Identificação de *starvation* de aplicação no Apache Spark

Enzo B. Boscatto[1], Anderson H. da S. Marcondes[2], Guilherme P. Koslovski[2]

[1]Universidade do Estado de Santa Catarina - UDESC

[2]Programa de Pós-Graduação em Computação Aplicada - PPGCAP

{enzo.boscatto, anderson.marcondes}@edu.udesc.br

guilherme.koslovski@udesc.br

Resumo. *Este trabalho investiga o fenômeno de starvation de aplicação no Apache Spark induzido por congestionamento de rede durante operações de shuffle. Em um ambiente virtualizado configurado com DCTCP, foram monitoradas métricas granulares de rede (backlog) e de tarefas (Flow Completion Time e tempo de CPU) sob tráfego concorrente. Os resultados revelam aumento superior a 110% no tempo total de execução, enquanto o tempo de uso da CPU permaneceu inalterado. Essa discrepância atesta ociosidade do processamento e sugere a ocorrência de starvation.*

1. Introdução

Sistemas de processamento distribuído, como o Apache Spark [Salloum et al. 2016], dependem do equilíbrio entre recursos computacionais e a infraestrutura de rede do *datacenter*. Contudo, operações de redistribuição de dados (*shuffle*) geram tráfego intenso, tornando o desempenho das aplicações altamente dependente da eficiência dos canais de comunicação. Nesse cenário, o congestionamento da rede vai além de causar atrasos e perdas de pacotes: ele pode induzir um estado de *starvation* [Arun et al. 2022], interrompendo o progresso da execução pela restrição no acesso aos dados oriundos da rede.

Este trabalho investiga a hipótese de que a contenção na rede causa um *starvation* induzido por I/O, manifestado pela subutilização da CPU enquanto o sistema aguarda a chegada de pacotes. Para validar essa premissa, propõe-se um cenário experimental que submete aplicações baseadas em Spark a diferentes condições de carga. A metodologia adotada permite correlacionar o congestionamento da infraestrutura com a degradação de desempenho sob a perspectiva final da aplicação utilizando métricas granulares de rede e de tarefas.

2. Cenário experimental

O ambiente experimental foi implantado em uma infraestrutura virtualizada (Ubuntu Server 22.04), interconectando sete Máquinas Virtuais (VMs) por meio de um nó roteador central, responsável pelo monitoramento granular do tráfego (Figura 1). O *cluster* Apache Spark 3.5.5 é composto por cinco instâncias: uma *master* e quatro executores, cada um configurado com 1 núcleo de CPU e 2 GB de RAM. Uma VM adicional foi dedicada exclusivamente à geração de tráfego concorrente por meio da ferramenta iPerf, visando induzir estados controlados de congestionamento.

A pilha de rede das VMs utilizou o protocolo *Data Center TCP* (DCTCP) [Alizadeh et al. 2010] com *Explicit Congestion Notification* (ECN) ativado.

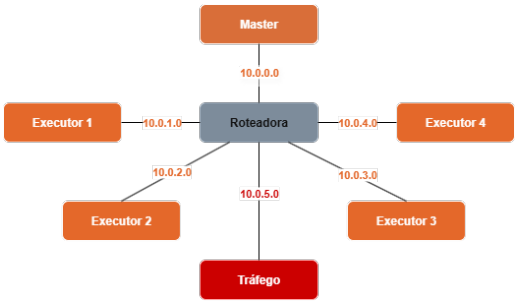

Figura 1. Topologia de rede do ambiente virtualizado.

No roteador, o gerenciamento de filas foi configurado com a disciplina *Random Early Detection* (RED) [Floyd and Jacobson 1993] (*limit* 2500000, *min* 500000, *max* 1000000), operando sob uma classe *Hierarchical Token Bucket* (HTB) com largura de banda limitada a 10 Gbit/s. Adicionalmente, aplicou-se um atraso artificial de 2 ms em todas as interfaces para emular a latência característica de links físicos de baixa latência.

Na *master*, o Protocolo de Controle de Transmissão (TCP) foi otimizado para as operações de *shuffle*: o mecanismo *slow start* foi desabilitado e as janelas iniciais (*initcwnd initrwnd*) foram elevadas para 850 e 1000 Maximum Segment Size (MSS), respectivamente, um aumento significativo frente ao padrão de 10 MSS. Essa configuração teve como objetivo forçar a rede a operar rapidamente em sua capacidade máxima, evidenciando de forma mais clara o impacto do congestionamento e o consequente fenômeno de *starvation*.

3. Metodologia

A metodologia experimental foi desenhada para identificar os efeitos do *starvation*, comparando o desempenho de duas aplicações com perfis distintos de comunicação. A primeira é uma aplicação customizada que processa 100 milhões de registros sintéticos, caracterizada pelo uso intensivo da rede devido à geração de um alto volume de pacotes pequenos. Já a segunda é a *GroupByTest*, integrante da suíte padrão do Spark, que apresenta uso intensivo de CPU e operações de *shuffle* com pacotes de maior tamanho e em menor quantidade.

Cada aplicação foi submetida a dois cenários distintos: "Sem Tráfego", para estabelecer uma linha de base com a rede ociosa, e "Com Tráfego", onde um fluxo contínuo foi gerado via iPerf para induzir congestionamento. Para garantir a validade estatística dos resultados, cada cenário foi executado 10 vezes. A avaliação foi conduzida em duas fases: execuções isoladas (utilizando todos os recursos do cluster) e simultâneas (recursos divididos para forçar competição por banda).

Para correlacionar o estado da rede com o desempenho das aplicações, a coleta de métricas foi estruturada em duas frentes complementares:

- **Métricas de rede:** focada na quantificação do congestionamento por meio de um *script* na VM roteadora. Utilizando o comando `tc -s qdisc show dev <interface>`, registrou-se o *backlog* da fila (em *bytes* e pacotes) com granularidade de 0,1 segundo.
- **Métricas de aplicação:** empregou um *listener* customizado anexado a cada *job* do Spark. Para cada tarefa concluída, registraram-se o tempo total de execução (*Task Time*), o tempo de processamento efetivo (*CPU Time*), o volume de dados

transferidos e o Flow Completion Time (FCT) [Dukkipati and McKeown 2006] para as operações de leitura e escrita

4. Resultados

A Figura 2 ilustra o comportamento do *backlog* da fila durante a execução isolada da aplicação customizada. O cenário "Sem Tráfego" mantém a fila próxima de zero, enquanto o cenário "Com Tráfego" apresenta um aumento drástico, com picos superiores a 700 kB. Este comportamento, que também induziu o fenômeno de *starvation* na aplicação *GroupByTest* e na execução simultânea, demonstra que o tráfego concorrente estabeleceu a rede como um recurso escasso e um gargalo de desempenho (*bottleneck*).

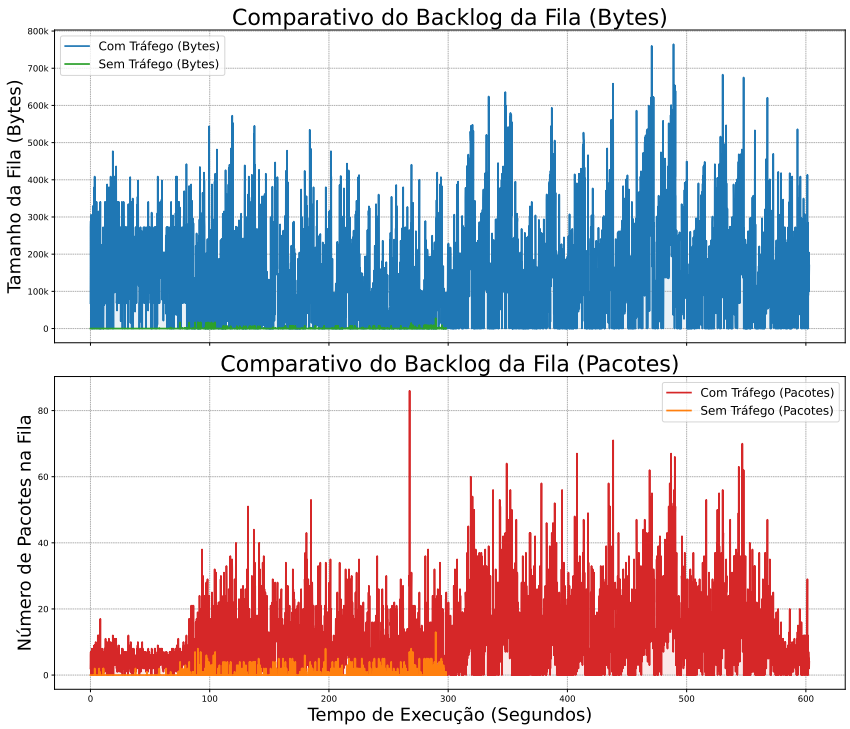

Figura 2. Comportamento do backlog da fila durante a execução da aplicação customizada, apresentando o volume em Bytes (gráfico superior) e o número de Pacotes (gráfico inferior).

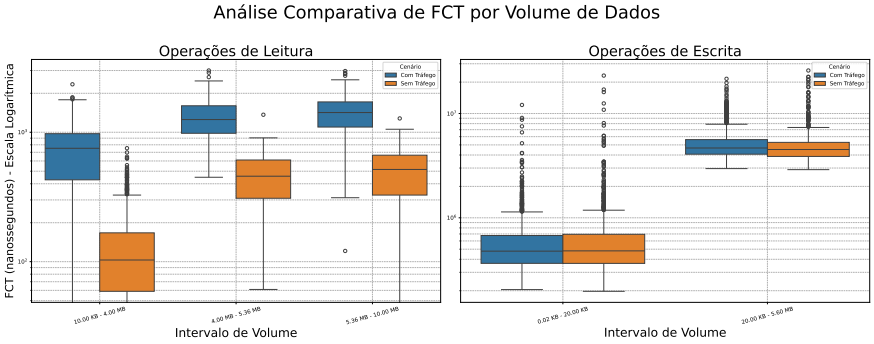

Figura 3. Comparativo *boxplot* de FCT (ns) por intervalo de volume de dados (MB) das tarefas durante a execução da aplicação customizada

O impacto direto desse gargalo no nível da aplicação é evidenciado pela métrica de FCT (Figura 3). Com a rede sobrecarregada, a duração da comunicação aumentou em ordens de magnitude. Esse cenário reflete um estado de *starvation* induzido por I/O, no qual o fluxo de dados não é suficiente para alimentar continuamente os executores do Spark.

Os dados quantitativos corroboram essa análise: o tempo médio de execução total da aplicação saltou de 295,4 s (σ = 24,9 s) no cenário ocioso para 622,6 s (σ = 69,2 s) sob restrição de rede. Apesar desse acréscimo superior a 110% no tempo total, o monitoramento granular revelou que o tempo de utilização de CPU (*CPU Time*) permaneceu inalterado em ambos os cenários. Este resultado é fundamental, pois confirma a hipótese inicial: o desempenho foi limitado pela subutilização da CPU enquanto o sistema aguardava a chegada de pacotes, caracterizando o fenômeno de *starvation*.

5. Conclusão

Este trabalho investigou como o congestionamento na rede induz o fenômeno de *starvation* no nível da aplicação, indicando que a contenção de rede atua como um severo gargalo de desempenho. Os resultados obtidos demonstram que, sob condições de tráfego concorrente, o tempo total de execução da aplicação sofreu um acréscimo superior a 110%. No entanto, observou-se que o tempo de uso efetivo de CPU pelas tarefas permaneceu inalterado, evidenciando que o sistema permanece ocioso aguardando dados da rede (I/O *bound*). A análise granular do *backlog* da fila e do *Flow Completion Time* (FCT) permitiu correlacionar picos de congestionamento de até 700 kB com o atraso direto no progresso da aplicação, validando a hipótese de *starvation* induzido pela infraestrutura.

Fica evidente que a subutilização de recursos computacionais em *clusters* Spark não decorre necessariamente de falta de hardware, mas de uma gestão ineficiente da camada de comunicação. Como trabalhos futuros, os experimentos serão expandidos para uma arquitetura física e distribuída, assim eliminando a camada de virtualização e os parâmetros artificiais impostos à rede.

Agradecimentos: Este trabalho recebeu apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Fundação de Amparo à Pesquisa e Inovação (FAPESC), desenvolvido no Laboratório de Processamento Paralelo e Distribuído (LabP2D).

Referências

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. (2010). Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74.

Arun, V., Alizadeh, M., and Balakrishnan, H. (2022). Starvation in end-to-end congestion control. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 177–192.

Dukkipati, N. and McKeown, N. (2006). Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM Computer Communication Review*, 36(1):59–62.

Floyd, S. and Jacobson, V. (1993). Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413.

Salloum, S., Dautov, R., Chen, X., Peng, P. X., and Huang, J. Z. (2016). Big data analytics on apache spark. *International Journal of Data Science and Analytics*, 1:145–164. DOI: https://doi.org/10.1007/s41060-016-0027-9.