Compressão de imagens em uma arquitetura de *streaming* usando K-Means

1st Samuel Amico Fidelis
Laboratório de Pesquisa em Sistemas Distribuídos - LaPeSD
UFSC
Florianópolis, Brasil
sam.fst@gmail.com

***Abstract*—Much data is being generated continuously and infinitely, being classified as unbounded data. Streaming architectures are being used to handle this data in near real time to generate fast and accurate inshigths. However, to obtain this almost real time problems like pressure on memory, intensive use of I / O writing on broker and limitation on the bandwidth of the internet end up being crucial to operate such data.**

This paper proposes a solution to reduce these problems through data compression by the K-Means algorithm, where the sequence and parallel implementation of the same will be compared.The type of data used will be color images and compression will be achieved by eliminating irrelevant information in the images.

***Index Terms*—K-Means, streaming processing, compression**

I. Introdução

O processamento de dados em *streaming* está crescendo no cenário atual. As empresas anseiam por *insights* cada vez mais relevantes e mais rápidos sobre seus dados, que adotam ferramentas de processamento para *streaming* de dados, visto que é uma boa maneira de alcançar tais objetivos [1]. Novas tecnologias do século 21 como sensores IoT das indústrias 4.0, aplicativos e websites geram um enorme fluxo de dados contínuos e sem fim, estes são mais facilmente controlados usando um sistema projetado para volumes intermináveis de dados, nos quais podem-se gerar como resultado final para os usuários ou engenheiros de software: dashboard, painéis de controle de fluxo e entre outros sistemas de análise contínua em tempo real ou quase [2].

Porém, existem vários desafios para realizar o processamento de *streaming*, o principal deles é encontrar no domínio do tempo. Para entender este problema é necessário ter conhecimento sobre as 2 variáveis envolvidas no domínio do tempo: a) *Event Time-et*: o tempo que o evento foi gerado; b) *Processing Time-pt*: o tempo que o evento foi observado pelo sistema. Em um mundo ideal, teria-se o *Event = Processing time*, contudo, o que realmente se obtém é um valor *fi* = et – *pt*.

A fim de gerar valores significativamente bons no *fi*, os mecanismos de processamento de *streaming*, normalmente, exigem que os dados de fluxo, vindo em grandes volumes, residam na memória principal para serem processados, podendo citar o *framework Spark* como um exemplo muito conhecido e utilizado [3].

Entretanto, essa solução é operacionalmente custoso, pois este tipo de processamento gera uma enorme pressão na capacidade e na largura de banda do sistema [4]. Uma forma de diminuir este problema é comprimir os dados, quando possível. Desse modo, o trabalho presente irá demonstrar os efeitos da compressão de *streaming* de dados em imagens vindas de um produtor que irá produzir um fluxo contínuo de imagens para um *broker* Kafka.

II. Compressão de dados em streaming

A compressão de dados é explorada em muitos domínios científicos e tem sido amplamente empregada para minimizar a utilização de disco, o consumo de largura de banda de rede e reduzir o consumo de energia em hardware [5]. Entretanto, devido as exigências e características para processamento de dados em *streaming*, a compactação de dados é distinta da compactação de dados tradicionais em banco de dados que utilizam de algoritmos sofisticados de compressão [5]. Pekhimenko *et al* definem dois pontos para avaliar a utilidade da compressão para streaming: A primeira é verificar se o fluxo de dados são compactáveis. A segunda é se a compressão melhora a taxa de transferência do processamento de fluxo [4]. Todavia, outro ponto importante: o *speedup* do algoritmo, com o intuito de não interferir de forma significativa no *throuput* de dados.

III. Compressão de Imagens

A compressão de imagens é a compressão de dados que representam a imagem (2D). As imagens possuem quantidades de dados irrelevantes ou redundantes, sendo classificada em três principais tipos de redundância: a) Redundância de codificação, b) Redundância espacial e temporal, c) Informações irrelevantes [14]. Uma das maneiras mais simples de comprimir um conjunto de dados é remover dados supérfluos do conjunto. No contexto de imagens digitais, as informações ignoradas pelo sistema visual humano são candidatas óbvias para a omissão. No caso em uma imagem colorida, vários tons de cores são representados com diferentes

valores RGB, mas são visualmente idênticas para o sistema visual. Portanto, é possível utilizar um tipo de codificação que aplica cores que tem tons semelhantes e as torna em um único valor RGB. Um possível algoritmo de codificação para solucionar este problema é o K-Means.

IV. Estado da Arte

Pekhimenko *et al* impulsionados por consultas em dados de *streaming* reais, identificaram fatores que influenciam à compactação de dados. Primeiramente, eles obtiveram resultados positivos ao realizar uma simples compressão de dados, eles testaram reduzir dados (Long para Char) e perceberam que quando a quantidade de dados transferidos é reduzida em X vezes, o *throughput* foi aumentado proporcionalmente. A melhoria significativa indica que, na ausência de outros gargalos artificiais, a taxa de transferência desse mecanismo de *streaming* simples é limitada apenas pela largura de banda da memória principal, e a compressão reduz a pressão da largura de banda em paralelo à taxa de compressão [4]. Em segundo lugar, o uso de algoritmos de compactação simples já oferece benefícios em relação ao não uso destes, tornando estes uma abordagem atraente para melhorar o desempenho do processamento de fluxo [4]. No presente trabalho, o algoritmo escolhido para compactação de dados em *streaming* (os dados são imagens coloridas) foi o K-Means e será comparado a sua forma sequêncial e paralela.

No que diz respeito ao autor Kucukyilmaz, ele obteve bons resultados utilizando o K-means em uma arquitetura de multiprocessadores [7]. Para avaliar se o desempenho do algoritmo k-means paralelo, Kucukyilmaz utilizou capacidades de *threads* diferentes, e teve como resultado que a partir de 40 mil instâncias, o algoritmo apresentou perdas no tempo de execução, devido ao fato que todas as *threads* ficaram saturadas e as instâncias precisaram esperar que estas terminassem antes de serem processadas. Entretanto, com um valor menor que 40 mil instâncias o algoritmo em sua versão paralela teve um ganho de 1/8 de tempo de execução comparada com sua versão sequêncial do algoritmo.

V. Algoritmo K-Means

O K-means é um algoritmo de aprendizagem de máquina não supervisionado que resolve problemas de *clustering* [8]. O *clustering* ou agrupamento de dados descreve uma técnica importante de classificação não supervisionada que organiza os dados em grupos baseados em padrões [9]–[11], é uma técnica muito utilizada em processamento de *streaming*, pois ao contrário da classificação que analisa instâncias rotuladas, o *clustering* não tem estágio de treinamento e as classes não são previamente conhecidas tornando esta técnica ideal para processar dados *unbounded*. A ideia principal do K-means é: um conjunto de n pontos de dados no espaço d-dimensional real, R_d, e um inteiro k, no qual o problema se baseia em determinar um conjunto de k pontos em R_d, chamados centros, em busca de minimizar a distância quadrada média de cada ponto de dados para seu centro mais próximo. Esta medida é frequentemente chamada de distorção de Erro Quadrático [12], [13] :

$$J = \sum_{i=1}^{N} \sum_{j=1}^{n} \parallel x_i^j - c_j \parallel^2$$

O algoritmo para segmentação de imagem pelo método de agrupamento k-means pode ser dividido nas seguintes etapas:

1) Fornecer os valores para os centroides – os k centroides devem receber valores iniciais;
2) Gerar uma matriz de distância entre cada ponto e os centroides predefinidos;
3) Colocar cada ponto nas classes de acordo com sua distância do centroide da classe;
4) Calcular os novos centroides para cada classes;
5) Repetir até a convergência.

VI. K-Means Paralelo

Nesta seção, será discutida a implementação de um algoritmo k-means baseado em multiprocessadores de memória compartilhada. Uma arquitetura de memória compartilhada é um sistema de computador com vários processadores, nos quais estes podem acessar um único bloco de memória simultaneamente [7]. O algoritmo paralelo é descrito em etapas semelhantes a sua forma sequencial:

1) Seleciona os K centroides e defini-los com valores;
2) Associa-se vários pontos ao cluster e cada cluster está associado a uma thread. Calcula-se então a média dos k clusters. (A associação de um ponto ao seu cluster é independente dos outros pontos, da mesma forma o cálculo da média de um cluster é independente dos outros);
3) Sincronização dos valores de média obtidos pelas threads – colocando cada ponto nas classes de acordo com sua distância do centroide. Calcular os novos centroides;
4) Repetir até a convergência.

É importante salientar que no passo 2, os dados são atribuídos as *threads* em *"batchs"*, por isso que o k-means é um exemplo de algoritmo paralelo síncrono em massa ou – *Bulk synchronous parallel algorithm*.

VII. Experimento e Resultados

Foi conduzido experimentos extensivos em implementações sequênciais e paralelas do algoritmo k-means. Nestes experimentos, ambas as versões foram testadas em um Intel i7-6500 com 8 núcleos funcionando a 2,5 GHz e com 12 GB de RAM. Foram escolhidas um total de 40 imagens, nas quais foram inseridas em um tópico Kafka com 2 Brokers rodando em uma Virtual Machine Linux local. Foram realizados 6 testes com valores de K (número de clusters) diferentes, porém obedecendo a seguinte regra: não deverá haver perda da qualidade visual ao sistema visual humano. O algoritmo foi escrito em Scala (compilado na JVM) utilizando as estruturas de dados sequênciais e paralelas da linguagem de programação. Foi utilizado a biblioteca ScalaMeter para fins de *Benchmarking*.

Para cada teste foi realizado um conjunto de 60 trials aplicando o *warmup* do Java, remoção de outliers, média e cálculo de variância dos tempos de execução do algoritmo. A tabela a seguir mostra um teste realizado em uma imagem de 323 Kb.

TABLE I
TEMPO DE EXECUÇÃO E TAMANHO DAS IMAGENS COMPRIMIDAS

K	Sequêncial(ms)	Paralelo(ms)	Tamanho (Kb)
12	36.407	19.3483	36
26	50.105	30.208	85
28	53.002	31.092	85
32	53.502	30.952	93
42	51.239	34.802	105
62	90.812	49.7142	117

As imagens abaixo com diferentes valores de K, variando da esquerda para direita: K = 12, K = 28, K = 32, K = 62.

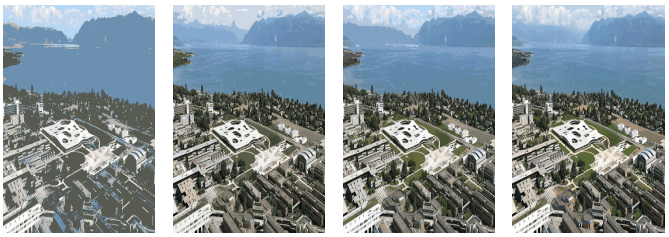

Fig. 1. Variações de K

Comparando os resultados mostrados na tabela acima, percebe-se que a maior diminuição alcançada foi com k = 26 – obtendo uma redução de 73% do tamanho em Kb em relação a imagem original, sem perdas para o sistema visual humano, e apresentando um *speedup* de 1.658.

Na arquitetura de processamento de *streaming*, a compressão de dados é realizada pelo produtor antes de enviar os eventos para o *broker* da fila de mensageria, neste presente trabalho foi usado o Kafka como software de mensageria. O próprio Kafka apresenta algoritmos de compressão de eventos como o ltz e gzip, entretanto, não é mostrado uma compressão específica para imagens/vídeo. A arquitetura de *streaming* empregada por este trabalho pode ser descrita na figura abaixo:

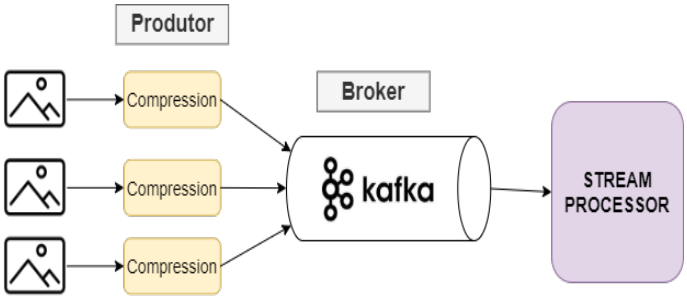

Fig. 2. Arquitetura *Streaming*

Para validar os testes de performance no broker kafka, foram obtidas as seguintes métricas, com a aplicação de imagens não comprimidas: Uso de CPU = 4.3%; Uso de Memória = 3.2%; Escrita I/O = 4.07%.

Com o emprego de imagens comprimidas com diferentes valores de K, foram geradas as seguintes métricas no mesmo broker kafka:

TABLE II
USO DE CPU COM IMAGENS COMPRIMIDAS

K	CPU (%)	I/O (%)
12	1.5	2.00
26	2.1	2.40
28	2.1	2.42
32	2.5	2.47
42	2.7	2.61
62	3.33	3.09

Em relação ao uso de mémoria, os valores obtidos usando compressão ou não, não apresentaram diferença significativa, um dos fatores se deve ao limite de imagens utilizadas no produto.

O tempo de envio dos eventos com compressão é menor do que os eventos sem compressão, todavia o tempo que leva para que cada imagem ser processada pelo algoritmo de K-Means acaba em média tornando o tempo de envio, aproximadamente, igual para os dois casos de imagens, com e sem compressão. Mas, o ganho real mostrado nos resultados é em relação ao uso de CPU e escrita em disco. O broker do Kafka aplica muito mais recursos de CPU e disco em eventos que não passaram pela técnica de compressão, podendo destacar uma enorme vantagem de se empregar o algoritmo do K-Means para processamento de *streaming*.

VIII. CONCLUSÃO

Ao longo deste trabalho, um algoritmo k-means paralelo em multiprocessadores de memória compartilhada foi comparado com sua versão sequêncial, com o objetivo de comprimir os dados para minimizar os defeitos atrelados ao processamento de dados em streaming em memória. Portanto, obtendo as seguintes contribuições:

1) O algoritmo do K-means apresentou um desempenho satisfatório ao diminuir o tamanho (em KB) das imagens processadas e sua versão paralela apresentou um tempo de execução igualmente satisfatório para uma arquitetura de *streaming*;

2) Além disso, gerou a diminuição do uso de recursos de CPU e memória no broker. Vale ressaltar que os resultados foram obtidos com apenas uma máquina executando todo trabalho de processamento, ou seja, somente um nó responsável.

Logo, as perguntas propostas no início do trabalho em relação a viabilidade da compressão de dados em um processamento de *streaming* foram respondida com sucesso para o uso da versão em paralelo do algoritmo K-means em imagens coloridas. Assim, obtendo resultados promissores para o emprego de apenas um nó em uma arquitetura de *streaming*.

REFERENCES

[1] Akidau, Tyler, Slava Chernyak, and Reuven Lax. Streaming systems: the what, where, when, and how of large-scale data processing. " O'Reilly Media, Inc.", 2018.

[2] Pekhimenko, Gennady, et al. "Tersecades: Efficient data compression in stream processing." 2018 USENIX Annual Technical Conference (USENIXATC 18). 2018.

[3] Barga, Roger S., et al. "Consistent streaming through time: A vision for event stream processing." arXiv preprint cs/0612115 (2006).

[4] SPARK, Apache. Apache spark. Retrieved January, v. 17, p. 2018, 2018.

[5] Graefe, Goetz, and Leonard D. Shapiro. Data compression and database performance. University of Colorado, Boulder, Department of Computer Science, 1990.

[6] Gonzalez, Rafael C., Richard Eugene Woods, and Steven L. Eddins. Digital image processing using MATLAB. Pearson Education India, 2004.

[7] Banerjee, Ayan, and Amiya Halder. "An efficient image compression algorithm for almost dual-color image based on k-means clustering, bitmap generation and RLE." 2010 International Conference on Computer and Communication Technology (ICCCT). IEEE, 2010.

[8] Dehariya, Vinod Kumar, Shailendra Kumar Shrivastava, and R. C. Jain. "Clustering of image data set using k-means and fuzzy k-means algorithms." 2010 International Conference on Computational Intelligence and Communication Networks. IEEE, 2010.

[9] Pujari, Arun K. Data mining techniques. Universities press, 2001.

[10] Alpaydin, Ethem. Introduction to machine learning. MIT press, 2020.

[11] Jain, Anil K., and Richard C. Dubes. Algorithms for clustering data. Prentice-Hall, Inc., 1988.

[12] Gersho, A., and R. M. Gray. "Vector quantization and signal compression Boston." Kluwer Academic (1992): 309-315.

[13] Kucukyilmaz, Tayfun, and University of Turkish Aeronautical Association. "Parallel k-means algorithm for shared memory multiprocessors." Journal of Computer and Communications 2.11 (2014): 15.