresentou Speedup de 92% da linearidade para dois nós e 57% da linearidade para oito nós, conforme apresentado na Figura 4.

Para todas as bases de dados (Twitter e IRL), os resultados de escalabilidade são apresentados com o ambiente computacional em seu estado padrão e parametrizado. Além disso é apresentado também os dados referentes à linearidade, para efeito de comparação.

4.4 Consumo de Recursos

Durante os experimentos, o consumo de recursos em cada máquina foi monitorado e armazenado utilizando a ferramenta Ganglia. A Figura 5 e a Figura 6 apresentam respectivamente para os grafos Twitter e IRL a média de utilização de CPU durante o processamento.

O comportamento do consumo de CPU para os dois grafos foi bastante similar, tendo o grafo IRL apresentado consumo um pouco maior que o Twitter. Os gráficos mostram que ao utilizarmos o Hadoop com os parâmetros de configuração sem ajustes (Categoria Padrão), o consumo médio de CPU praticamente não se alterou para ambos os grafos, ou seja, independentemente do número de CPU's disponíveis, o consumo não passou de 3. Ao atribuirmos novos valores para os parâmetros de configuração utilizando as Equações 1, 2 e 3, o consumo de CPU foi aumentado, principalmente quando aumentamos a quantidade de nós do *cluster*.

Outro recurso que medimos foi o consumo de memória RAM do *cluster*. A Figura 7 apresenta o consumo de memória do grafo Twitter e a Figura 8 apresenta o consumo de memória para o grafo IRL. Ambos os gráficos destacam a quantidade total de memória RAM disponível, a média de consumo de memória Cache, a média de consumo sem ajuste nos parâmetros e a média de consumo com os parâmetros ajustados.

Durante os experimentos percebemos que o Hadoop usa a memória *Cache* frequentemente, porém essa característica não é usada durante as iterações do algoritmo, pois a cada iteração os dados precisam ser lidos e gravados no sistema de arquivos distribuídos da ferramenta (HDFS) o que causa aumento no tempo de execução da aplicação.

A média de Mpbs trafegados entre os nós do *cluster* são apresentados na Figura 9 para o grafo Twitter e Figura 10 para o grafo IRL. Nos gráficos

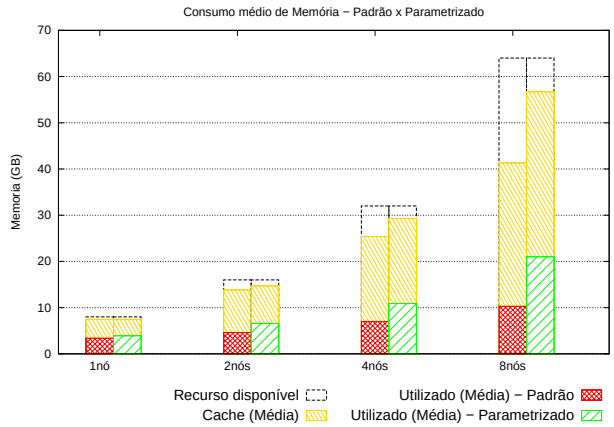

Figura 7: Consumo de Memória - Grafo IRL

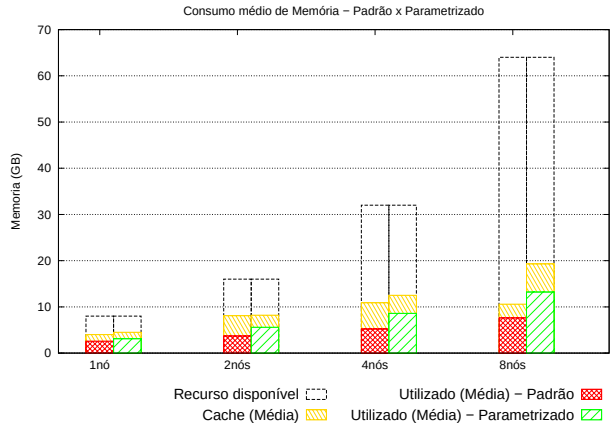

Figura 8: Consumo de Memória - Grafo IRL

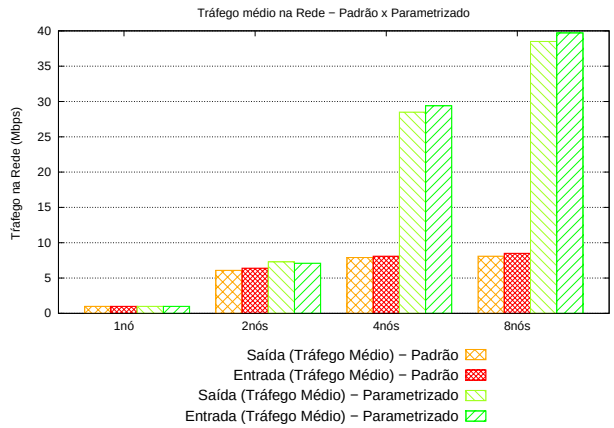

Figura 9: Tráfego de Rede - Grafo Twitter

são apresentados os dados de entrada e saída, com o Hadoop em sua configuração padrão e parametrizada.

O tráfego de rede para os dois conjuntos de dados (IRL e Twitter) foram consideravelmente altos quando aplicada a parametrização, especialmente para o *cluster* com maiores números de nós. Esse fato é esperado devido a grande quantidade de nós na rede e, consequentemente, o aumento da comunicação entre eles. O oposto ocorreu com o *cluster* configurado

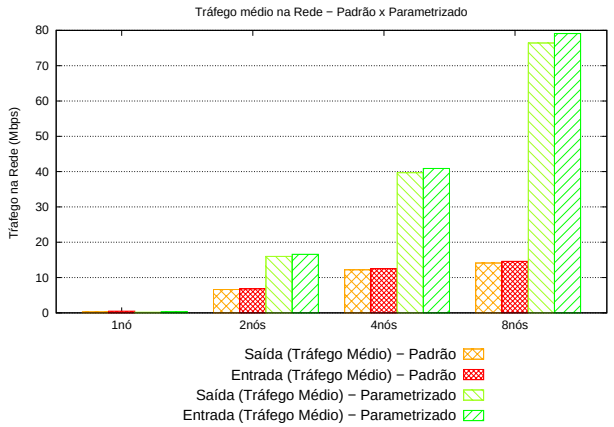

Figura 10: Tráfego de Rede - Grafo IRL

com apenas um nó, o que é esperado, pois não houve troca de informações entre nós. Embora o ajuste de parâmetros aumente o tráfego da rede e essa situação seja desfavorável, a maior média encontrada foi 80Mbps para o *cluster* com 8 nós executando a base de dados IRL. Esse valor não compromete o trabalho, uma vez que as redes atuais suportam facilmente tráfegos médios muito maiores.

4.5 Riscos da Parametrização

A atribuição de valores para os parâmetros do Hadoop pode ser amplamente efetiva, porém uma escolha equivocada desses valores pode resultar em erros de execução, conforme ocorreu em nossa Categoria MapReduce para o grafo IRL ou pode resultar também em uma piora no tempo de execução quando comparado à configuração padrão do ambiente. A Figura 11 apresenta todos os tempos obtidos durante todos os experimentos utilizando o conjunto de dados Twitter. As cores que estão no eixo X destacam cada categoria de parâmetros utilizados nos experimentos. É importante observarmos que, mesmo atribuindo novos valores para os parâmetros em algumas situações o tempo de execução foi pior que a configuração padrão dos parâmetros do Hadoop, mostrando que é necessário muita atenção ao atribuir esses valores.

4.6 Validação do Modelo

Nessa seção iremos validar as Equações 1, 2 e 3 por meio de novos experimentos com um novo conjunto de dados denominado grafo Epinions.

Primeiramente foram definidos os valores dos parâmetros de configuração que serão usados nesse novo experimento. Para isso foram usadas as Equações 1, 2 e 3. Os experimentos foram executados com o *cluster* configurado com 1, 2, 4 e 8 nós. Utilizamos o Hadoop em sua configuração padrão e parametrizado com os novos valores. Com isso, realizamos 8 novos experimentos.

A melhoria alcançada no tempo de execução com o grafo Epinions foi muito próxima daquela alcançada nos experimentos com os grafos IRL e Twitter, o que demonstra a utilidade do modelo proposto. A Figura 12 apresenta os tempos de

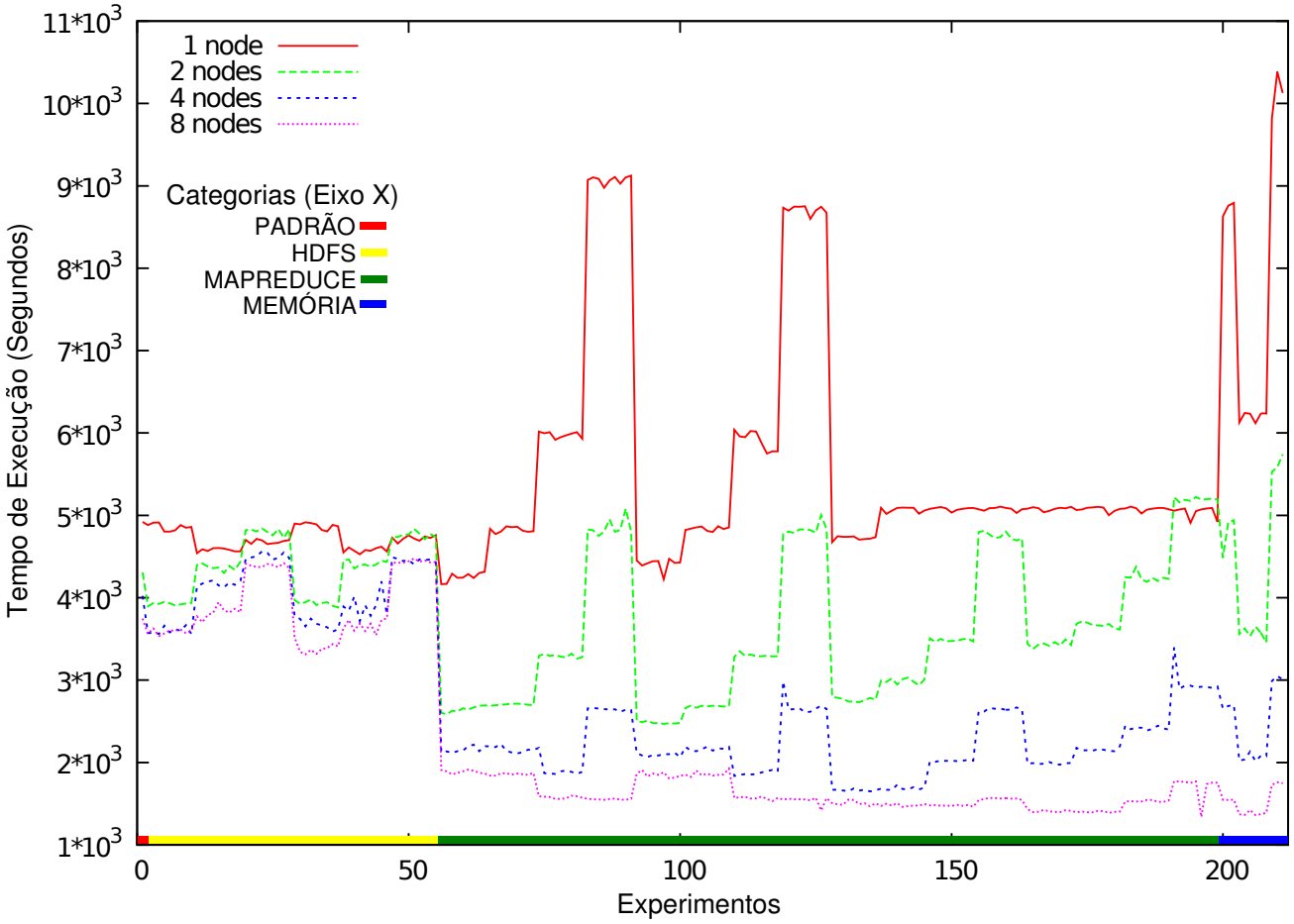

Figura 11: Tempos em todas as categorias de experimentos – Grafo Twitter

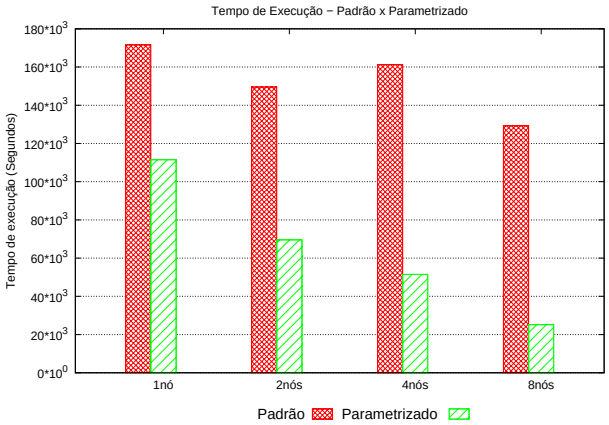

Figura 12: Tempo de Execução. Validação do modelo. Grafo Epinions.

execução dos experimentos com a configuração padrão comparada com os experimentos realizados com o ambiente parametrizado, usando o modelo matemático proposto.

Um ponto importante a ser observado é que ao aumentarmos o número de nós no *cluster* com o ambiente em sua configuração padrão, o tempo de execução não melhora muito e, em algumas situações, chega até a piorar, como foi o caso do *cluster* configurado com dois e quatro nós. Essa melhora não acontece devido ao fato do consumo de recursos (CPU e memória RAM) continuarem o mesmo na configuração padrão, mesmo quando aumentamos o número de nós. A subutilização do *cluster* com a configuração padrão também aconteceu com as bases de dados IRL e Twitter.

A configuração dos parâmetros de acordo com o modelo matemático proposto por esse trabalho permitiu que os recursos do *cluster* fossem melhor aproveitados, aumentando os percentuais de uso de recursos e diminuindo o tempo de execução, a medida que mais nós foram adicionados. Com isso alcançamos mais de 80% de melhoria no tempo de execução para o *cluster* configurado com 8 nós.

As Figuras 13 e 14 apresentam respectivamente o consumo médio de CPU e de memória RAM alcançados durante os experimentos com o grafo Epinions. É possível observar que tanto a CPU quanto a memória RAM ficaram subutilizados com a configuração padrão do Hadoop e o consumo desses recursos foi aumentado com a parametrização, especialmente nas configurações de *cluster* com mais nós.

A Figura 15 apresenta a média de tráfego na rede durante os experimentos com o grafo Epinions. O tráfego, embora aumentado com a parametrização, ainda não compromete o tempo de execução do experimento, uma vez que ainda é considerado baixo diante do total suportado pela rede. Na execução com a configuração padrão e *cluster* com oito nós, o tráfego na rede foi bastante baixo, menor que o

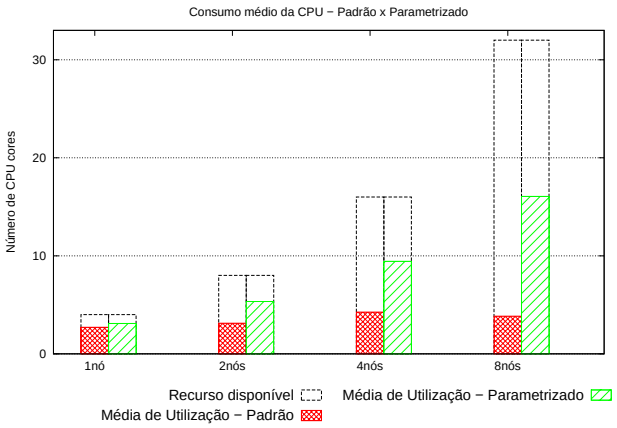

Figura 13: Uso médio de CPU. Validação do modelo. Grafo Epinions.

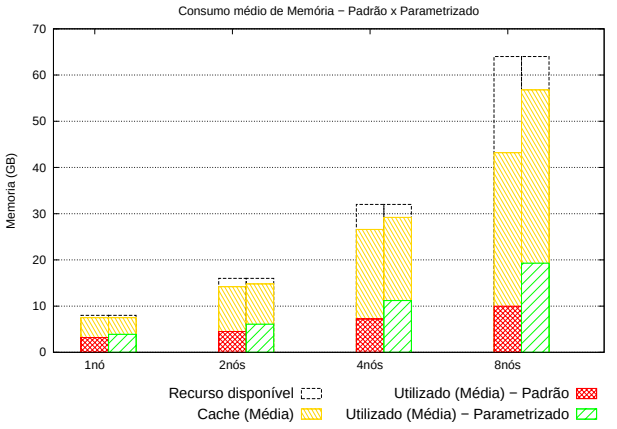

Figura 14: Uso médio de memória. Validação do modelo. Grafo Epinions.

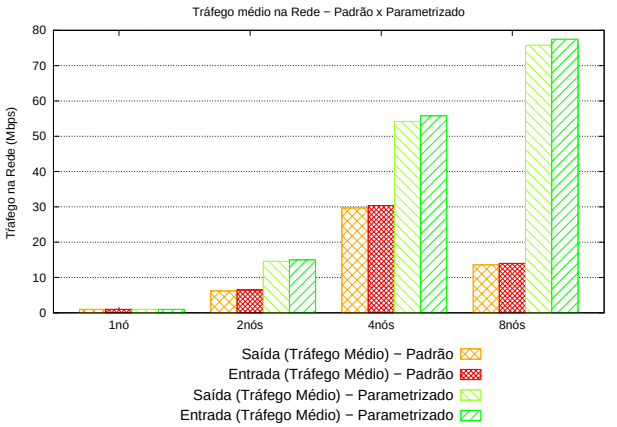

Figura 15: Tráfego de rede. Validação do modelo. Grafo Epinions.

experimento com quatro nós. Isso se deve ao fato da baixa utilização do *cluster*, onde algumas máquinas simplesmente não são usadas, gerando baixo tráfego.

A Figura 16 apresenta a escalabilidade do HEDA usando o grafo Epinions. Podemos notar que o Hadoop em sua configuração padrão não

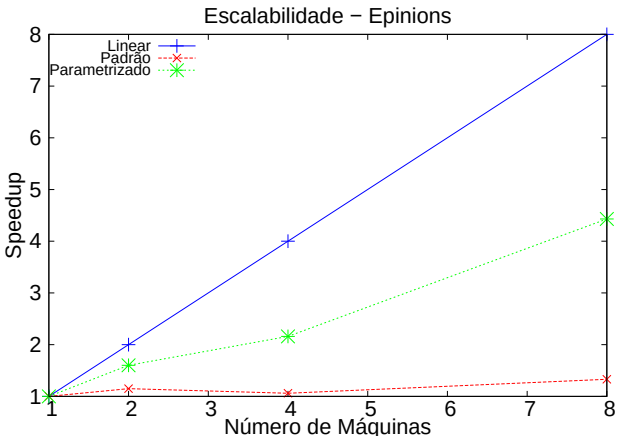

Figura 16: Escalabilidade. Validação do modelo. Grafo Epinions.

apresenta uma boa escalabilidade. Por outro lado, quando ajustamos os parâmetros de configuração do Hadoop usando o modelo proposto nesse trabalho, alcançamos uma melhoria considerável na escalabilidade, o que indica a utilidade e importância do modelo proposto.

5 Conclusão

Esse artigo apresentou um estudo para avaliar o impacto da mudança dos valores dos parâmetros de configuração do Hadoop, usando um algoritmo iterativo para calcular medidas de centralidade em grafos grandes. Atribuir os valores corretos para os parâmetros de configuração do Hadoop não é uma tarefa trivial, devido ao grande número de combinações que podem ser feitas com o grande número de parâmetros que o Hadoop possui. O Hadoop em sua configuração padrão pode não apresentar bons resultados de tempo de execução, devido a subutilização dos recursos do *cluster*, como memória, CPU e rede. O Hadoop também não apresenta escalabilidade satisfatória executando algoritmos iterativos em sua configuração padrão.

Executamos diversos experimentos para avaliar alguns parâmetros de configuração do Hadoop. Esses parâmetros estão relacionados ao sistema de armazenamento distribuído (HDFS), ao modelo MapReduce, ao núcleo do *framework* e à memória da JVM. Foram criadas quatro categorias de parâmetros: Padrão, HDFS, MapReduce e Memória. Com os melhores experimentos de cada categoria foi possível criar um modelo matemático que define quais valores deverão ser utilizados para cada parâmetro estudado. Esse modelo será útil para os avanços nos estudos dos parâmetros de configuração da ferramenta Hadoop. Nós validamos o modelo matemático criado e alcançamos resultados satisfatórios em tempo de execução e consumo de recursos.

Os objetivos desse trabalho foram alcançados, ao fornecer as seguintes contribuições: (i) criação de um modelo matemático para recomendação de valores para parâmetros de configuração do Hadoop. (ii) validação desse modelo com dados de grafos reais. (iii) foi possível mostrar que os parâmetros de configuração do Hadoop têm grande influência no

tempo de execução, uso de recursos (CPU, memória e rede) e escalabilidade do algoritmo iterativo HEDA, destacando que foi possível reduzir o tempo de execução em aproximadamente 80% em um dos experimentos realizados. Como trabalho futuro pretendemos realizar mais experimentos com novas categorias de parâmetros de configuração, para ampliar o modelo de recomendação de valores.

Agradecimentos

Este trabalho foi realizado com suporte das agências: Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Fundação de Amparo à Pesquisa de Minas Gerais (FAPEMIG) e Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Referências

Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D. and Yelick, K. (2009). A view of the parallel computing landscape, Vol. 52, ACM, New York, NY, USA, pp. 56–67.

Bei, Z., Yu, Z., Zhang, H., Xiong, W., Xu, C., Eeckhout, L. and Feng, S. (2016). Rfhoc: A random-forest approach to auto-tuning hadoop's configuration, Vol. 27, IEEE Computer Society, Los Alamitos, CA, USA, pp. 1470–1483.

Bondi, A. B. (2000). Characteristics of scalability and their impact on performance, *Proceedings of the 2nd International Workshop on Software and Performance*, ACM, New York, NY, USA, pp. 195–203.

Bondy, J. A. (1976). *Graph Theory With Applications*, Elsevier Science Ltd., Oxford, UK, UK.

Breiman, L. (2001). Random forests, Vol. 45, Kluwer Academic Publishers, Hingham, MA, USA, pp. 5–32.

Chen, Y., Ganapathi, A. and Katz, R. H. (2010). To compress or not to compress - compute vs. io tradeoffs for mapreduce energy efficiency, *Proc. of the First ACM SIGCOMM Workshop on Green Networking*, ACM, New York, NY, USA, pp. 23–28.

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

Eager, D. L., Zahorjan, J. and Lozowska, E. D. (1989). Speedup versus efficiency in parallel systems, Vol. 38, IEEE Computer Society, Washington, DC, USA, pp. 408–423.

Fatema, K., Emeakaroha, V. C., Healy, P. D., Morrison, J. P. and Lynn, T. (2014). A survey of cloud monitoring tools: Taxonomy, capabilities and objectives, *J. Parallel Distrib. Comput.* **74**(10): 2918–2933.

Fred Buckley, F. H. (1990). *Distance in Graphs*, The Advanced Book Program, 1st edn, Addison-Wesley Pub. Co.

Garvit, B., Anshul, G., Utkarsh, P., Manish, S. and Subhasis, B. (2014). A framework for performance analysis and tuning in hadoop based clusters, *Smarter Planet and Big Data Analytics Workshop (SPBDA 2014), held in conjunction with International Conference on Distributed Computing and Networking (ICDCN 2014)*, Coimbatore, INDIA, pp. 49–62.

Gross, J. and Yellen, J. (1999). *Graph theory and its applications*, CRC Press, Inc., Boca Raton, FL, USA.

Guo, Z. and Fox, G. (2012). Improving mapreduce performance in heterogeneous network environments and resource utilization, *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, IEEE Computer Society, Washington, DC, USA, pp. 714–716.

Krishna, T. L. S. R., Ragunathan, T. and Battula, S. K. (2014). Performance evaluation of read and write operations in hadoop distributed file system, *Sixth International Symposium on Parallel Architectures, Algorithms and Programming, PAAP 2014, Beijing, China, July 13-15, 2014*, pp. 110–113.

Kumar, S., Padakandla, S., Lakshminarayanan, C., Parihar, P., Gopinath, K. and Bhatnagar, S. (2016). Performance tuning of hadoop mapreduce: A noisy gradient approach, *10th IEEE International Conference on Cloud Computing (CLOUD)*, Honolulu, HI, USA, pp. 375–382.

Li, C., Zhuang, H., Lu, K., Sun, M., Zhou, J., Dai, D. and Zhou, X. (2014). An adaptive auto-configuration tool for hadoop, *Engineering of Complex Computer Systems (ICECCS), 2014 19th International Conference on*, pp. 69–72.

Massie, M., Chun, B. and Culler, D. (2004). The ganglia distributed monitoring system: design, implementation, and experience, *Parallel Computing* **30**(5-6): 817–840.

Mathiya, B. J. and Desai, V. L. (2015). Apache hadoop yarn parameter configuration challenges and optimization, *Soft-Computing and Networks Security (ICSNS), 2015 International Conference on*, pp. 1–6.

Nascimento, J. P. and Murta, C. (2012). Um algoritmo paralelo em hadoop para cálculo de centralidade em grafos grandes, *XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pp. 393–406.

Nghiem, P. P. and Figueira, S. M. (2016). Towards efficient resource provisioning in mapreduce, *Journal of Parallel and Distributed Computing* **95**: 29 – 41.

Opsahl, T., Agneessens, F. and Skvoretz, J. (2010). Node centrality in weighted networks: Generalizing degree and shortest paths, *Social Networks* **32**(3): 245 – 251.

Shafer, J. and Rixner, S. (2010). *A Storage Architecture for Data-intensive Computing*, PhD thesis, Houston, TX, USA. AAI3421190.

Wang, K., Lin, X. and Tang, W. (2012). Predator – an experience guided configuration optimizer for hadoop mapreduce., *CloudCom*, IEEE Computer Society, pp. 419–426.

White, T. (2009). *Hadoop: The Definitive Guide*, 1st edn, O'Reilly Media, Inc.