Processamento Distribuído de Operações *Hash Join* em *Switches* Programáveis: Uma Análise via Modelo de Custo *

Marisa S. Franco, Simone Dominico, Tiago R. Kepe, Luiz C. P. Albini, Eduardo C. de Almeida, Marco A. Z. Alves

[1]Departamento de Informática – Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brasil

{masf18,sdominico,trkepe,albini,eduardo,mazalves}@inf.ufpr.br

Abstract. *The query processing cost in distributed database systems is directly associated with transferring data cost over the network. The Software-Defined Wide-Area Network (SD-WAN) allows (re)program network devices via software. The programmability of this architecture provides new possibilities for dynamically manage the network topology, also enabling the data processing on these devices. This paper presents an evaluation of the distributed processing of the hash join operation to the database and network devices, using an cost model. Our results show that processing the hash join operation using network switches could achieve competing results compared to traditional servers with similar data traffic.*

Resumo. *O custo do processamento de uma consulta em sistemas de banco de dados distribuídos liga-se diretamente ao custo da transferência de dados na rede. A Software-Defined Wide-Area Network (SD-WAN) é uma tecnologia que permite (re)programar dispositivos de rede via software. Sua programabilidade proporciona novas possibilidades para gerenciar topologias de forma dinâmica, possibilitando ainda o processamento de dados nesses dispositivos. Este artigo avalia o processamento distribuído da operação hash join em switches de rede, a partir de um modelo de custo. Os resultados mostram que o processamento dessas operações em switches de rede alcança desempenho comparável ao processamento tradicional em servidores com um tráfego de dados similar.*

1. Introdução

Na última década, é nítido o crescimento na quantidade de dados criados, capturados, copiados e consumidos em todo o mundo. Estima-se que, em 2020, esse volume de dados atingiu $64{,}2$ zettabytes, e que a geração global de dados chegue a 79 zettabytes apenas em 2021 [Holst 2021] – e a impressionantes 181 zettabytes em 2025. Paralelamente, a capacidade computacional de coleta, processamento e armazenamento de dados aumentou significativamente no mesmo período. Ainda segundo Holst [Holst 2021], a base instalada da capacidade de armazenamento de dados atingiu $6{,}7$ zettabytes em 2020 e deve crescer a uma taxa composta anual de $19{,}2\%$ entre 2020 e 2025.

Nesse contexto, as pesquisas sobre bancos de dados distribuídos ganham mais relevância. Um dos principais focos dos estudos é o custo de comunicação para transferência dos dados. Trata-se de um gargalo importante em sistemas de banco de dados nas

*Trabalho suportado por CNPq, CAPES e Instituto Serrapilheira (grant Serra-1709-16621).

quais a comunicação entre servidores é exigida, como durante a realização de operações de *hash join*. Essa operação une duas tabelas do banco de dados a partir de um atributo comum de relacionamento, chamado também de atributo de *join*.

Além disso, com a expansão dos serviços na Internet, a tradicional *Wide Area Network* (WAN) perde espaço para a tecnologia *Software-Defined Wide-Area Network* (SD-WAN) a qual se apresenta mais flexível. Tal qualidade é essencial para alguns serviços, como servidores de *cloud*. A SD-WAN busca reduzir a complexidade da WAN por meio da virtualização de serviços. Dessa forma, existe uma flexibilidade maior na administração, permitindo a programabilidade da rede via *software* desenvolvido para essa finalidade [Shin et al. 2012]. A programabilidade da arquitetura SD-WAN proporciona novas possibilidades para mitigar latências de forma dinâmica. Na literatura, trabalhos como [Yang et al. 2019] analisam os avanços em cada camada de rede proporcionado pela SD-WAN. Já o uso da programabilidade dos elementos da rede para acelerar consultas analíticas foi discutido em [Lerner et al. 2019] e [Lerner et al. 2020], nos quais os autores descrevem oportunidades alcançadas para sistemas de banco de dados a partir da evolução das tecnologias de rede.

Diante desse cenário, o objetivo deste trabalho é avaliar o processamento distribuído da operação de *join* em rede. Por meio de uma análise baseada em modelo de custo, será estimada a execução do algoritmo *hash join* em dispositivos de rede. A análise realizada contempla diferentes cenários de execução de *hash join* distribuído, incluindo a execução da operação em um *switch* de rede. Assim, buscou-se avaliar o potencial do processamento de dados em dispositivos de rede para ganho de desempenho.

Os resultados mostram que o modelo de processamento de dados em *switches* apresenta performance equiparável com processamento tradicional em servidores com um tráfego de dados similar. Porém, os dois melhores tempo de execução, entre os seis cenários apresentados, foram obtidos por cenários em que há menos transferência de dados e o processamento acontece em paralelo nos servidores locais. A pesquisa demonstrou que o paralelismo no processamento é um ponto crítico extremamente relevante para o ganho de desempenho em tempo de execução nas consultas que envolvem processamento distribuído de operações *hash join*. Portanto, o presente estudo cria uma base para pesquisas futuras que avaliem o potencial do processamento de dados em dispositivos de rede.

O restante deste trabalho está organizado da seguinte forma: os trabalhos relacionados são discutidos na Seção 2; a Seção 3 apresenta uma visão sobre banco de dados distribuídos e a operação de *hash join*; a metodologia e a modelagem teórica são discutidas na Seção 4; a Seção 5 apresentação a análise dos resultado; por fim, a Seção 6 apresenta a conclusão e elenca possíveis desdobramentos para trabalhos futuros.

2. Trabalhos Relacionados

Nos últimos anos, diversas abordagens apresentadas buscam soluções para acelerar o processamento de consultas distribuídas através da rede. Em [Narayana et al. 2017], os autores utilizam um *switch* de rede programável como um *cache* para operações de agregação com o objetivo de detectar possíveis problemas – como congestionamento da rede –, visando fornecer uma solução rápida para que os problemas não se propaguem. Contudo, apesar de buscar melhorias para operações de consulta por meio da rede, os autores não

abordam a tecnologia SD-WAN.

Já em [Xiong et al. 2014], os autores discutem a utilização de *Software Defined Network* (SDN) para coletar informações para otimizações de planos de consulta. Outros trabalhos, como [Binnig et al. 2016, Salama et al. 2017], buscam utilizar os avanços de velocidade de rede para propor novos algoritmos de operação *join* distribuídos. Esses algoritmos aproveitam a velocidade de rede de forma mais eficiente. Os autores consideram ainda que a rede não é mais um gargalo relevante para a execução de consultas distribuídas. Em contraste, a presente pesquisa busca utilizar a programabilidade disponível na SD-WAN para analisar as possíveis vantagens do processamento de operação de *join* em um *switch*. Além disso, no presente trabalho chegou-se a mesma conclusão desses correlatos, uma vez que o tempo de comunicação pareceu de baixa influência no resultado final dos testes apresentados.

Os autores de [Jin et al. 2018] apresentam um mecanismo que coordena o armazenamento dos dados em *switches* programáveis. O objetivo principal é executar todo plano de consulta dentro desses dispositivos de rede. Em [Lerner et al. 2019], os autores buscam utilizar a programabilidade disponível no *switch* para melhorar a execução de consultas analíticas. É apresentado nesse trabalho um mecanismo com componentes diferentes que tem por objetivo agendar operações de consultas para acontecer no *switch* e também observar as comunicações na rede. Além disso, os autores apresentam um novo conjunto de operações para acelerar o processamento usando *switches*.

Atualmente, o trabalho de [Lerner et al. 2020] discute e apresenta os desafios e oportunidades de aceleração de consultas utilizando dispositivos de rede. Os autores argumentam que, melhorando os algoritmos, é possível aproveitar novas tecnologias de rede e reduzir a movimentação de dados pela rede. Tal pesquisa contribui diretamente para evidenciar as possibilidades do processamento de operações de *join* no *switch* e analisa, com diferentes tecnologias de rede, a possibilidade de ganho por meio de um modelo teórico.

3. Bancos de Dados Distribuídos e Operação *Hash Join*

Um sistema de gerência de banco de dados distribuídos gerencia uma coleção de banco de dados espalhada em diversos servidores conectados através de uma rede de computadores. Cada servidor na rede, também chamado de nó, é responsável por armazenar e processar dados. A arquitetura geral desse tipo de sistema baseia-se, principalmente, no modelo "cliente-servidor", na qual o nó cliente envia uma consulta para nós servidores. O processamento distribuído de consulta costuma ser colaborativo onde os nós precisam trocar dados entre si para computar o resultado final. A transferência de dados costuma ser realizada por meio de uma rede dedicada para suportar um alto fluxo de dados permutado entre dispositivos da rede – tais como *switches* de rede –, que atuam como agentes passivos, realizando somente operações de gerenciamento e entrega de pacotes na rede.

Entretanto, o custo do processamento de uma consulta em bancos de dados distribuídos está diretamente ligado ao custo da transferência de dados na rede [Kossmann 2000]. Assim, diferentes algoritmos de processamento de consulta concentram-se na geração de planos de consulta que minimizem a quantidade de dados trafegados. As consultas que efetuam a operação de *join* utilizam *semi-joins* e *bloom filters* para minimizar a transferência dos dados na rede [Polychroniou et al. 2014].

Uma operação de *join* em uma consulta é um modo de recuperar dados de

várias tabelas de banco de dados relacionais por meio de combinações lógicas entre as tabelas. Em um ambiente distribuído, podem existir várias abordagens de *join* [Valduriez and Gardarin 1984, Huang et al. 2014, Polychroniou et al. 2018]: (i) *Broadcast joins*, em que uma tabela é replicada e enviada a todos os nós de processamento, cada um dos quais com parte de uma tabela maior. O caso de uso canônico é uma grande tabela com uma pequena tabela de referência; (ii) *Hash joins* nos quais, no momento do armazenamento dos dados, o particionamento entre os servidores é feito via *hash*. O *join* entre os dados das relações maior e menor pode ser feita localmente, sem qualquer transferência de dados entre os nós, porque todos os dados relevantes já estão co-localizados; (iii) Uma operação de *hash join* completa, em que os dados dos servidores são usados na criação de tabelas *hash* e essas são transferidas aos outros nós para conclusão do processamento nos servidores.

A cardinalidade tem um impacto bem grande nas operações de *join*, sendo um dos fatores levados em consideração pelo otimizador para a escolha do algoritmo de junção (*hash*, *merge*, *NL*, entre outros). Na presente pesquisa, optou-se por focar nas operações de *hash join* visando entender o impacto de tabelas de alta cardinalidade que cabem em memória. Além disso, estudos anteriores [Kepe et al. 2019] mostraram que o operador de *hash join* consta como um dos quatro operadores que mais contribuem para o tempo de execução e memória utilizada do *benchmark* TPC-H [Council 2020] em bancos de dados colunares. Outro fato importante é a ampla utilização de *hash joins* no processamento distribuído de consultas devido ao particionamento dos dados e às abordagens de transmissão das relações [Valduriez and Gardarin 1984, Huang et al. 2014, Polychroniou et al. 2018].

Hash join é o algoritmo comumente utilizado para efetuar operações de *join* nas implementações de sistemas de gerenciamento de banco de dados relacionais, os quais usam particionamento via *hash*. Conforme ilustrado no Código 1, o algoritmo de *hash join* possui duas fases [Blanas et al. 2011], a construção e a análise. Na fase de construção, a menor relação (a relação externa R) é percorrida e as chaves de interesse são armazenadas em uma tabela *hash* (HT), aplicando-se uma função *hash* $h1$. Na fase de análise, a relação maior (a relação interna S) é percorrida, aplicando-se a mesma função *hash* $h1$, em busca de correspondências para os valores mapeados na tabela *hash* e a operação de *hash join* é realizada para gerar a relação de saída da consulta.

Código 1. Pseudocódigo do algoritmo básico de *hash join*. frame

```
1 build hash table HT for R
2 foreach tuple s in S
3    output, if h1(s) in HT
```

4. Metodologia e Modelagem Teórica

Para a avaliação do processamento distribuído de *hash join*, foram adotados alguns cenários de transmissão de dados baseados em topologia estrela, com dois servidores de banco de dados e um cliente conectados por um *switch*, como mostra a Figura 1.

Foram escolhidas ainda simplificações das consultas *query-10* e *query-11* do *benchmark* TPC-H [Council 2020], que incluem apenas a operação de *hash join*, conforme os Códigos 2 e 3 descritos em SQL.

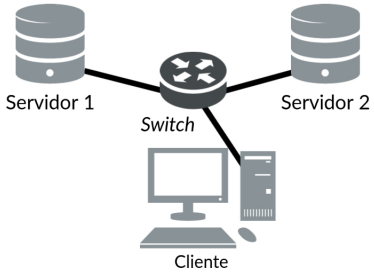


Figura 1. Topologia estrela adotada para a definição de cenários de processamento distribuído de *hash join*.

Código 2. Query-10 do TPC-H.

```
query 10:
SELECT C_CUSTKEY, C_NAME
FROM CUSTOMER, ORDERS
WHERE C_CUSTKEY = O_CUSTKEY
```

Código 3. Query-11 do TPC-H.

```
query 11:
SELECT PS_PARTKEY
FROM PARTSUPP, SUPPLIER
WHERE PS_SUPPKEY = S_SUPPKEY
```

O TPC-H é um *benchmark* de apoio à decisão, que engloba um conjunto de consultas *ad-hoc* orientadas para os negócios e modificações de dados simultâneas. Ele ilustra sistemas de apoio à decisão que examinam grandes volumes de dados e executam consultas com alto grau de complexidade.

As consultas foram implementadas em C[1] e os dados foram armazenados no formato DSM (*Decomposition Storage Model*). Foram implementados dois algoritmos de cada consulta - *query-10* e *query-11* - , ambos de execução linear. Para a criação das tabelas *hash*, foram usadas as funções FNV-1a [Fowler et al. 2013] e MurmurHash3 [Appleby 2016]. Ambas mostraram, em trabalhos anteriores da literatura [Estébanez et al. 2014] [Scheidt de Cristo et al. 2019], excelente desempenho em termos de espalhamento de dados e tempo de execução. Foram considerados também seis cenários de processamento distribuído de *hash join* detalhados na Tabela 1. A tabela traz as etapas de processamento, dividido em armazenamento inicial dos dados, comunicação e processamento para cada um dos cenários avaliados.

Sendo M o número de posições na tabela *hash* e N o número de chaves da tabela de símbolos, o fator de carga (*load factor*) é $\alpha = N/M$. A função de *hashing* produz uma colisão quando duas chaves diferentes têm o mesmo valor *hash* e, portanto, são mapeadas para a mesma posição da tabela *hash*. Nos experimentos, o critério de escolha para o tamanho das tabelas *hash* (M) foi um número primo imediatamente abaixo de uma potência de dois, que propiciasse um fator de carga entre 5 e 10, com objetivo de favorecer um melhor espalhamento dos dados, evitando colisões [Sedgewick 1998].

Cada cenário foi avaliado com três cargas de trabalho (1 GB, 10 GB e 100 GB) e usando quatro tecnologias de rede (*Ethernet* 100 Gb, *Ethernet* 200 Gb, *Ethernet* 400 Gb e *InfiniBand* HDR 12×), com comunicação *full-duplex*. O critério de escolha das tecnologias de rede foi temporal: todos os padrões analisados entraram em uso na última década (em 2014, 2017, 2017 e 2021, respectivamente). Considerando os cenários de processamento em dispositivo de rede (cenários 3 e 5), foram avaliadas três velocidades

[1] *Hash join*: https://github.com/marisasel/hashjoin

Tabela 1. Cenários analisados de processamento distribuído de *hash join*.

Cenário	(A) Armazenamento inicial, (C) Comunicação e (P) Processamento
1	A: Servidor 1 possui relação maior. Servidor 2 possui relação menor. C: Relação maior é transmitida. P: Processamento no servidor 2.
2	A: Servidor 1 possui relação maior. Servidor 2 possui relação menor. C: Relação menor é transmitida. P: Processamento no servidor 1.
3	A: Servidor 1 possui relação maior. Servidor 2 possui relação menor. C: Ambas relações são transmitidas ao *switch*. P: **Processamento no *switch***.
4	A: Cada servidor possui uma metade das duas relações P: Cada servidor processa a fase de análise utilizando a relação menor. C: Compartilhamento das tabelas *hash* entre servidores. P: Cada servidor faz parte da análise, usando as duas *hashes* e sua metade da relação maior.
5	A: Cada servidor possui uma metade das duas relações. C: Ambas relações são transmitidas ao *switch*. P: **Processamento no *switch***.
6	A: Cada servidor possui uma metade das duas relações, e dados relevantes co-localizados. P: Processamento local nos servidores de cada uma das metades.

de processamento do *switch*: igual às CPUs dos servidores, 5% e 10% mais lentas do que os servidores. Dessa maneira, foi possível avaliar a influência desse processamento no tempo final de entrega dos resultados.

As estimativas desta pesquisa foram alimentadas com dados de execução real. Nos experimentos, foi utilizada uma máquina com dois soquetes, cada um deles com um *Intel Xeon Silver* 4114 (com microarquitetura *Skylake*). Cada soquete *Xeon* tem dez núcleos com cache L1 (I + D) privado (32 KB), cache L2 privado (1 MB) e um cache L3 compartilhado (14 MB). A máquina possui 128 GB de memória principal DDR-4 e 14 TB de disco, executando o sistema operacional Ubuntu, versão $18.04.01$ LTS. Para obtenção dos tempos de execução, executou-se $20\times$ cada um dos 24 experimentos-base. Cada um dos experimentos combinou os diversos fatores nos níveis definidos a seguir: (i) as *query-10* e *query-11*; (ii) funções *hash* MurmurHash3 e FNV-1a; (iii) tamanho da carga de trabalho (1 GB, 10 GB e 100 GB de dados); (iv) algoritmo regular e algoritmo simulando paralelismo da rede.

Para cada experimento-base foram realizadas 20 repetições, totalizando 480 execuções. O coeficiente de variação (CV) é usado para expressar a variabilidade dos dados tirando a influência da ordem de grandeza da variável. Ele é obtido dividindo-se o desvio padrão pela média dos dados e expresso por um número puro entre 0 e 1. Quanto menor o CV, mais homogêneo é o conjunto de dados. Usualmente, conjuntos de dados com CV entre 0 e $0{,}30$ apresentam pouca variabilidade. Foram observados os CVs dos tempos das fases de construção e análise dos resultados de cada um dos 24 experimentos-base - cujas médias são usadas nos cálculos de custo do modelo adotado. Não houve nenhum CV acima de $0{,}30$ nas baterias de execução de cada experimento-base para esses parâmetros. Para o tempo de construção, o valor mínimo do CV foi $0{,}001$ e o valor máximo foi $0{,}206$. Já para o tempo de análise, o menor CV foi $0{,}003$ e o maior foi $0{,}075$. Assim, como as amostras dos resultados dos parâmetros usados na análise mostraram-se

extremamente homogêneas para cada bateria de testes, justificam-se as 20 execuções de cada experimento-base.

O modelo de custo proposto neste trabalho é inspirado na máquina teórica e modelo LogP, de [Culler et al. 1993] – um modelo tradicional em programação paralela. A Tabela 2 descreve, matematicamente, quais foram os cálculos feitos para estimar os tempos de execução total em cada um dos seis cenários de processamento distribuído de *hash join* considerados. Os algoritmos implementados para simular os cenários 4 e 6, nos quais há processamento paralelo distribuído em dois servidores, diferem dos algoritmos-base por armazenar duas relações com cerca de metade do tamanho das relações regulares, assim como relações de saída com metade do tamanho das usadas originalmente.

Tabela 2. Cálculo de custo utilizado em cada cenário.

Cenário	Cálculo de tempo total
1	$PD + ((N + R)/BW)$
2	$PD + ((n + R)/BW)$
3	$PD + (((max(N; n)) + R)/BW)$
4	$PD + (((H/2) + R)/BW)$
5	$PD + ((N/2 + n/2 + R)/BW)$
6	$PD + (R/BW)$

PD: tempo de processamento dos dados:
Cenários 1, 2, 3 e 8, análise e construção sequenciais;
Cenários 4 e 6, análise e construção paralelas;
N: Tamanho da relação maior.
n: Tamanho da relação menor.
H: Tamanho da tabela *hash* - espalhamento perfeito.
R: Tamanho da relação de saída. **BW:** Largura de banda.

Obs: "Tamanho" refere-se à número de tuplas da tabela multiplicado pela soma dos tamanhos das colunas pertinentes à consulta.

Na fase de construção desses algoritmos que emulam paralelismo de rede, há dois laços independentes, que percorrem cada um uma metade da relação menor para construir as tabelas *hash* **A** e **B**. Na fase de análise, também há dois laços independentes, cada um percorrendo uma metade da relação maior, buscando correspondências tanto na tabela *hash* **A** quanto na **B** para gerar a saída, feita com duas chamadas da função de impressão dos resultados. Dessa forma, nas execuções que simulam paralelismo em dois servidores, as médias dos tempos de execução da fase de construção foram obtidas a partir do valor máximo entre as duas fases de construção executadas pelo algoritmo em cada bateria de testes. O mesmo foi feito para a fase de análise.

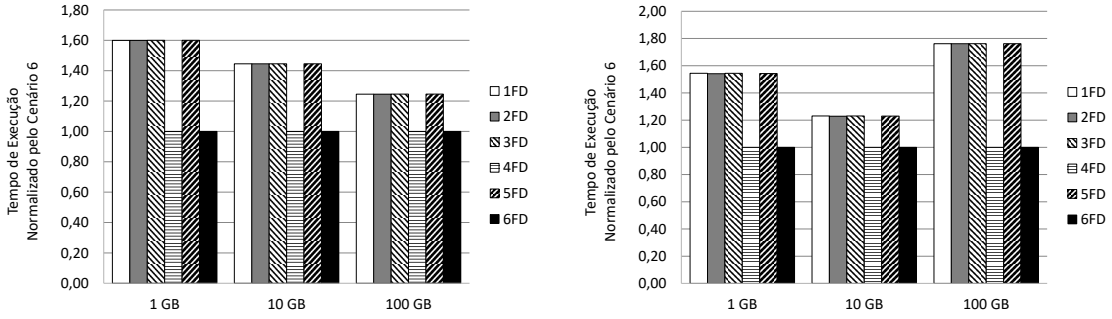
Foram feitas simplificações para definição do modelo de custo. Para os cálculos de transmissão dos dados, consideraram-se apenas as capacidades máximas teóricas da largura de banda de cada tecnologia (100 Gb/s, 200 Gb/s, 400 Gb/s e 600 Gb/s, respectivamente, em *half-duplex*). Ainda em relação às limitações do modelo de custo proposto, o modelo trabalha apenas com o envio do volume bruto de dados, sem considerar cabeçalhos e divisão das mensagens em pacotes (o que resultaria em variação média menor que 2% do volume total de dados enviados). Também não foram consideradas nos cálculos as latências de rede e as distâncias entre os nós. Nas estimativas realizadas nesta pequisa, a latência inicial de comunicação é negligenciável, uma vez que são realizados poucos envios, cada um deles com grandes volumes de dados. Considerou-se o espalhamento perfeito dos dados nas tabelas *hash* para mensurar o volume de dados transferido no cenário 4.

5. Análise dos Resultados

Nesta Seção, são apresentados os resultados das consultas *query-10* e *query-11* nos cenários descritos na Tabela 1. Seguindo a modelagem teórica apresentada anteriormente,

foram avaliados os tempos de execução. Optou-se por explorar os melhores resultados para cada consulta, de acordo com a função de espalhamento. Para ambas as consultas, os melhores resultados preliminares foram obtidos com a função MurmurHash3. Assim, os resultados aqui apresentados estão relacionados às execuções com tal função.

As Figuras 2(a) e 2(b) mostram o "melhor cenário": *InfiniBand* HDR 12×, *switch* 100%, com os tempos de execução de cada carga de trabalho (1 GB, 10 GB e 100 GB) normalizados pelo seu cenário 6. Em todos os experimentos, o cenário 6 apresentou o melhor desempenho em relação ao tempo de execução para os diferentes tamanho de carga de trabalho nas duas consultas avaliadas.



(a) *Query-10* (b) *Query-11*

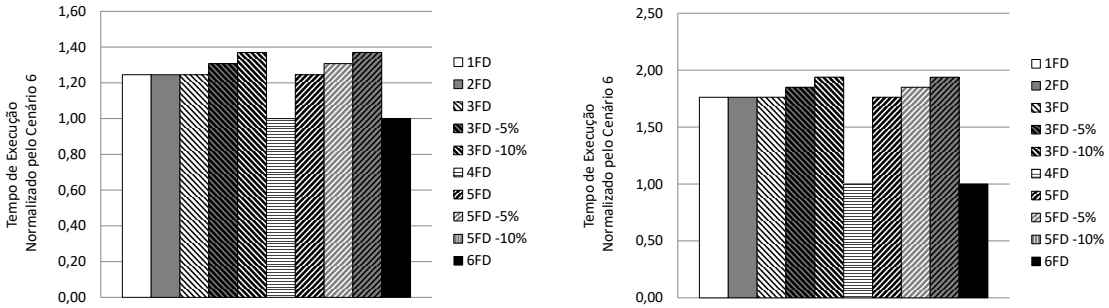
Figura 2. Tempos conforme variação da carga de trabalho, usando InfiniBand HDR com *switch* CPU 100% e função MurmurHash3. Valores normalizados pelo seu cenário 6.

Tais resultados evidenciam não só a importância da execução das consultas em paralelo em dois servidores para o bom desempenho final, como também o papel da transferência de dados na composição do custo total de processamento distribuído de *hash join*. O cenário 6 é o que transfere a menor quantidade de dados: apenas os dados da relação de saída de consulta. Ao mesmo tempo, possui o maior sobrecusto de armazenamento e sincronização de dados nos servidores, pois os dados estão co-localizados, ou seja, replicados.

Em ambas as consultas, o segundo melhor desempenho foi obtido com o cenário 4. Tal cenário realiza o processamento em paralelo em dois servidores, assim como o cenário 6, e trafega a segunda menor quantidade de dados: tabela *hash* e relação de saída. Na *query-11*, o terceiro melhor desempenho foi obtido pelo cenário 2. Enquanto os cenários 1 e 2 equiparam-se na *query-10*, pois nela são solicitados mais dados (colunas) provenientes da relação menor – cujos dados são transferidos no cenário 2.

Nas Figuras 3(a) e 3(b), a combinação dos parâmetros apresenta o impacto das diferentes velocidades do *switch* nos resultados. Como observado em ambas as consultas, um processamento feito no *switch* com capacidade 5% menor afeta o tempo total de processamento dos cenários 3 e 5 em $4{,}999\%$, enquanto um realizado no *switch* com capacidade 10% menor aumenta em $9{,}998\%$ quando avaliados os cenários com carga de trabalho máxima (100 GB) e transferência de dados mais rápida (*InfiniBand* HDR 12×). Tais resultados refletem o peso das fases de construção e fase de análise na composição total do tempo de processamento com essas variáveis. Em ambas as consultas, para os seis cenários de processamento distribuído de *hash join* analisados, pode-se ver a alta

influência do tempo de processamento no tempo final.

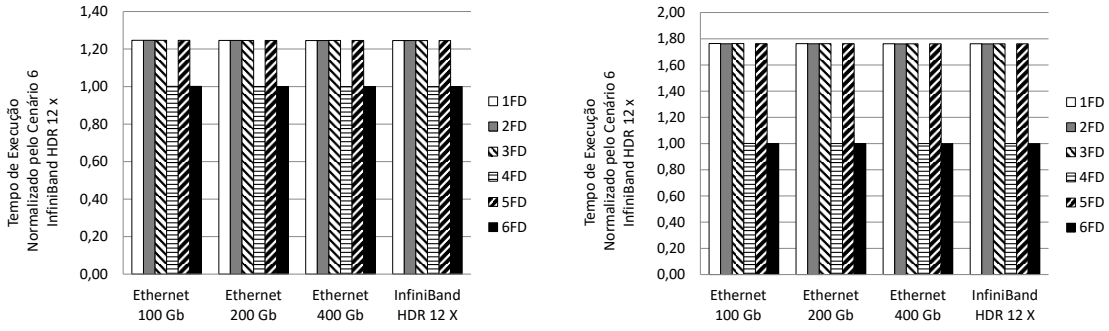


(a) *Query-10* (b) *Query-11*

Figura 3. Tempos conforme variação na capacidade de processamento do *switch*, usando InfiniBand HDR com carga de trabalho de 100 GB e função MurmurHash3. Valores normalizados pelo seu cenário 6.

Em geral, o desempenho em relação ao tempo de execução dos cenários 3 e 5, quando com capacidade do *switch* idêntica à das CPUs dos servidores, é igual ao dos cenários 1 e 2. Isso porque a quantidade de dados transferida é a mesma e não há paralelismo no processamento. No entanto, é importante observar que a capacidade de processamento do *switch* influencia no ganho de desempenho dos cenários estudados. Se fosse considerado no modelo um *switch* que possuísse uma capacidade de processamento superior a das CPUs dos servidores e/ou um cenário com processamento paralelo nesse dispositivo de rede, existiria ganho de desempenho nos tempos de execução dos cenários 3 e 5 em relação aos cenários 1 e 2.

Por fim, as Figuras 4(a) e 4(b) mostram o impacto da variação da capacidade de rede. O fato do modelo de custo proposto na presente pesquisa não considerar atrasos de rede e os custos de empacotamento/desempacotamento das mensagens, bem como as fases de construção e análise serem majoritárias na composição do tempo de processamento, tornou o desempenho nas diferentes tecnologias de rede muito próximo, como pode ser visto nos resultados apresentados.



(a) *Query-10* (b) *Query-11*

Figura 4. Tempos conforme variação na capacidade de vazão da rede. Carga de trabalho de 100 GB com *switch* CPU 100% e função MurmurHash3. Valores normalizados pelo seu cenário 6 da InfiniBand HDR.

A Tabela 3 mostra a porcentagem que cada etapa analisada no modelo de custo possui em relação ao tempo total de processamento de consulta no pior cenário (1) e no

melhor cenário (6), usando *Ethernet* 100 Gb. Utilizou-se a tecnologia de rede mais lenta para apresentar esses dados, pois, à medida que a velocidade de rede aumenta, o tempo de transferência de dados torna-se proporcionalmente menos significante.

Tabela 3. Porcentagem do tempo de cada etapa para a *query-10* e *query-11*. Considerando o pior cenário (1) e melhor cenário (6), com *Ethernet* 100 Gb.

Cenário	Consulta	Carga de trabalho (GB)	Construção (%)	Análise (%)	Transferência (%)
1	10	1	3,71	95,49	0,80
6	10	1	2,57	96,28	1,16
1	10	10	5,88	93,88	0,24
6	10	10	1,99	97,70	0,31
1	10	100	7,36	92,51	0,13
6	10	100	4,73	95,11	0,15
1	11	1	2,00	95,94	2,06
6	11	1	1,55	97,38	1,07
1	11	10	1,75	96,92	1,33
6	11	10	0,97	98,48	0,55
1	11	100	1,49	98,37	0,14
6	11	100	0,43	99,49	0,08

É possível notar que o peso do tempo de transferência de dados no tempo total de processamento em uma mesma consulta e cenário diminui conforme os tamanho da carga de trabalho aumenta. Salienta-se ainda que, embora a menor quantidade dados transferida conte a favor do melhor desempenho do cenário 6, ao se analisar a fatia de tempo ocupada por cada trecho da execução das consultas, fica evidente que o ganho em tempo de execução deve-se também ao fato de cenário 6 realizar tanto o processamento da fase de construção quanto o da fase de análise em paralelo, em dois servidores.

6. Conclusões e Trabalhos Futuros

Neste artigo, foi apresentada uma análise de diferentes cenários de execução de *hash join* distribuído, incluindo execução da operação em um *switch* de rede, a partir de um modelo de custo. A escolha pela topologia estrela (com dois servidores, um *switch* e um cliente) e pela operação de *hash join* mostrou-se exitosa porque variações básicas de distribuição dos dados e processamento foram capazes de diferenciar os desempenhos e indicar quais os melhores cenários a serem explorados para avançar na pesquisa.

O melhor desempenho tanto nos experimentos das consultas avaliadas foi alcançado pelo cenário 6, em que cada servidor possui uma metade das duas relações, com dados relevantes co-localizados, e o processamento é feito localmente nos dois servidores. Esse é o cenário que transfere menos dados ao mesmo tempo que paraleliza a execução. Entretanto, vale ressaltar que, para manter os dados co-localizados, existe um custo para particionar as tabelas nos servidores que não foi considerado na execução das consultas. Além disso, a execução paralela pode ter um impacto negativo no desempenho final de sistemas com alta demanda de consultas distintas.

Dado que, historicamente, as velocidades de transmissão das tecnologias de rede não crescem na mesma proporção que o volume de dados transferidos anualmente, a latência na transferência de dados segue sendo um gargalo importante em sistemas distribuídos. Porém, vale ressaltar que o modelo e os resultados apresentados na presente

pesquisa demonstram que o paralelismo no processamento é um ponto crítico extremamente relevante para o ganho de desempenho em tempo de execução nas consultas que envolvem processamento distribuído da operação de junção em bancos de dados.

Quanto aos riscos e desvantagens de se usar redes programáveis para o processamento distribuído de operações *hash join*, o ambiente de rede considerado foi o privado. Entretanto, questões ligadas à tolerância a falhas e limitações de memória nesses sistemas devem ser discutidas em trabalhos futuros. Com o desenvolvimento deste trabalho, notou-se que o uso exclusivo de processamento em dispositivos de rede poderá levar a uma centralização, criando um gargalo no processamento – conforme discutido acima. Entretanto, entende-se que processamentos de consultas recorrentes podem se beneficiar do processamento em redes programáveis.

Espera-se que os resultados aqui apresentados contribuam para um conhecimento mais amplo sobre diferentes estratégias de processamento distribuído de dados e o potencial do processamento de dados em dispositivos de rede para ganho de desempenho, motivando novos estudos.

Como trabalhos futuros, pretende-se analisar o percentual de carga útil, descontando a sobrecarga das transmissões de dados das consultas. Futuramente, pretende-se ainda criar um ambiente virtualizado para simular os diferentes cenários aqui apresentados e analisar as implicações do processamento do *hash join* distribuído no *switch*, simulando a tecnologia SD-WAN. Além disso, planeja-se avaliar cenários mais robustos com maior número de servidores e requisições paralelas, a fim de avaliar a escalabilidade das soluções em rede.

Referências

Appleby, A. (2016). Smhasher. `https://github.com/aappleby/smhasher/`. Acessado em 26/01/2020.

Binnig, C., Crotty, A., Galakatos, A., Kraska, T., and Zamanian, E. (2016). The end of slow networks: It's time for a redesign. *Proc. VLDB Endow.*

Blanas, S., Li, Y., and Patel, J. M. (2011). Design and evaluation of main memory hash join algorithms for multi-core cpus. In *Proc. of the ACM Inter. Conf. on Management of Data (SIGMOD)*.

Council, T. P. P. (2020). Tpc benchmark h. `http://www.tpc.org/tpch/`. Acessado em 25/11/2020.

Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K. E., Santos, E., Subramonian, R., and von Eicken, T. (1993). Logp: Towards a realistic model of parallel computation. *SIGPLAN Not.*, 28(7):1–12.

Estébanez, C., Sáez, Y., Recio, G., and Isasi, P. (2014). Performance of the most common non-cryptographic hash functions. *Software: Practice and Experience*.

Fowler, G., Vo, P., and Noll, L. C. (2013). Fnv hash. `http://www.isthe.com/chongo/tech/comp/fnv/index.html/`. Acessado em 14/04/2020.

Holst, A. (2021). Amount of data created, consumed, and stored 2010-2025. `https://www.statista.com/statistics/871513/worldwide-data-created/`. Acessado em 25/06/2021.

Huang, J., Venkatraman, K., and Abadi, D. J. (2014). Query optimization of distributed pattern matching. In *30th Inter. Conf. on Data Engineering (ICDE)*.

Jin, X., Li, X., Zhang, H., Foster, N., Lee, J., Soulé, R., Kim, C., and Stoica, I. (2018). Netchain: Scale-free sub-rtt coordination. In *Conf. on Networked Systems Design and Implementation (NSDI)*.

Kepe, T. R., de Almeida, E. C., and Alves, M. A. Z. (2019). Database processing-in-memory: An experimental study. *Proc. VLDB Endow.*, 13(3):334–347.

Kossmann, D. (2000). The state of the art in distributed query processing. *ACM Comput. Surv.*

Lerner, A., Hussein, R., Cudre-Mauroux, P., and eXascale Infolab, U. (2019). The case for network accelerated query processing. In *Conf. on Innovative Data Systems Research (CIDR)*.

Lerner, A., Hussein, R., Lee, A. R. S., and Cudré-Mauroux, P. (2020). Networking and storage: The next computing elements in exascale systems? *IEEE Data Eng. Bull.*

Narayana, S., Sivaraman, A., Nathan, V., Goyal, P., Arun, V., Alizadeh, M., Jeyakumar, V., and Kim, C. (2017). Language-directed hardware design for network performance monitoring. In *Proc. of the ACM Special Interest Group on Data Communication (SIGCOMM)*.

Polychroniou, O., Sen, R., and Ross, K. A. (2014). Track join: Distributed joins with minimal network traffic. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, page 1483–1494, New York, NY, USA. Association for Computing Machinery.

Polychroniou, O., Zhang, W., and Ross, K. A. (2018). Distributed joins and data placement for minimal network traffic. *ACM Transactions on Database Systems (TODS)*.

Salama, A., Binnig, C., Kraska, T., Scherp, A., and Ziegler, T. (2017). Rethinking distributed query execution on high-speed networks. *IEEE Data Eng. Bull.*

Scheidt de Cristo, F., Almeida, E., and Alves, M. (2019). Vivid cuckoo hash: Fast cuckoo table building in simd. In *Simp. de Sistemas Computacionais de Alto Desempenho (WSCAD)*.

Sedgewick, R. (1998). *Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching*. Addison-Wesley.

Shin, M., Nam, K., and Kim, H. (2012). Software-defined networking (sdn): A reference architecture and open apis. In *Int. Conf. on ICT Convergence*.

Valduriez, P. and Gardarin, G. (1984). Join and semijoin algorithms for a multiprocessor database machine. *ACM Trans. Database Syst.*

Xiong, P., Hacigumus, H., and Naughton, J. F. (2014). A software-defined networking based approach for performance management of analytical queries on distributed data stores. In *Proc. of Inter. Conf. on Management of Data (SIGMOD)*.

Yang, Z., Cui, Y., Li, B., Liu, Y., and Xu, Y. (2019). Software-defined wide area network (sd-wan): Architecture, advances and opportunities. In *Int. Conf. on Computer Communication and Networks (ICCCN)*.