Estudo Sobre Spark nas Aplicações de Processamento de Log e Análise de Cliques

Luan Dopke, Dalvan Griebler

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

luandopke@gmail.com,dalvangriebler@setrem.com.br

Resumo. *O uso de aplicações de processamento de dados de fluxo contínuo vem crescendo cada vez mais, dado este fato o presente estudo visa mensurar a desempenho do framework Apache Spark Strucutured Streaming perante o framework Apache Storm nas aplicações de fluxo contínuo de dados, estas sendo processamento de logs e análise de cliques. Os resultados demonstram melhor desempenho para o Apache Storm em ambas as aplicações.*

1. Introdução

Fluxo contínuo de dados ou *data stream* pode ser caracterizado por dados que continuamente são transmitidos de alguma fonte, como maquinários ou qualquer ferramenta que ininterruptamente infere dados. O principal objetivo de manipular fluxos contínuos de dados é processar dados atuais com total integração, provendo informação em tempo real e resultados para os usuários finais, enquanto monitora e auxilia na tomada de decisões.

Fazer a escolha de Mecanismos de Processamento de Fluxo de Dados (*Data Stream Processing Engines* - DSPEs) adequados para as aplicações é uma tarefa árdua e dificultosa. Benchmarks específicos para processamento de dados em fluxo contínuo costumam ser utilizados para mensurar o desempenho de DSPEs em ambientes e aplicações distintas. A suíte de benchmark DSPBench, desenvolvida no trabalho de [Bordin et al. 2020], possibilita mensurar o desempenho de diversas aplicações implementadas no Apache Storm, Apache Flink e Apache Spark Structured Streaming.

Estudos prévios abordam o desempenho do Spark e demais *frameworks*, o trabalho de [van Dongen and Van den Poel 2020] permite ter diversas percepções sobre os frameworks avaliados. Há desempenho superior na latência das aplicações executadas pelo Flink, enquanto a aplicação arquitetada em Spark Structured Streaming demonstra taxa de *throughput* superior em detrimento de latência.

O estudo de [Lu et al. 2014], avalia o desempenho de ambos sob ambiente de tolerância a falhas. Os experimentos demonstraram maior *throughtput* para o Apache Spark Streaming e menor impacto em caso de falhas se comparado com o Apache Storm, por outro lado, o Storm demonstra menor latência na maioria dos cenários.

O objetivo do trabalho é utilizar o benchmark DSPBench ([Bordin et al. 2020]) para avaliar o *throughtput* dos *frameworks* Apache Storm e Apache Spark Structured Streaming ao executar duas aplicações: Processamento de Logs e Análise de cliques. O experimento será executado em Nuvem Privada, alternando a configuração de processamento paralelo de ambos *frameworks* em 1, 2, 4 e 8, visando identificar a escalabilidade de ambos.

2. Experimentos

O presente estudo usará o benchmark DSPBench para mensurar o desempenho de duas aplicações que utilizam o processamento de dados em fluxo contínuo: processamento de

logs e analises de cliques, ambas disponíveis no catálogo de aplicações do benchmark. A aplicação de processamento de logs recebe diversos logs no formato Common Log Format e divide seu processamento em três linhas, a primeira, Volume Counter, conta o número de visitas ao *endpoint* por minuto, o Status Counter, contabiliza o número de ocorrências de cada status HTTP. O Geo Finder, incumbido de localizar o usuário que realizou a requisição utilizando o banco de dados de IP GeoIP, em sequência, é emitido o país e a cidade do respectivo IP para o operador Geo Stats, que irá contabilizar o número de acessos pelos mesmos.

A aplicação análise de cliques recebe dados de usuários que estão acessando um site e implementa duas linhas de processamento. A primeira, com o operador Repeat Visit, identifica se o usuário é recorrente e já visitou o site anteriormente, o VisitStats realiza a contagem total de acessos e a contagem de acessos únicos. A segunda linha de processamento utiliza o operador Geo Finder, incumbido de localizar o usuário que realizou a requisição utilizando o banco de dados GeoIP, culminando no número de visitas por país e cidade.

Ambos os ambientes do experimento, tanto o *cluster* do Apache Storm quanto o *cluster* do Apache Spark são VMs hospedadas na nuvem privada do LARCC, sobre o virtualizador KVM e a ferramenta OpenNebula 6.0.0.2. São compostos por sete nodos, três destes empenhando a função de trabalhadores com 4 vCPUS, 4GB de memória RAM e 20GB de disco, um mestre, composto por 2 vCPUS, 2GB de memória RAM e 20GB de disco. Um nodo executa o Apache Kafka com 2 vCPUS, 12GB de memória RAM e 30GB de disco outro nodo responsável por executar o Yarn (para o Spark) e o Apache Zookeeper (Para o Kafka e Storm), com 3 vCPUS, 4GB de memória RAM e 10GB de disco e um último incumbido de coletar o uso de recursos com o Zabbix.

A nuvem contém os seguintes recursos físicos: três máquinas HP Proliant DL385 G6, com um processador AMD Opteron 2425 2100 MHz 6-Core e 32 GB de memória RAM DDR3, cada uma possuindo 5 interfaces de rede Gigabit. Também há a máquina constituída pelo processador Intel Xeon Silver 4108 8-core/16 threads, 64 GB de memória DDR4 e HD 2 TB. As máquinas virtuais executam o Ubuntu Server 20.04 LTS e contém os softwares: Java JDK e JRE 1.8.0_352, Apache Spark 3.3.1, Apache Hadoop 3.3.0, Apache Storm 2.4.0, Apache ZooKeeper 3.8.0, Apache Kafka 3.3.1, Zabbix 6.0.12.

Para execução do experimento foi criado um *script* visando repetir cinco vezes a execução de cada aplicação de maneira intercalada. Em cada execução são realizados os seguintes passos: todas as aplicações ativas são encerradas, os processos que executam o *framework* do mestre e seus trabalhadores são encerrados, em seguida a memória é limpa em cada nodo utilizando o comando específico para limpar a memória fornecido pelo Kernel, evitando cache dos dados da aplicação, acompanhado da inicialização dos serviços do *framework* e execução da aplicação. A aplicação é finalizada após 10 minutos e estes passos são repetidos para a próxima aplicação. Os passos deste *script* foram executados quatro vezes, para cada configuração de fator de replicação (1, 2, 4 e 8), este, oferecido pela própria configuração do *framework*, fornecendo operações paralelas entre os nodos.

Os operadores selecionados para realizar operações paralelas são caracterizados como *stateless*, isto é, não guardam estados de tuplas. Esta decisão foi tomada visando manter a integridade e consistência da aplicação, pois ao paralelizar operadores *statefull* cria-se mais que um estado por operador, podendo interferir no resultado.

Na Figura 1 é possível visualizar o desempenho de ambas as aplicações nos dois *frameworks*. No Spark foi executado os experimentos com dois parâmetros distintos: tamanho de *micro-batchs* de 10 mil e 100 mil, visando identificar diferença na performance de ambos. O desempenho é mensurado através métrica de média total de operações.

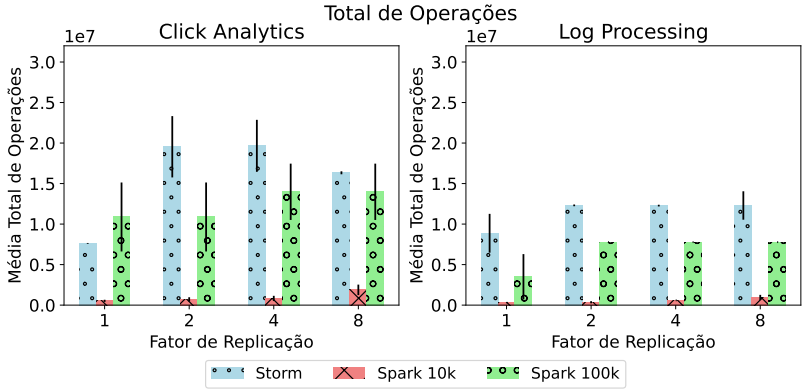

Figura 1. Avaliação de Desempenho das aplicações.

O Apache Storm demonstrou melhores condições de escalabilidade no fator de replicação 2, melhorando entorno de 3x o número de operações quando comparado ao fator de replicação 1 na aplicação de análise de cliques. Na aplicação de processamento de logs, igualmente há melhora significativa entre o fator de replicação 1 e o 2. Enquanto que ambas as configurações do Spark demonstraram ter pouca escalabilidade, com resultados semelhantes quando considerado o desvio padrão.

O baixo desempenho do Spark com *micro-batch* de 10 mil, pode ser explicado pelo comportamento de sua API Structured Streaming. A API espera o processamento de um *micro-batch* terminar e apenas após isso, inicia o processamento do próximo *micro-batch*. Esta metodologia diferencia-se da adotada pelo Storm, o qual processa cada tupla de entrada continuamente, numa *pipeline*.

Na Figura 2 é visível o uso de recursos na execução da aplicação de processamento de logs, nota-se pontos importantes no comportamento de cada *framework*. O Storm passou a utilizar todos os nodos disponíveis ao aumentar o fator de replicação, contudo, um dos nodos sempre demanda mais recursos de CPU. O nodo mestre do Spark obteve o maior uso de recursos, chegando a até 45% do uso de CPU, seus trabalhadores variaram de 20% a 30% de consumo de processamento.

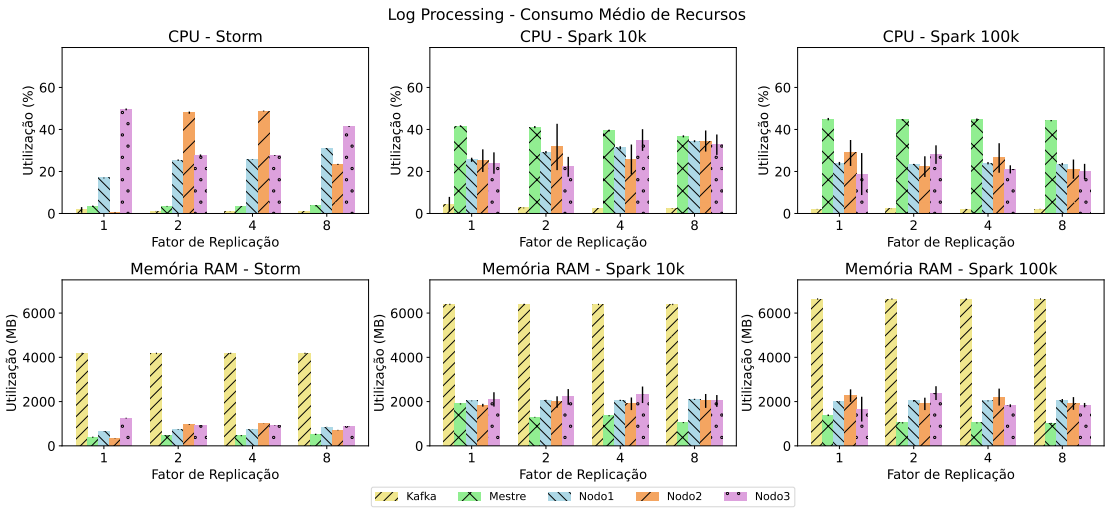

Figura 2. Consumo de recursos da aplicação Processamento de Logs.

Na Figura 3 demonstra-se o uso de recursos durante a execução da aplicação análise de cliques, o poder de processamento do Storm e do Spark são similares ao consumo da aplicação de processamento de log. Nos ambientes com replicação ativa, o Storm melhor distribui a carga entre seus nodos, evidenciado pela utilização acima dos 50% dos nodos 2 e 3. O uso de CPU do nodo mestre no Spark é ligeiramente superior com o *micro-batch* em 100 mil, ao comparar com a configuração de 10 mil.

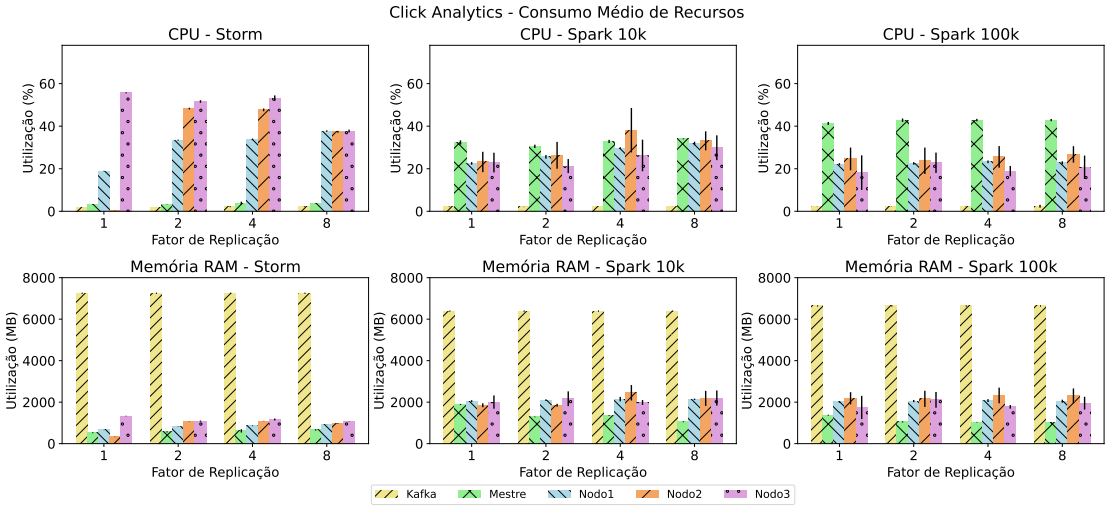

Figura 3. Consumo de recursos da aplicação Análise de Cliques.

Em ambos os *frameworks*, o nodo responsável por injetar os dados na aplicação através do Kafka fez seu maior uso, utilizando até 7GB de memória para o Storm e um valor semelhante para o Spark. Denota-se maior consumo de memória por parte do Spark, com seus trabalhadores frequentemente atingindo o consumo de 2GB, oque não chega a ocorrer com o Storm, que tende a manter seus trabalhadores em torno de 1GB.

3. Conclusões

O artigo avaliou o desempenho de dois *frameworks* de processamento de fluxo de dados contínuo, utilizando as aplicações de processamento de logs e análise de cliques disponíveis na suíte de benchmark DSPBench. Conclui-se que o *framework* Apache Spark apresentou desempenho inferior quando comparado ao Apache Storm neste ambiente, justificando-se pela metodologia adotada pela API Structured Streaming. Realizou-se testes com o fator de replicação em 1, 2, 4 e 8, visando detectar a escalabilidade apresentada pelos *frameworks*. Nisto, Apache Storm demonstrou escalabilidade até 2 replicações simultâneas enquanto que o Spark não obteve resultados satisfatórios. Como trabalho futuro, deseja-se avaliar o Spark Streaming, outra API de processamento de fluxo contínuo de dados do Apache Spark, além de executar experimentos com outras aplicações.

Referências

Bordin, M. V., Griebler, D., Mencagli, G., Geyer, C. F. R., and Fernandes, L. G. L. (2020). Dspbench: A suite of benchmark applications for distributed data stream processing systems. *IEEE Access*, 8:222900–222917.

Lu, R., Wu, G., Xie, B., and Hu, J. (2014). Stream bench: Towards benchmarking modern distributed stream computing frameworks. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 69–78.

van Dongen, G. and Van den Poel, D. (2020). Evaluation of stream processing frameworks. *IEEE Transactions on Parallel and Distributed Systems*, 31(8):1845–1858.