Compressão de Dados em Clusters HPC com Flink, MPI e SPar

Gabriel Rustick Fim[1], **Greice Aline Welter**[1], **Júnior Löff**[2], **Dalvan Griebler**[1,2]

[1] Laboratório de Pesquisas Avançadas para Computação em Nuvem (LARCC), Faculdade Três de Maio (SETREM), Três de Maio, Brasil

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

{gabifimtm, greicewelter8, loffjh}@gmail.com, dalvan.griebler@pucrs.br

Resumo. *Este trabalho visa avaliar o desempenho do algoritmo de compressão de dados Bzip2 com as ferramentas de processamento de stream Apache Flink, MPI e SPar utilizando um cluster Beowulf. Os resultados mostram que as versões com maior desempenho em relação ao tempo sequencial são o MPI e SPar com speed-up de* $7,6$ *e* $7,2$ *vezes, respectivamente.*

1. Introdução

O paradigma de processamento de *stream* têm crescido em popularidade nos últimos anos como uma solução viável frente a enorme quantidade de dados que vêm sendo produzidos. O processamento de stream consegue lidar com fluxos contínuos de dados, processando-os sob-demanda, podendo alcançar baixa latência e alta vazão. Em algumas aplicações, os dados processados pelo sistemas de *streaming* não podem ser descartados, já que o seu valor está na informação. É o caso de sistemas financeiros, sistemas em nuvem e projetos globais, como o Wikimedia[1]. Nessas aplicações, os dados são armazenados em discos para acumular conhecimento. Entretanto, a quantidade de dados é tão grande que torna-se necessário comprimi-los para atingir tal objetivo.

Dentre os algoritmos para compressão de dados, o Bzip2 [Gilchrist 2004] é uma importante ferramenta para comprimir e descomprimir dados a fim de reduzir o tamanho dos arquivos a serem armazenados em um servidor. O Bzip2 é um algoritmo de compressão *lossless*, ou seja, garante que dados não são perdidos entre compressão e descompressão. Consequentemente, o algoritmo é mais custoso e requer maior poder de processamento para realizar a compressão. Desta forma, o Bzip2 pode explorar arquiteturas distribuídas para acelerar a computação e aumentar sua escalabilidade, a fim de viabilizar a compressão. Especialmente no contexto de processamento de *stream* distribuído.

No contexto de processamento de *stream* distribuído, várias soluções foram propostas na literatura. Neste trabalho, serão investigadas três delas: Apache Flink, MPI e SPar. (1) Apache Flink é uma solução implementada em Java e representa o estado-da-arte para processamento de *stream* distribuído. (2) O MPI é uma biblioteca escrita em Fortran e C/C++ que é utilizada no desenvolvimento de aplicações HPC (*High Performance Computing*). (3) A SPar é uma linguagem específica de domínio que pode ser utilizada para expressar paralelismo no código através de anotações de alto nível.

A contribuição desse trabalho é uma avaliação de desempenho dessas três importantes ferramentas distribuídas. Para isso será utilizada uma aplicação real do domínio

[1]https://dumps.wikimedia.org

de compressão de dados (Bzip2) com cargas de trabalho realistas. Através do estudo, espera-se contribuir com os domínios de compressão de dados e sistemas distribuídos para processamento de *stream*. Esse artigo possui a seguinte estrutura. A Seção 2 contém os trabalhos relacionados. A Seção 3 apresenta a aplicação e detalhes de implementação. A Seção 4 discute os resultados e a Seção 5 apresenta as conclusões e trabalhos futuros.

2. Trabalhos Relacionados

Trabalhos relacionados são descritos na literatura. No artigo [Mello et al. 2021], os autores implementaram o algoritmo de compressão de dados Bzip2 com o *framework* Apache Flink e SPar, a fim de avaliar o desempenho em sistemas *multicore*. Os autores concluem que os experimentos utilizando a SPar obtiveram melhor desempenho comparado ao Flink. No artigo [Griebler et al. 2017] os autores buscaram avaliar a programabilidade e eficiência da linguagem SPar. Os autores utilizaram os compressores Lzip e Bzip2, que revelaram que o SPar é capaz de explorar com eficiência o paralelismo de *stream*, bem como fornecer abstrações adequadas com menos intrusão de código e refatoração de código. No trabalho [Astsatryan et al. 2020], os autores utilizaram o Apache Hadoop e Apache Spark a fim de monitorar o desempenho da memória e o tempo de transferência entre discos de um *Cluster* utilizando diferentes algoritmos de compressão, dentre eles o Bzip2. O objetivo deles era selecionar o melhor algoritmo e também seus parâmetros. Diferente desses trabalhos, o objetivo deste artigo é avaliar e investigar três ferramentas para processamento de *stream* distribuído utilizando o Bzip2 para a avaliação de desempenho.

3. Implementação

Para a análise de desempenho proposta nesse artigo foram investigadas três implementações paralelas da aplicação Bzip2. O seu método de compressão é baseado na transformada de Burrows-Wheeler capaz de otimizar caracteres que aparecem frequentemente em um mesmo conjunto de dados. Nesse método, a compressão de dados é computacionalmente custosa, enquanto que a descompressão é mais rápida. Uma das principais características do Bzip2 é que o desempenho está associado com o arquivo de entrada, onde diferentes conjuntos de dados podem introduzir desbalanceamento na aplicação.

O paralelismo no Bzip2, nas versões Flink [Mello et al. 2021] e SPar (este trabalho), foi introduzido utilizando o padrão paralelo Farm com três estágios. Na versão MPI [Gilchrist 2004], a implementação é uma versão oficial encontrada na literatura que utiliza o padrão paralelo mestre-escravo. A principal diferença entre as versões é no Farm existem o emissor, trabalhadores paralelos e o coletor. O emissor e coletor são responsáveis por ler o arquivo de entrada e escrever o resultado no arquivo de saída, respectivamente. Por outro lado, na versão oficial do MPI existe o mestre e trabalhadores/escravos paralelos. Ou seja, o mestre fica responsável por ambas tarefas de leitura e escrita.

As três versões (Flink, MPI e SPar) possuem estratégias paralelas similares. Além disso, a implementação da aplicação Bzip2 também é equivalente. Apesar da versão sequencial para Flink ser implementada em Java, essa versão é uma tradução direta entre Java e C++ conduzida por Mello et al. [Mello et al. 2021]. Para executar os experimentos no Apache Flink, foi necessário configurar o ambiente Java e Maven em cada nodo do *cluster*. No MPI a configuração foi um pouco diferente já que a versão pré-instalada nos nodos apresentou erros, por isso foi necessário recompilar outra versão do mesmo.

Na SPar foram testadas duas versões de escalonamento para avaliar o seu impacto no desempenho. Uma delas foi implementada utilizando o escalonamento *on-demand*,

ou seja, as tarefas são distribuídas sob demanda para os trabalhadores. A versão alternativa utiliza o escalonamento *round-robin*, que distribui as tarefas em uma proporção equivalente e de forma circular entre os trabalhadores.

4. Resultados

Os experimentos foram executados na infraestrutura do LARCC[2]. Criou-se cinco máquinas virtuais dentro da ferramenta OpenNebula 6.0.0.2 com o virtualizador KVM, sendo um *Master* com 2 vCPUS, 4GB de memória RAM e 65GB de disco e quatro *Workers* com 2 vCPUS, 4GB de memória RAM e 15GB de disco. Estas foram configuradas como um *cluster Beowulf* com NFS para compartilhamento de arquivos. As máquinas virtuais onde os testes foram executados possuem as mesmas versões de software: Sistema operacional Ubuntu *Server* 20.04.3 LTS (Focal Fossa) (5.15.12-051512-generic), JDK e JRE 11.0.10, Apache Flink 1.14.2 e Open MPI 4.1.2.

Os testes foram executados em três versões paralelas do algoritmo de compressão BZIP2. O `Flink` representa uma versão do BZIP2 convertida para a linguagem Java e implementada com o *framework* Apache Flink [Mello et al. 2021]. As versões `SPar` e `SPar-RR` foram geradas automaticamente pelo compilador da SPar através de anotações no código sequencial do BZIP2 em C++. As anotações fazem parte da linguagem da SPar e expressam informações sobre o fluxo de dados no código. A versão MPI é uma versão oficial do MPIBZIP2 utilizando o Open MPI para sistemas distribuídos. Essa versão foi baseada em uma versão paralela para arquiteturas multi-core [Gilchrist 2004].

Nos testes foram utilizados dois arquivos *dump* obtidos do servidor da WikiMedia. Um deles está no formato XML e outro no formato SQL, com tamanho aproximado de 2GB e 415MB respectivamente. Cada configuração de processos foi repetido intercaladamente 5 vezes e foram calculados a média aritmética e o desvio padrão a partir dos resultados. O desvio padrão está representado através de barras de erro, que talvez não sejam visíveis quando o valor é baixo. Os gráficos apresentam o tempo de execução em segundos (eixo y) em relação a quantidade de processos paralelos (eixo x). Não foi possível realizar a comparação dos códigos sequenciais para comparar a versão Java vs. C++. O motivo é que algumas versões não possuem suporte a esta forma de execução.

A Figura 1 ilustra os resultados dos experimentos. Nota-se que apesar do Apache Flink ser o estado-da-arte para processamento de *stream* distribuído, o seu desempenho foi menor que as demais versões. É possível observar que todas as versões possuem uma curva de tempo decrescente e que o `Flink` possui uma diferença significante no tempo necessário para o processamento da compressão quando comparado aos códigos do `MPI` e da `SPar`, isto pode ser explicado pela diferença entre as linguagens Java e C++. Nota-se também que apesar das versões utilizarem diferentes padrões paralelos, não há uma diferença expressiva entre os tempos obtidos com as versões `MPI` e `SPar`.

Os resultados também revelaram que o código paralelo gerado automaticamente pela SPar é comparável com um código MPI implementado e otimizado manualmente por um especialista. Os melhores *speed-ups* foram obtidos pelas versões `MPI` e `SPar`, sendo o *speed-up* médio de $7,6$ e $7,2$ vezes, respectivamente. As versões `SPar-RR` e `Flink` obteram um *speed-up* de $6,7$ e 6 vezes comparando a execução sequencial do BZip2.

Referente aos dois tipos de escalonamento testados nas SPar, *on-demand* (`SPar`) e *round-robin* (`SPar-RR`), percebe-se que houve uma pequena diferença no desempenho.

[2]Laboratório de Pesquisas Avançadas para Computação em Nuvem: http://larcc.setrem.com.br/

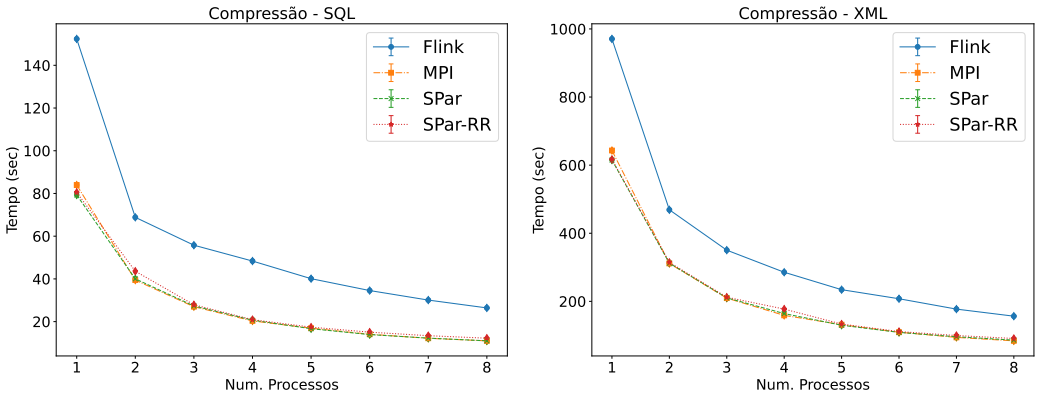

Figura 1. Resultados dos testes no ambiente.

O *speed-up* máximo obtido pela versão `SPar` foi $7,4\%$ maior que o *speed-up* obtido pela `SPar-RR`. Isso corrobora com a explicação do BZIP2 discutida na Seção 3 sobre o desbalanceamento de carga. Em outras palavras, o escalonamento *on-demand* consegue melhorar o balanceamento de carga, pois um trabalhador recebe outra tarefa somente após terminar a anterior. Desta forma, se tarefas mais pesadas forem atribuídas para o mesmo trabalhador, ele receberá menos que os demais. Diferenças maiores são observadas com cargas de trabalho mais desbalanceadas.

5. Conclusões

Esse trabalho apresentou uma avaliação de desempenho de três importantes ferramentas no contexto de processamento de *stream* distribuído: Apache Flink, MPI e SPar. Considerando a aplicação Bzip2 e o ambiente distribuído dos experimentos, os resultados com diferentes cargas de trabalho revelaram que o MPI e a SPar possuem desempenho superior ao Apache Flink, que é uma das principais ferramentas utilizadas na indústria. Os testes também mostraram que a SPar, mesmo sendo uma linguagem de alto-nível, possui desempenho equivalente ao MPI nesta aplicação de compressão de dados. Esses resultados são positivos e promovem que abstrações são capazes de aumentar a programabilidade sem perdas significativas de desempenho. Como trabalhos futuros pretende-se avaliar outras aplicações de processamento de *stream* e executar testes em um ambiente distribuído maior e com graus de paralelismo maiores.

Referências

Astsatryan, H., Kocharyan, A., Hagimont, D., and Lalayan, A. (2020). Performance Optimization System for Hadoop and Spark Frameworks. *Cybernetics and Information Technologies*, 20:13.

Gilchrist, J. (2004). Parallel data compression with bzip2. In *Proceedings of the 16th IASTED international conference on parallel and distributed computing and systems*, volume 16, pages 559–564. Citeseer.

Griebler, D., Hoffmann, R. B., Loff, J., Danelutto, M., et al. (2017). High-Level and Efficient Stream Parallelism on Multi-core Systems with SPar for Data Compression Applications. In *XVIII WSCAD*, pages 16–27, Campinas, SP, Brasil. SBC.

Mello, F., Griebler, D., Manssour, I., and Fernandes, L. G. (2021). Compressão de dados em multicores com flink ou spar? In *Anais da XXI Escola Regional de Alto Desempenho da Região Sul*, pages 77–80, Porto Alegre, RS, Brasil. SBC.