PROCESSAMENTO PARALELO DE IMAGENS DE PARTÍCULAS MINERAIS COM TAMANHOS DISTINTOS UTILIZANDO HADOOP E MR4C *

Felipe Schimith Batista [1]
Guilherme Lucio Abelha Mota [2]
Gilson Alexandre Ostwald Pedro da Costa [3]
Leandro Augusto Justen Marzulo [4]
Alexandre da Costa Sena [5]
Otávio da Fonseca Martins Gomes [6]

Resumo

O presente trabalho desenvolveu uma solução, utilizando Hadoop e Map Reduce for C (MR4C), capaz de paralelizar a segmentação de fissuras em imagens de partículas minerais. Um conjunto de 221 imagens de tamanhos distintos de partículas de minério de cobre obtidas por microscopia eletrônica de varredura foi utilizado para os testes. A variação do número de nós de processamento apresentou um ganho do tempo de execução que escalou de forma quase que linear. O balanceamento de carga estática promoveu um *speedup* de 1,3. A solução proposta, quando comparada à implementação sequencial, considerando um cluster com 9 nós de processamento, atingiu um *speedup* de 19,1.

Palavras-chave: Big data; Processamento paralelo; Processamento de imagens; Caracterização de minério.

PARALLEL PROCESSING OF MINERAL PARTICLE IMAGES WITH DIFFERENT SIZES USING HADOOP AND MR4C

Abstract

The present work developed a solution, using Hadoop and Map Reduce for C (MR4C), able to parallelize the segmentation of cracks in mineral particles images. A set of 221 images with different sizes of copper ore particles obtained by SEM was used for the tests. The variation in the number of processing nodes presented a speedup that increased almost linearly. Static load balancing promoted a speedup of 1.3. The proposed solution, when compared to the sequential implementation, considering a cluster with 9 processing nodes, reached a speedup of 19.1.

Keywords: Big data; Parallel processing; Image processing; Ore characterization.

[1] *Cientista da Computação, M.Sc., Prog. de Pós-grad. em Ciências Computacionais, Uerj, Brasil.*
[2] *Eng. de Sistemas e Comput., D.Sc., Prog. de Pós-grad. em Ciências Computacionais, Uerj, Brasil.*
[3] *Eng. de Computação, D.Sc., Prog. de Pós-grad. em Ciências Computacionais, Uerj, Brasil.*
[4] *Bacharel em Informática e Tecnologia da Informação, D.Sc., Google LLC, EUA.*
[5] *Bacharel em Informática, D.Sc., Prog. de Pós-grad. em Ciências Computacionais, Uerj, Brasil.*
[6] *Eng. Químico, D.Sc., CETEM e PPGEO / Museu Nacional / UFRJ, Brasil.*

** Contribuição técnica ao 23° Seminário de Automação e TI, parte integrante da ABM Week 2019, realizada de 01 a 03 de outubro de 2019, São Paulo, SP, Brasil.*

1 INTRODUÇÃO

A indústria mineral, estimulada por regulamentações sociais e ambientais cada vez mais restritivas, está se orientando para a extração de metais a partir de minérios e rejeitos com baixo teor e para o aproveito integral de jazidas minerais polimetálicas, assim como para a redução do consumo de água e energia. Deste modo, é preciso inovar nas técnicas de caracterização tecnológica de minérios a fim de desenvolver processos industriais mais eficientes e sustentáveis [1].

A caracterização tecnológica de minérios compreende um conjunto de métodos, incluindo operações de beneficiamento de minérios em escala de laboratório e diversas técnicas de análise instrumental, empregados para estudar minérios a fim de avaliar produtos e desenvolver e otimizar rotas de processo. Atualmente, o uso combinado de processamento de imagens com microscopia ótica [2-6], microscopia eletrônica de varredura [7,8] ou ambas [9,10] desempenha um papel crucial na indústria mineral, pois são esses os principais métodos para obtenção de informações essenciais sobre o minério, como associações minerais, textura, liberação e exposição.

A lixiviação em pilha e em especial a biolixiviação tem ganho importância na indústria como tecnologias de baixo custo para o processamento de minérios e rejeitos com baixo teor [11]. Tais tecnologias podem ser utilizadas no processamento de agregados e partículas grandes, reduzindo os custos relacionados à cominuição, etapa que representa frequentemente em torno de 50% do custo operacional de uma planta de processamento mineral [1]. Apesar das evidentes vantagens econômicas e ambientais, o processamento de partículas grandes apresenta uma limitação intrínseca, já que apenas os grãos na superfície destas partículas estão expostos à solução de lixiviação. Uma alternativa para aumentar a exposição dos minerais de interesse é o emprego prévio de operações de britagem a fim de gerar fissuras nas partículas para que a solução de lixiviação possa atingir grãos internos [12].

A avaliação da exposição pode ser realizada a partir de técnicas de processamento de imagens. Em [12], foram empregadas imagens, obtidas por um Microscópio Eletrônico de Varredura (MEV), de amostras de um minério sulfetado de cobre processado de diferentes formas. A rotina de aquisição de imagens obteve, para cada amostra, 1400 *frames* (2000 x 1726 pixels) com resolução espacial de 0,5 µm/pixel e sobreposição de 10% em ambas as direções X e Y. Como a maioria das partículas tem tamanho maior do que um *frame*, um algoritmo de *stitching* foi empregado para juntar os 1400 *frames*, de modo a formar uma imagem de campo estendido (imagem mosaico) com cerca de 65.000 x 65.000 pixels, totalizando mais de 4 Gpixel, para cada amostra. A partir do mosaico, cada partícula é identificada e extraída em um recorte retangular. A Figura 1a apresenta um exemplo de partícula.

A complexidade do processamento de uma amostra sofre influência do tipo de algoritmo empregado para a segmentação das fissuras nos recortes. O procedimento de segmentação de fissuras apresentado em [12] é também utilizado no presente trabalho (Figura 1b). O método é baseado no algoritmo de crescimento de regiões [13], cuja implementação original possui ordem de complexidade $O(n^3)$ [14], sendo n o tamanho da imagem.

Em termos da abordagem computacional, a implementação original de [12], realizada em Matlab [15], emprega processamento sequencial. O tempo de processamento demandado para o processamento de todas as partículas de cada amostra, da ordem de horas, tem se mostrado incapaz de fazer frente ao volume de dados e à necessidade de repetição de execuções para a determinação dos valores ótimos dos parâmetros do algoritmo. Em uma etapa preliminar da presente pesquisa, uma versão do algoritmo sequencial de segmentação foi implementada em linguagem C utilizando a biblioteca OpenCV [16], obtendo-se um *speedup* (razão entre os tempos de processamento da versão de referência e da versão em avaliação) de apenas 1,15.

Figura 2. Segmentação: (a) imagem exemplo de uma partícula; (b) resultado da segmentação de (a).

O processamento de grande volume de dados traz desafios para a área de tecnologia da informação. O presente trabalho investiga o uso de técnicas de processamento paralelo para acelerar a segmentação de fissuras em partículas minerais. O objetivo é desenvolver uma solução, utilizando Hadoop [17] e Map Reduce for C (MR4C) [18], capaz de paralelizar a segmentação de fissuras e avaliar seu ganho de desempenho. Para os testes foram utilizados o algoritmo de segmentação de [12] e um subconjunto de sua base de dados, composto por imagens de partículas (3,36 mm a 0,841 mm) de uma minério processado em um britador de mandíbula e em seguida submetido à fragmentação eletrodinâmica [19].

Em linhas gerais, o Hadoop possibilita o desenvolvimento de métodos escaláveis de processamento paralelo, capazes de armazenar e processar grande volume de dados. A linguagem nativa no Hadoop é java. Por questões de desempenho e praticidade, ao definir esta vertente da pesquisa, percebeu-se a necessidade de utilização de alguma estrutura de *software* que simplificasse a utilização de código nativo do sistema operacional como o caso do executável gerado a partir da versão C do algoritmo. Para tanto, foi utilizada a biblioteca MR4C. Esta biblioteca agrega ao Hadoop a possibilidade do uso de código nativo da plataforma básica do sistema operacional dos nós, viabilizando o uso de linguagens compiladas, como o C++, e permitindo o uso de bibliotecas como o OpenCV.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Hadoop

A biblioteca de software Apache Hadoop é um *framework* que permite o processamento distribuído de grandes conjuntos de dados em *clusters* de computadores. O Hadoop é projetado para escalar recursos de armazenamento e processamento a partir de um único servidor para potencialmente milhares de máquinas. Em vez de confiar em hardware para garantir a disponibilidade do sistema, a biblioteca em si é concebida para detectar e tratar falhas na camada de aplicação. Assim, é possível prover um serviço altamente disponível a partir de um conjunto de computadores passíveis de falhar [20].

A arquitetura do Hadoop pode ser decomposta em três camadas principais: armazenamento, computação e aplicação. A camada de armazenamento faz a gestão dos conteúdos e é representada pelo Hadoop Distributed File System (HDFS). A camada de computação faz a distribuição do processamento e é representada pelo Hadoop YARN. A camada de aplicação faz uso dos recursos de computação e armazenamento e pode ser representada pelo MapReduce.

2.2 MR4C

O Map Reduce for C (MR4C) é um *framework* baseado em C++ que foi desenvolvido utilizando o Java Native Interface. A intenção era possibilitar que um software escrito originalmente nas linguagens C ou C++ pudesse utilizar a infraestrutura do HDFS. O MR4C é um projeto *opensource* da Google [19] concebido para facilitar a grande escala de processamento de imagens de satélite e de dados geoespaciais.

O MR4C possibilita a utilização do Hadoop YARN para o gerenciamento e monitoramento de recursos. Um dos requisitos do MR4C é possibilitar a manipulação de dados do HDFS em conjunto com o poderoso ecossistema de bibliotecas de processamento de imagem desenvolvidas em C++, conforme expressa a Figura 2.

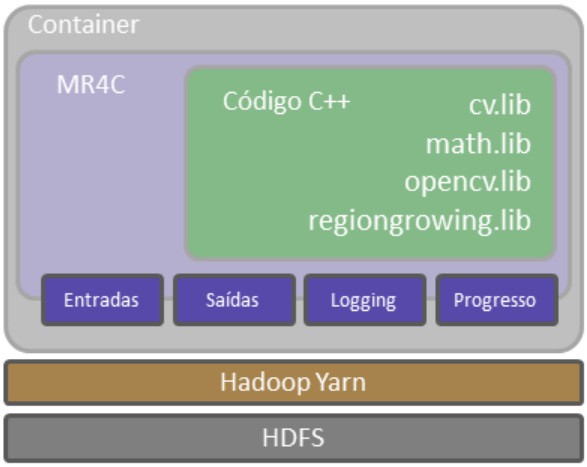

Figura 2. Arquitetura do MR4C.

A Figura 2 apresenta a arquitetura do MR4C, que utiliza os *containers* do Hadoop YARN para processar os blocos dos conteúdos localizados no HDFS. Os algoritmos necessários para o processamento de dados, representados em verde, são transformados em objetos compartilhados nativos. Assim, após serem encapsulados em *containers*, as tarefas podem ser replicadas para os nós de processamento.

O MR4C possibilita alocação de recursos do *cluster* para as tarefas da aplicação. Assim, em função do volume da entrada, do hardware disponível e das configurações usadas na instanciação, o volume de recursos é definido quando a aplicação é instanciada. São disponibilizados três parâmetros para configuração: o número de tarefas, a memória por tarefa e o número de *cores* por tarefa.

A memória por tarefa deve ser configurada para viabilizar o processamento de seu maior item de dado. O objetivo é estabelecer o valor mínimo necessário e, assim, possibilitar a instanciação de um número maior de tarefas. Por sua vez, o número de *cores*, também definido por tarefa, deve ser configurado levando em conta o tipo de algoritmo a ser utilizado. Deve-se mencionar o fato de que, na arquitetura do MR4C, os itens de dados serão distribuídos pelas tarefas no momento de sua instanciação. Além disto, não há possibilidade de redistribuir um item de dado previamente associado a alguma tarefa. Especificamente, a forma como ele faz a distribuição dos itens de dados em tarefas é, por *default*, sequencial, em ordem crescente pelo nome dos arquivos. Por sua vez, o YARN fica responsável pela definição do nó onde a tarefa será instanciada. Esta forma de distribuição dos dados pelas tarefas do MR4C pode causar inconvenientes. Em situações onde o volume de dados diverge, podem ocorrer problemas de balanceamento de carga.

É necessário destacar que os valores dos parâmetros da aplicação MR4C influenciam diretamente o desempenho da execução. Uma configuração adequada pode minimizar o tempo de processamento. Entretanto, para utilizar o *cluster* de forma ideal, os recursos dele devem ser considerados em função das características da aplicação a fim de explorar ao máximo o potencial do equipamento.

3 SOLUÇÃO PROPOSTA

Nas tentativas preliminares de segmentação de um conjunto de partículas realizadas ao longo do desenvolvimento deste trabalho, dois fatores demonstraram ter significativo impacto sobre o tempo de processamento. Por um lado, o balanceamento de carga, que visa equilibrar os volumes dos subconjuntos de imagens destinados às tarefas, minimizando o tempo ocioso dos *cores*. Por outro lado, a definição, no âmbito do MR4C, do número de tarefas, do volume de memória e do número de *cores* alocados por tarefa. Estes valores devem ser ajustados de forma a maximizar o número de tarefas passíveis de serem executadas em paralelo.

Uma das características do problema em questão corresponde à discrepância nos tamanhos das imagens das partículas. Este fato pode, em função do critério *default* de distribuição de carga do MR4C (alternância circular pelos nós do *cluster* em ordem alfabética) gerar problemas de balanceamento de carga. Além disto, é preciso enfatizar que, no MR4C, uma vez definida a tarefa para o qual uma dada imagem foi atribuída, não é possível rever tal definição. Assim sendo, resta como

possibilidade para a minimização do desnível de complexidade das tarefas, o uso *a priori* de uma abordagem estática.

A implementação realizada força a distribuição das imagens em alternância circular por ordem decrescente de tamanho, simplesmente renomeando as imagens. Desta forma, a maior imagem vai para a primeira tarefa, a segunda maior imagem para a segunda e assim por diante. Deve ser enfatizado que a proposta não abdica das estratégias de balanceamento dinâmicas realizadas naturalmente no nível do Hadoop, que, no caso, em termos conceituais, ocorrem de forma transparente.

Parâmetros relativos às tarefas definidos junto ao MR4C também possuem influência no tempo total de execução do procedimento de segmentação. São eles: o número de tarefas, o número de *cores* por tarefa e o volume de memória reservado para cada tarefa.

Na abordagem apresentada, o número de tarefas pressupõe a definição para cada nó do *cluster* do número máximo de tarefas que podem ser processadas simultaneamente. Esta escolha, por sua vez, depende do número de *cores* e do volume de memória requeridos por tarefa e está submetida ao número de *cores* e memória disponível no hardware.

A configuração ideal de memória para cada nó de processamento pode ser obtida analisando o valor mínimo necessário da memória máxima requisitada por tarefa para o processamento da maior imagem da base de dados. O valor da memória obtido deve ser multiplicado pelo número de imagens a que se deseja processar de forma paralela. O resultado deste valor deve ser disponibilizado pelo nó. Deve-se enfatizar que um limite superior de paralelismo em um nó de processamento é dado pela razão da memória disponível no nó pela memória utilizada na configuração da tarefa.

Adicionalmente, deve-se levar em conta a memória necessária para o sistema operacional. Assim, o nó de processamento deve dispor de memória múltipla do valor a ser alocado pelas tarefas paralelamente executadas, acrescida da memória mínima necessária para o sistema operacional. Outro limitador pode ser o número de *cores*, que deve ser igual ou maior ao número de tarefas a que se deseja processar em paralelo por nó de processamento, adicionado, ao menos, um *core* dedicado ao sistema operacional. Por fim, é preciso mencionar que, no caso, o número de nós de processamento deve ser o maior possível até o limite do número de imagens a serem processadas.

4 PROCEDIMENTO EXPERIMENTAL, RESULTADOS E DISCUSSÃO

Os experimentos realizados têm como objetivo avaliar a escalabilidade e ganhos de desempenho com o balanceamento e paralelismo da solução proposta no presente trabalho.

Serão utilizadas duas métricas para a avaliação dos resultados, o tempo de execução e o *speedup*, que consiste na razão entre os tempos de processamento da versão de referência e da versão em avaliação. Além disto, foi utilizado como parâmetro de comparação o *speedup* teórico para o caso linear. Nos experimentos

onde foram utilizados como referência, também foi apresentada a expressão do cálculo para a obtenção do caso linear.

O algoritmo de segmentação de fissuras com uso do *region growing* foi executado no Hadoop variando-se o número de nós de processamento. Os objetivos são a obtenção de configurações otimizadas, a avaliação da distribuição do processamento e a análise de ganhos e limitações do método desenvolvido, além de sua escalabilidade.

4.1 Infraestrutura do cluster utilizado nos experimentos

Os experimentos foram realizados em um *cluster* composto de 10 nós. Por sua vez, cada nó do *cluster* possui dois processadores Intel Xeon-E5345, com 4 *cores* cada um, frequência de *clock* de 2,33 GHz e 8 GiB de memória RAM DDR2 667 MT/s *dual channel*. Os nós do *cluster* são interconectados através de uma rede ethernet Gigabit.

No *cluster*, o Hadoop foi instalado e configurado a partir da distribuição Cloudera, versão 5.9, que incorpora a versão 2.6 do Hadoop, o MR4C e o OpenCV, juntamente com todas as dependências de cada produto mencionado. A arquitetura do *cluster* é composta por dez nós, um dos quais é reservado como nó principal de gerenciamento e os outros nove são nós de processamento.

O nó principal possui os recursos de gerenciamento do YARN, o ResourceManager, e do HDFS, o NodeManager, juntamente com o MR4C. Já os nós de processamento possuem o recurso de processamento do YARN, o NodeManager, e o recurso de armazenamento do HDFS, o DataNode.

4.2 Base de dados

A base de dados empregada nos experimentos é composta de 221 imagens com tamanho variando entre 1,8 e 76,4 Mpixel, resultando em uma média de 12,9 Mpixel por imagem e um total de 2.844,5 Mpixel. O coeficiente de variação no tamanho das imagens é de 87,9%, o que pode resultar em desbalanceamento na carga computacional.

4.3 Execução sequencial da base de dados

Este experimento visa medir o tempo de execução da versão sequencial do algoritmo de segmentação ao analisar o conjunto de imagens presentes na base de dados. No caso específico, a medida de tempo se refere à execução direta a partir do código executável derivado da implementação C do algoritmo *region growing*.

Esta implementação emprega somente um *core*, uma vez que o algoritmo sequencial não fornece meios de utilizar mais de um core, nem, tampouco, permite ajustar o volume de memória. O presente resultado corresponde à execução direta em um dos nós de processamento do *cluster*. O tempo de execução obtido foi de 17.259 s, equivalente a 4 h 47 min 39 s.

4.4 Influência do número de tarefas

O presente experimento tem como objetivo avaliar a influência do número de tarefas numa aplicação MR4C Hadoop. O conjunto de dados usados neste experimento foi concebido para reduzir a necessidade e a influência do procedimento de balanceamento de carga. Para tanto, a base de dados foi composta a partir de cópias da imagem 031 da base de dados original, cuja resolução é de aproximadamente 2,2 Mpixel. Ao considerar o número de nós, decidiu-se pela utilização de 180 cópias desta imagem, uma vez que este valor é múltiplo do número de nós disponível.

Na configuração deste experimento, para cada tarefa, foram reservados 2,5 GiB e um *core*, enquanto, manteve-se o número de nós do *cluster* fixo em nove. Para avaliar a influência do número de tarefas, nestes experimentos serão instanciadas 9, 18, 36, 45, 90 e 180 tarefas em uma única aplicação por vez e serão medidos os tempos necessários à segmentação de todo o conjunto de imagens. Estes resultados são apresentados na Figura 3. Além dos resultados produzidos pelo Hadoop, essa figura também apresenta uma curva que expressa, tomando como referência o tempo obtido para nove tarefas, os tempos hipotéticos caso o desempenho apresentado tivesse um fator de crescimento linear em função do número de tarefas.

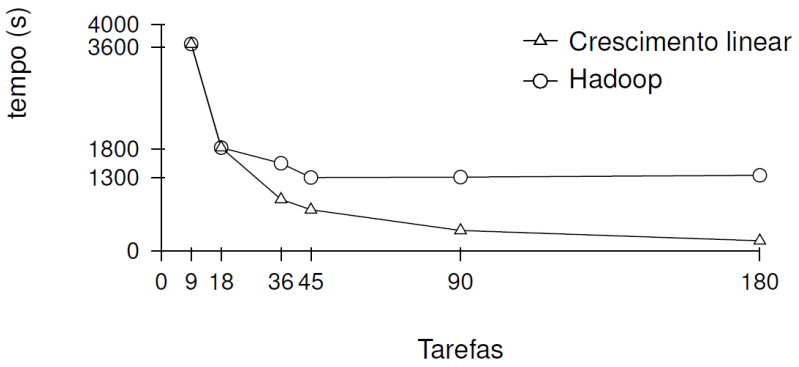

Figura 3. Tempo de execução obtido para aplicações MR4C Hadoop com distintas quantidades de tarefas concorrentes.

Analisando os tempos de execução apresentados na Figura 3, nota-se que, dentre as configurações avaliadas neste experimento, o menor tempo de processamento foi obtido para 45 tarefas. Na figura, observa-se também que a capacidade de processamento do *cluster* apresentou crescimento linear entre 9 e 18 tarefas. O desempenho continuou a apresentar melhora até o experimento com a instanciação de 45 tarefas concorrentes. Apesar disto, entre 18 e 45 tarefas o desempenho se distanciou do linear, sendo que, a partir de 90 tarefas, o resultado começa a apresentar piora.

Esse comportamento pode ser explicado pelo fato de os nós de processamento possuírem somente 8 GiB de memória RAM, sendo que os recursos do Hadoop estão configurados para utilizar até 7.5 GiB de memória, dedicando pelo menos 512 MiB para os serviços do sistema operacional. Em contrapartida, cada tarefa foi configurada com 2,5 GiB de memória, o que torna possível a execução de até três

tarefas em paralelo por nó de processamento. Assim, utilizando nove nós de processamento, é possível haver até 27 tarefas executando em paralelo.

A análise dos *logs* de execução do YARN corroborou os comportamentos esperado e observado. Ficou evidenciado que, se o número de tarefas instanciado exceder o limite de recursos de hardware, as tarefas que excederem ao limite de recursos terão que aguardar a liberação de recursos para poderem entrar em execução. Além disto, quanto mais tarefas estiverem em espera mais trocas de contexto serão necessárias para a execução da aplicação. Assim, o *overhead* das trocas de contexto fica evidenciado, sobretudo para além de 45 tarefas concorrentes.

4.5 Impacto do balanceamento de carga

A presente seção visa à avaliação do impacto do método estático de balanceamento de carga. Para tanto, foram instanciadas 45 tarefas, sendo reservados para cada uma um *core* e 3 GiB de memória em um *cluster* com 9 nós de processamento.

O experimento foi repetido por algumas vezes com e sem a utilização do método de balanceamento de carga. Os tempos médios obtidos foram 2007 s sem e 1540 s com o balanceamento de carga. Portanto, para o caso avaliado, o uso do balanceamento promoveu um *speedup* de 1,3.

4.6 Influência do número de nós de processamento

Este experimento objetiva avaliar a influência do número de nós de processamento no desempenho da execução do procedimento de segmentação no Hadoop. Para todas as configurações de quantidade de nós no *cluster*, serão executadas simultaneamente duas aplicações. A primeira aplicação utiliza 25 tarefas com 3 GiB e um *core* para segmentar as 25 maiores imagens da base de dados. Os arquivos relativos a estas imagens possuem entre 10 MiB e 30 MiB. A segunda aplicação utiliza 20 tarefas de 1,5 GiB para processar as demais imagens. Em todas as execuções, a aplicação referente ao subconjunto das menores imagens foi submetida ao procedimento de balanceamento de carga.

A justificativa para a execução simultânea de duas aplicações dedicadas ao processamento de conjuntos de imagens com tamanhos distintos se dá no fato de possibilitar uma melhor utilização dos recursos de memória do *cluster*. Para tanto, é necessário definir parâmetros de memória diferentes para conjuntos de imagens com faixas de tamanhos distintos. Os valores ótimos dos parâmetros de configuração das aplicações e tarefas MR4C foram definidos a partir de um conjunto de experimentos preliminares.

A Figura 4 apresenta o resultado obtido variando o número de nós no *cluster* de um a nove. Novamente, além dos resultados produzidos pelo Hadoop, a gráfico também apresenta uma curva que expressa os tempos hipotéticos caso o desempenho apresentado tivesse um fator de crescimento linear em função do aumento no número de nós. Tais valores tomam como referência o tempo obtido para um nó de processamento.

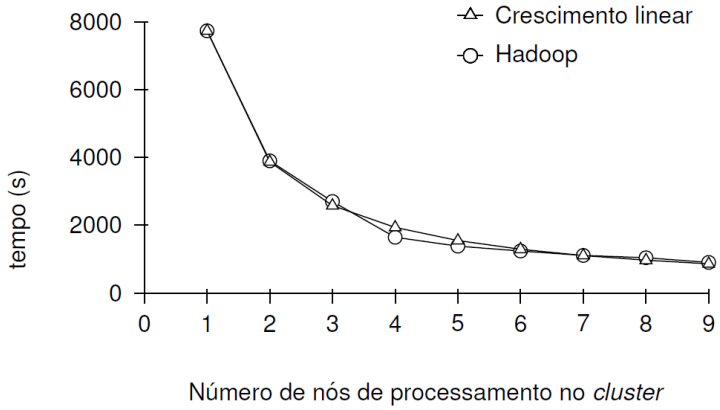

Figura 4. Tempo de execução obtido para duas aplicações MR4C Hadoop sendo executadas concorrentemente em *clusters* com quantidades distintas de nós de processamento.

Analisando os resultados apresentados na Figura 4 nota-se que o aumento do número de nós de processamento proporcionou redução no tempo de processamento da amostra. Além disto, o resultado produzido pelo Hadoop para as configurações avaliadas foi análogo ao linear. Assim sendo, apesar de possivelmente haver um limite próximo a partir do qual o desempenho do Hadoop se distancia do caso linear, é possível afirmar que, na faixa de números de nós avaliada, a proposta demonstrou boa escalabilidade.

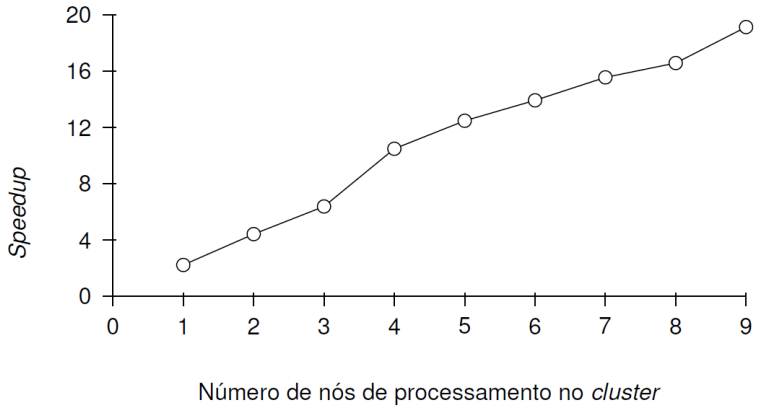

Figura 5. *Speedup* obtido para duas aplicações MR4C Hadoop sendo executadas concorrentemente em *clusters* com quantidades distintas de nós de processamento tendo como referência o resultado da implementação sequencial.

A Figura 5 apresenta os resultados de *speedup* da implementação MR4C Hadoop, usando como referência a implementação sequencial, em função números de nós de processamento no *cluster*. Com nove nós de processamento, obteve-se um *speedup* de 19,1. Este valor está abaixo do potencial estimado de capacidade de processamento no *cluster*, obtido multiplicando-se 7 *cores* por nó (permanecendo um reservado para o sistema operacional) e 9 nós, totalizando 63. Entretanto, é importante ressaltar que, por conta das limitações de memória, só foi possível instanciar 3 tarefas por nó, resultando em potencial de *speedup* de 27, portanto mais próximo do obtido na aplicação.

5 CONCLUSÃO

O presente trabalho desenvolveu uma solução, utilizando Hadoop e MR4C, capaz de paralelizar a segmentação de fissuras em imagens de partículas minerais. Nos experimentos realizados, foi possível identificar comportamentos de diversos parâmetros a partir das variações na configuração do método paralelo de segmentação de fissuras.

A variação do número de nós de processamento apresentou um ganho do tempo de execução que escalou de forma quase que linear para a faixa de valores de número de nós de processamento no *cluster* avaliada.

As configurações do ambiente e os recursos alocados para cada tarefa do MR4C influenciaram diretamente no tempo de execução do método de segmentação de fissuras. Assim, fica enfatizada a necessidade de conhecer tanto a arquitetura do *cluster* quanto a da solução do problema encapsulado no *container* MR4C.

Outro fator crítico no contexto da presente proposta corresponde ao fato do MR4C vincular definitivamente imagens às respectivas tarefas. Esta característica força desenvolvedores a buscarem estratégias estáticas para o balanceamento de carga, pois seu impacto pode ser considerável.

Os resultados demonstraram o potencial da utilização da solução definida e apresenta uma alternativa ao algoritmo proposto em [12]. Dentre as características interessantes da presente arquitetura pode ser mencionada a versatilidade, podendo esta mesma solução ser aplicada a problemas análogos. A solução se adapta sobretudo a problemas em que não haja dependência no processamento dos diversos itens de dado.

É importante ressaltar que os ganhos de desempenho computacional medidos neste estudo foram obtidos utilizando infraestrutura com baixo recurso de memória em cada nó de processamento. Ficou constatado que, neste *cluster*, a memória é um gargalo e limita a utilização dos *cores*. Apesar dessa limitação, a arquitetura apresentada visou explorar ao máximo os recursos do *cluster* e a capacidade de processamento paralelo do Hadoop.

Como trabalho futuro, os autores sugerem a adaptação do algoritmo para que este execute de forma paralela dentro de uma tarefa MR4C, permitindo que se tire proveito dos *cores* que, por conta dos limites de memória, estão ociosos.

Agradecimentos

Os autores agradecem o apoio das agências financiadoras brasileiras CNPq, Finep e FAPERJ. Os autores agradecem à FAPERJ pelo apoio financeiro ao presente projeto de pesquisa (E-26/110.124/2014). OdFM Gomes agradece ao CNPq e à FAPERJ pelo apoio financeiro, processos 311179/2014-2 e E-26/202.862/2015, respectivamente.

REFERÊNCIAS

1 Schneider CL, Matiolo E, Neumann R, Gomes OFM. Beneficiamento de minérios. In: Melfi AJ, Misi A, Campos DdA, Cordani UG, org. Recursos Minerais no Brasil: problemas e desafios. Rio de Janeiro: Academia Brasileira de Ciências; 2016. p. 256–262.

2 Donskoi E, Suthers SP, Fradd SB, Young JM, Campbell JJ, Raynlyn TD, et al. Utilization of optical image analysis and automatic texture classification for iron ore particle characterisation. Minerals Engineering. 2007; 20(5):461–471.

3 Gomes OdFM, Paciornik S, Iglesias JCA. A simple methodology for identifying hematite grains under polarized reflected light microscopy. In: Proceedings of 17th International Conference on Systems, Signals and Image Processing – IWSSIP 2010. EdUFF Editora da Universidade Federal Fluminense; 2010. p. 428–431.

4 Gomes OdFM, Iglesias JCA, Paciornik S, Vieira MB. Classification of hematite types in iron ores through circularly polarized light microscopy and image analysis. Minerals Engineering. 2013; 52:191–197.

5 Donskoi E, Poliakov A, Holmes R, Suthers S, Ware N, Manuel J, et al. Iron ore textural information is the key for prediction of downstream process performance. Minerals Engineering. 2016; 86:10–23.

6 Iglesias JCA, Augusto KS, Gomes OdFM, Domingues ALA, Vieira MB, Casagrande C, et al. Automatic characterization of iron ore by digital microscopy and image analysis. Journal of Materials Research and Technology. 2018; 7(3):376–380.

7 Sutherland DN, Gottlieb P. Application of automated quantitative mineralogy in mineral processing. Minerals Engineering. 1991; 4(7):753–762.

8 Gu Y. Automated scanning electron microscope based mineral liberation analysis. Journal of Minerals Materials Characterization Engineering. 2003; 2(1):33–41.

9 Gomes OdFM, Paciornik S. Multimodal Microscopy for Ore Characterization. In: Kazmiruk V (ed.), Scanning Electron Microscopy. Rijeka: InTech; 2012; 313–334.

10 Castellanos RM, Iglesias JCA, Gomes OdFM, Augusto KS, Paciornik S. Characterization of iron ore pellets by multimodal microscopy and image analysis. REM – International Engineering Journal. 2018; 71:209–215.

11 Watling HR. The bioleaching of sulphide minerals with emphasis on copper sulphides – A review. Hydrometallurgy. 2006; 84(1):81–108.

12 Gomes OdFM, Oliveira DMd, Sobral LGS, Pirard E. Characterization of Particle Damage and Surface Exposure of a Copper Ore Processed by Jaw Crusher, HPGR and Electro-dynamic Fragmentation. In: Characterization of Minerals, Metals, and Materials 2014. John Wiley & Sons, Ltd; 2014. p. 245–252.

13 Adams R, Bischof L. Seeded region growing. IEEE Transactions on Pattern Analysis and Machine Intelligence. 1994;16(6): 641–647.

14 Gofman E. Developing an Efficient Region Growing Engine for Image Segmentation. In: Proceedings of the International Conference on Image Processing, ICIP 2006, October 8-11, Atlanta, Georgia, USA; 2006. p. 2413–2416.

15 The Mathworks I. MATLAB version 8.5.0.197613 (R2015a). Natick, Massachusetts; 2015.

16 Bradski G, Kaehler A. Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media; 2008.

17 Bengfort B, Kim J. Analítica de Dados com Hadoop: uma introdução para cientistas de dados. São Paulo: Novatec Editora Ltda.; 2016.

18 Kennedy-Bowdoin T. MapReduce for: Run Native Code in Hadoop; 2015 [acesso em 11 out. 2016]. Disponível em: https://opensource.googleblog.com/2015/02/ mapreduce-for-c-run-native-code-in.html.

19 Andres U, Jirestig J, Timoshkin I. Liberation of minerals by high-voltage electrical pulses. Powder Technology. 1999; 104(1):37–49.

20 Apache. Welcome to Apache Hadoop; 2016. [acesso em 11 out. 2016]. Disponível em: http://hadoop.apache.org/.