Compressão de Dados em Multicores com Flink ou SPar?

Fernanda Mello, Dalvan Griebler, Isabel Manssour, Luiz Gustavo Fernandes

[1] Escola Politécnica, Grupo de Modelagem de Aplicações Paralelas (GMAP),
Pontifícia Universidade Católica do Rio Grande do Sul, PUCRS. Porto Alegre, Brasil

fernanda.mello02@edu.pucrs.br,{dalvan.griebler,isabel.manssour}@pucrs

Resumo. *Neste trabalho, foi implementada uma versão do algoritmo de compressão de dados Bzip2 com o framework para processamento de stream Apache Flink, a fim de avaliar seu desempenho em comparação com a versão do Bzip2 já existente na linguagem de domínio específica SPar. Os experimentos revelaram que a versão com SPar possui um desempenho muito superior ao Flink.*

1. Introdução

Algoritmos de compressão de dados representam aplicações de processamento de stream usadas em massa, e que podem ser eficientemente divididas em um fluxo de estágios e executadas de forma paralela [Andrade et al. 2014]. Destarte, muitas versões desses algoritmos, a exemplo do Bzip2 [Seward 2017], já foram feitas usando *frameworks* para programação paralela, a fim de analisar o desempenho destas tecnologias. Pesquisas com uma aplicação para compressão neste formato já foram realizadas usando a linguagem SPar [Griebler et al. 2017a], a qual obteve desempenho próximo ao de outros *frameworks* como Pthreads, FastFlow ou TBB [Griebler et al. 2017b, Griebler et al. 2018].

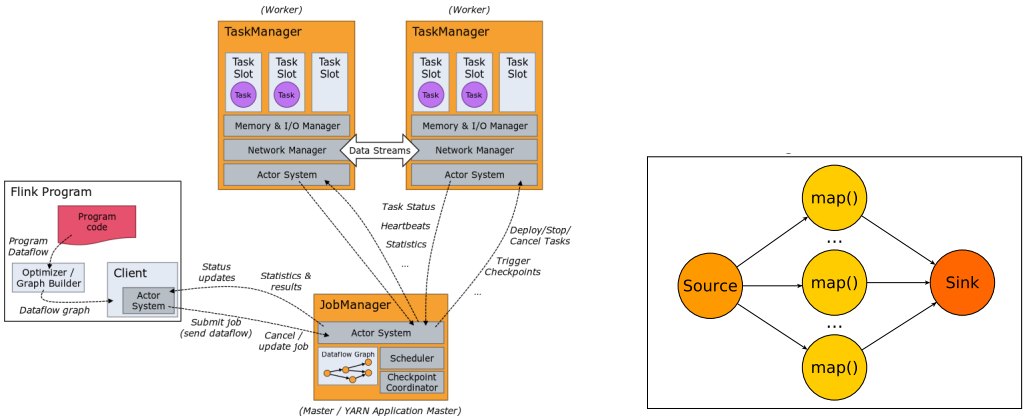
As versões do Bzip2 paralelas se encontram atualmente em *frameworks* de C++, enquanto outras ferramentas populares de processamento de stream, como Apache Flink, ainda não foram utilizadas para implementação deste algoritmo, apesar de apresentarem suporte para mais linguagens de programação e serem tecnologias populares para Big Data e processamento de dados. Acredita-se que o Flink não consegue atingir resultados tão positivos quanto aqueles destes *frameworks* para alto desempenho, mas ainda não foi feita uma comparação detalhada desta tecnologia com a SPar para aplicações de compressão de dados. Diante disso, este trabalho visa criar uma versão Flink do algoritmo Bzip2, bem como realizar uma análise comparativa do desempenho do Flink e da SPar em ambientes multicore usando suas respectivas versões deste algoritmo.

Neste artigo, primeiramente, a Seção 2 explica como é a estrutura de um programa Flink e como foi desenvolvida a aplicação de compressão Bzip2. Então a Seção 3 explica como este programa e seu equivalente em SPar foram testados, avaliados e os resultados finais desta análise. Por fim, a Seção 4 apresenta as conclusões do trabalho e possíveis passos a serem tomados no futuro.

2. Desenvolvimento

O Apache Flink é uma ferramenta para processamento de stream que implementa o padrão paralelo *pipeline* [Andrade et al. 2014]. Como pode ser visto na Figura 1a, o Flink atua como um sistema distribuído, sendo que o processo mestre, chamado de *Job Manager*, coordena a execução e agenda as tarefas a serem executadas em *slots*. Estes *slots* pertencem aos processos chamados *Task Managers*. Estes as executam na Máquina Virtual Java (JVM), de acordo com o seu número de *slots* disponível e o nível de paralelismo especificado [Deshpande 2017], de modo que ele empregue o paralelismo de stream.

Uma das principais APIs do Flink, DataStream API, possibilita o processamento de streams por meio de abstrações de alto nível e, por isso, foi utilizada para o desenvolvimento da aplicação de compressão. Todo programa DataStream deve obter, em primeiro



(a) Arquitetura do Flink [Foundation 2019] **(b) Estratégia de Paralelismo**

Figura 1. Detalhes da ferramenta Flink

lugar, um ambiente de execução, e a execução deste deve ser disparada para que as operações Flink sejam realizadas. Em segundo lugar, os dados devem ser carregados em uma coleção de dados chamada DataStream a partir de uma *Source*, isto é, a fonte de onde chegam os dados de entrada. Em seguida, estes dados passam por uma sequência de transformações, que são determinadas por funções básicas da API, como *map*, *filter*, ou *keyBy*. Por fim, deve-se indicar no programa uma *Sink*, que é o local onde os resultados são registrados ou apresentados. Este fluxo, em que o stream é produzido e consumido por tarefas, representa o *Job* a ser executado pelo Flink.

Para o desenvolvimento de uma aplicação para compressão Bzip2 com esta estrutura, baseou-se na versão serial do Bzip2 [Seward 2017]. Primeiramente, determinou-se que as operações da compressão e a descompressão dividiam-se em três estágios: receber dados do arquivo de entrada, comprimi-los ou descomprimi-los e escrevê-los no arquivo de saída. Seguindo o padrão de programação descrito anteriormente, a leitura do arquivo de entrada representaria a *Source* da execução enquanto a escrita representaria a *Sink*, e a compressão ou descompressão dos dados poderia ser realizada por uma operação, formando a estratégia da Figura 1b. Optou-se por usar a função *map* para esta operação, pois se recebe um conjunto de bytes por entrada e se retorna um conjunto de bytes de saída, este comprimido ou descomprimido. Ademais, assim como na versão SPar, esta tarefa intermediária é aquela onde podem haver múltiplas instâncias paralelas de execução.

Em grande parte das outras aplicações Bzip2 com *frameworks* para programação paralela em C++, a alternativa utilizada para comprimir ou descomprimir, efetivamente, os blocos do stream era a biblioteca libbzip2, de Julian Seward [Seward 2017]. Para esta implementação, foram buscadas bibliotecas equivalentes para programação em Java, e optou-se pela utilização da biblioteca *Apache Commons Compress* [1].

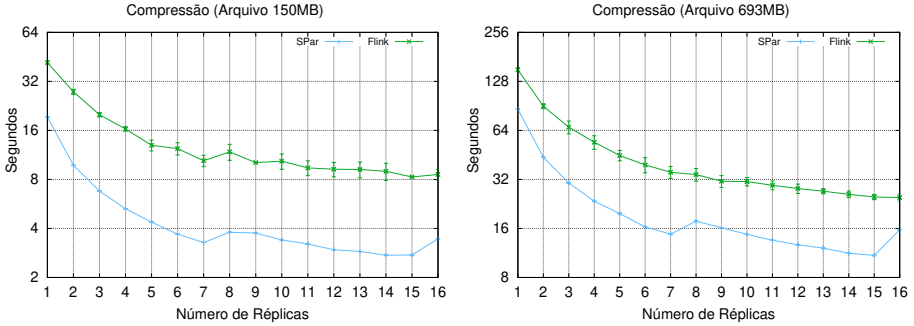
3. Resultados

Os experimentos foram realizados em uma máquina equipada com dois processadores Intel(R) Xeon(R) CPU E5530 2.40Ghz, totalizando 16 threads, e 16 GB de memória RAM. O sistema operacional era *Ubuntu Server 64 bits* com kernel *5.4.0-65-generic*. Os programas foram desenvolvidos com a versão 11 do Java e a versão 1.12.0 do Flink, e compilados com a versão 9.3.0 do compilador G++. No caso da versão em C++, ressalta-se também que foi utilizada a *flag* -O3 para otimização. Os testes foram executados com dois arquivos: um de 150 MB (banco de dados do wikipedia) e outro de 693 MB

[1] https://commons.apache.org/proper/commons-compress/

(XML do wikipedia) [2]. Para compressão, foi considerado o tamanho padrão de blocos de stream Bzip2 (900kB). Após rodadas de *warm-up* da JVM a fim de descartar o tempo de compilação do resultado final, estes experimentos foram repetidos 10 vezes para cada amostra e foram calculados a média aritmética e o desvio padrão a partir dos resultados.

Primeiramente, analisou-se os dois programas em sua forma sequencial. Nestas circunstâncias, a compressão do arquivo de 150 MB resultou em uma média de 37.30 segundos para o código em Java e em 19.52 segundos para o código em C++, e a sua descompressão levou em média 8.26 segundos para o Java e 5.82 segundos para o C++. Já para o arquivo de 693 MB, a média da compressão foi de 135.21 segundos para a aplicação Java e 88.19 segundos para seu equivalente em C++, enquanto a média da descompressão foi de 48.48 segundos para o Java e de 36.54 para o C++. Os valores resultantes revelam um tempo de execução mais baixo para o C++, principalmente durante a compressão. Para as execuções feitas no padrão *pipeline*, os gráficos das Figuras 2 e 3 ilustram os resultados da média do tempo de execução para as operações de compressão e descompressão, onde o eixo x representa o grau de paralelismo de 1 a 16, o número de réplicas do segundo estágio. Ademais, a linha azul representa os resultados da versão SPar, enquanto a linha verde os da versão Flink, e o desvio padrão dos resultados foi marcado no gráfico por barras verticais.



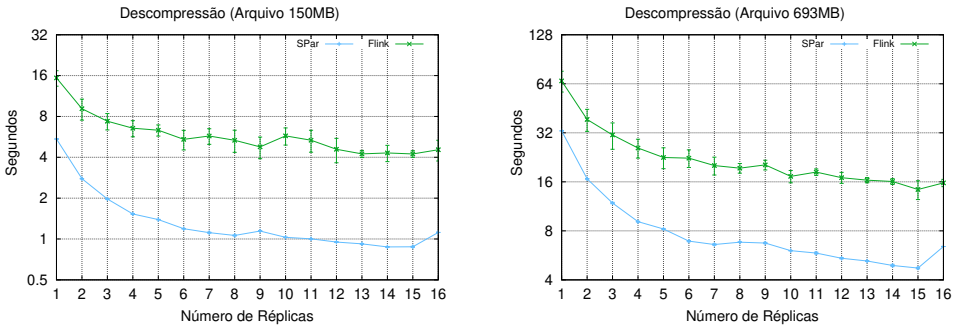
(a) Arquivo de 150MB **(b) Arquivo de 693MB**

Figura 2. Média do tempo de execução da compressão, em segundos

Conforme explicitado na Figura 2, o tempo de execução da versão SPar foi significativamente menor em todos os cenários testados, ou seja, independente do número de réplicas. As mesmas observações podem ser feitas sobre os gráficos da Figura 3, pois, embora a descompressão seja mais rápida para as duas versões Bzip2, a diferença entre os tempos de execução da SPar e do Flink se mantém. Além disso, como visto com clareza na Figura 3a, por exemplo, o Flink apresentou um grande desvio padrão, enquanto o valor desta medida para os testes com SPar foi mínimo.

Isso pode ser explicado, principalmente, pelas diferenças entre a execução em Java e em C++, tendo em vista que mesmo a versão sequencial da nova implementação Bzip2 já apresenta um desempenho inferior à versão sequencial em C++. Mesmo com estratégias como a compilação *just-in-time* e *adaptive compilation*, o sistema de *runtime* do Java tende a ter uma sobrecarga significativa, perdendo um pouco da velocidade em troca de mais portabilidade e segurança. Acredita-se que funcionalidades adicionais do Flink, a exemplo da criação de *snapshots* e do uso de pontos de verificação para lidar com falhas, podem agravar estes resultados, mas que os valores obtidos se devem sobretudo ao Java. Tendo em vista estes resultados, ainda que ambos alcancem alto desempenho, a implementação do algoritmo de compressão Bzip2 em Flink não apresentou desempenho

[2]https://dumps.wikimedia.org/plwiki/



(a) Arquivo de 150MB **(b) Arquivo de 693MB**

Figura 3. Média do tempo de execução da descompressão, em segundos

superior à versão em SPar, de maneira que a linguagem SPar e o código em C++ se revelaram mais adequados do que o Flink para este modelo de programa.

4. Conclusões

Este trabalho apresentou alguns conceitos básicos da ferramenta Apache Flink e o padrão de seus programas, além de criar uma nova versão do algoritmo de compressão de dados Bzip2 usando desta ferramenta, com base em uma versão já existente para a linguagem SPar. Esta implementação foi testada e comparada com a da SPar e foi possível observar que a implementação em SPar apresentou um desempenho melhor do que o novo programa desenvolvido com Flink. Como trabalhos futuros, pretende-se desenvolver e avaliar outros programas que exigem muito processamento de stream com Flink, tais como aplicações de vídeo. Além disso, planeja-se criar uma nova versão do Bzip2 em Java usando JNI para chamar a biblioteca libbzip2, e então avaliar o desempenho desta versão em comparação com a que utiliza a Apache Commons Compress.

Referências

Andrade, H. C. M., Gedik, B., and Turaga, D. S. (2014). *Fundamentals of Stream Processing: Application Design, System and Analytics*. Cambridge University Press, Cambridge CB2 8BS, United Kingdom.

Deshpande, T. (2017). *Learning Apache Flink*. Packt Publishing, Birmingham, United Kingdom.

Foundation, A. S. (2019). Apache Flink® — Stateful Computations over Data Streams.

Griebler, D., Danelutto, M., Torquati, M., and Fernandes, L. G. (2017a). SPar: A DSL for High-Level and Productive Stream Parallelism. *Parallel Processing Letters*, 27(01):1740005.

Griebler, D., Hoffmann, R. B., Danelutto, M., and Fernandes, L. G. (2018). High-Level and Productive Stream Parallelism for Dedup, Ferret, and Bzip2. *International Journal of Parallel Programming*, 47(1):253–271.

Griebler, D., Hoffmann, R. B., Loff, J., Danelutto, M., and Fernandes, L. G. (2017b). High-Level and Efficient Stream Parallelism on Multi-core Systems with SPar for Data Compression Applications. In *XVIII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 16–27, Campinas, SP, Brasil. SBC.

Seward, J. (2017). A Program and Library for Data Compression. http://www.bzip.org/1.0.5/bzip2-manual-1.0.5.html.