Evaluation of Hash Join Operations Performance Executing on SDN Switches: A Cost Model Approach

Marisa S. Franco[1], Simone Dominico[1], Tiago R. Kepe[1,2], Luiz C. P. Albini[1], Eduardo C. de Almeida[1], Marco A. Z. Alves[1]

[1] Department of Informatics – Federal University of Paraná (UFPR) – Curitiba, Brazil
[2] Federal Institute of Paraná (IFPR) – Curitiba, Brazil
{masf18,sdominico,trkepe,albini,eduardo,mazalves}@inf.ufpr.br

Abstract. Distributed database systems store and manipulate data on multiple machines. In these systems, the processing cost of query operations is mainly impacted by the data access latency between machines over the network. With recent technology advances in programmable network devices, the network switches provide new opportunities for dynamically managing the network topology, enabling the data processing on these devices with the same network throughput. In this paper, we explore the programmable network switches in query processing, evaluating the processing performance of a cost model in executing the hash join operation. We assume the storage of the hash table built from outer relation and the materialization of the join probing are made in switches using advanced matching techniques similar to package inspections enabled by Ternary Content-Addressable Memories (TCAM) or SRAM via hashing. Our results show that processing the hash join operation using network switches achieved the best results compared to traditional servers, with an average time reduction of 91.82% (Query-10 from TPC-H) and 96.52% (Query-11 from TPC-H).

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Distributed databases/Query processing*; C.2.M [**Computer Systems Organization**]: Computer-Communication Networks—*Miscellaneous*; C.1.3 [**Processor Architecture**]: Other Architecture Styles—*Adaptable architectures*

Keywords: DBMS, Distributed Database, Network Processing, SDN

1. INTRODUCTION

The growth in the amount of data created, captured, copied, and consumed worldwide has been disruptive in the last decade, overtaking the zettabytes boundary in 2020. Global data generation was estimated to reach 79 zettabytes in 2021, growing to impressive 181 zettabytes in 2025 [Holst 2021]. On the other hand, the computational capacity for collecting and storing data has increased significantly over the same period. As a result, the installed base of data storage capacity reached 6,7 zettabytes in 2020 and is expected to grow at a compound annual rate of 19.2% between 2020 and 2025 [Holst 2021]. This vast data generation also increases the databases size, demanding advances in query processing.

Distributed databases are used to scale up query processing of very large databases. However, in distributed databases, the communication cost for data transfer is a significant bottleneck, especially when query processing accesses data stored in different machines, e.g., a hash join operation between two distributed tables.

At the same time, the expansion of Internet services demands flexible technologies, such as the *Software-Defined Wide-Area Network* (SD-WAN), contrasting to the traditional *Wide Area Network*

This work is supported by CNPq, CAPES and Serrapilheira Institute (grant Serra-1709-16621).

(WAN). Furthermore, flexibility is essential for modern services, such as cloud servers that host distributed databases. SD-WAN seeks to reduce WAN complexity through service virtualization, providing greater flexibility in administration and allowing programmable networks via software [Shin et al. 2012]. The programmability feature brings new possibilities to mitigate latencies dynamically. In the literature, researchers, such as Yang et al. [2019], analyze the advances in each network layer provided by SD-WAN. Lerner et al. [2019; 2020] exploit and evaluate the programmability of network elements to speed up database analytical queries. These studies disclose the opportunities that database systems might/shall explore with the evolution of network technologies.

Our research aims to evaluate the processing performance of the distributed join operation in a local/dedicated network with SD-WAN devices. We present a cost model to estimate the execution time of a distributed hash join algorithm. We also present the evaluation of the cost model contemplating different scenarios, including the one executing the operation in a network switch to evaluate the potential of data processing and performance gains provided by SD-WAS devices.

Our results show that processing data using switches presents higher performance when compared to traditional processing in servers with similar data traffic. This network processing achieves an average of more than 90% of runtime reduction in all experiments performed against the best CPU processing scenario – which exploits parallelism – and with an average speedup greater than 12. *Software-Defined Network* (SDN) based systems enable high throughput when processing hash operations as SDN converts traditional processing steps into hardware-based filtering. Nevertheless, our results support the recommendation to focus on overflow-avoiding systems to obtain the maximum performance of such SDN approaches.

The rest of our work is organized as follows: Related work are discussed in section 2; Section 3 presents an overview of distributed databases, hash join operations and SDN processing; Methodology and theoretical modeling are discussed in section 4; Section 5 presents the analysis of the results; Finally, section 6 presents the conclusion and lists possible developments for future work.

2. RELATED WORK

Several approaches have already investigated solutions to accelerate the processing of distributed queries over the network. Narayana et al. [2017] describe the use of a programmable switch as a cache for aggregation operations, detecting network issues (e.g., congestion) and providing a solution for such problems. However, the authors do not address the SD-WAN technology, despite seeking improvements for query operations over the network.

Xiong et al. [2014] discuss using SDN to collect information for query plan optimizations. Other researchers, such as Binnig et al. [2016] and Salama et al. [2017], seek to use advances in network speed to propose new distributed join algorithms, which leverage network speed more efficiently. The authors also consider that the network is no longer a relevant bottleneck while executing distributed queries. In contrast, we seek to use the programmability available in SD-WAN to analyze the possible advantages of processing join operation in a switch. In addition, we corroborate Binnig et al. [2016] and Salama et al. [2017], since the communication time seemed to have a low influence on the final result of our test cases.

Jin et al. [2018] present a mechanism that coordinates data storage in programmable switches. The main objective is to execute every query plan within these network devices. In Lerner et al. [2019], the authors implement NetAccel, that offloads entire queries to switches seeking to use the programmability available in the switch to improve the execution of analytical queries. NetAccel sends some data to the switch and uses packet-processing hardware called Match-Action Unit (MAU) to match packet fields with a row in this table by exact matching. NetAccel uses resources for SRAM storage and matching via hashing. Furthermore, the authors present a new set of operations to speed up processing using switches. On the other hand, we introduce in this work an evaluation with different components that aims to schedule query operations on the switch and observe the communications in the network.

Lerner et al. [2020] discuss the challenges and opportunities of accelerating queries using network devices. The authors use the switch as an active element that stores an entire aggregation table enabling fast data filtering. They argue that improving the algorithms allows taking advantage of new network technologies and reduces data movement across the network. This related work is one of the primary bases for our research. Based on their findings, we propose a cost model approach to evaluate hypothetical scenarios to identify bottlenecks to network-based query processing.

This paper is an extension of our previous work presented at SBBD 2021 [Franco et al. 2021]. It includes new experimental scenarios, with network switch processing emulating advanced matching techniques, such as Ternary Content Addressable Memory (TCAM) with different hypothetical overflow scenarios. The main difference of such processing is that, in the best scenario, data can be filtered with the same throughput from the network. Our evaluations did not consider the initial cost of reconfiguring the switch table. This cost depends on the TCAM update time. Nevertheless, our approach is orthogonal to previous work, such as Qiu et al. [2019] and Wan et al. [2022], that propose ways to reduce such overhead.

3. BACKGROUND ON DBMS AND SDN PROCESSING

This section provides background to the distributed database systems and query processing cost. Moreover, we describe in detail the join operation. We also discuss the technique to perform query processing in network devices using a SDN processing.

3.1 Distributed DBMS

A distributed database management system maintains a collection of databases spread across several servers connected through a network. Each server on the network, also called a node, is responsible for storing and processing data. The general architecture for this type of system is the client-server based model, in which the client node sends a query to a controller node that forward commands to the required server nodes. Distributed query processing is usually collaborative, depending on the data distribution scheme, where nodes need to exchange data for computing the final query result. Data transfer is usually performed through a dedicated network to support a high flow of exchanged data among network devices, such as network switches, which act as passive agents, performing only operations management and delivering packets through the network.

However, the cost of query processing in distributed databases is bound to the cost of transferring data over the network [Kossmann 2000]. Thus, different query processing algorithms focus on generating query plans that minimize the amount of data transferred. Queries that perform the join operation use semi-joins and bloom filters to minimize data transfer across the network [Polychroniou et al. 2014].

3.2 Join Operation

A join operation in a query retrieves data from multiple relational database tables through logical combinations among them. In a distributed environment, there may be several join [Valduriez and Gardarin 1984; Huang et al. 2014; Polychroniou et al. 2018] approaches: **(i)** Broadcast joins, where a table is replicated and sent to all processing nodes, the canonical use case is a large table with a small reference table; **(ii)** Hash joins in which, at the time of data storage, partitioning between servers is done via a hash algorithm. The join between major and minor relationship data can be done locally, without any data transfer between nodes, because all relevant data is already co-located;

(iii) A complete hash join operation, where data from servers is used to create hash tables, and these are transferred to other nodes for completion of the processing on servers.

Cardinality has a considerable impact on join operations that optimizers consider when choosing the join algorithm (hash, merge, NL, among others). Therefore, we focused on hash join operations to understand the impact of high cardinality tables that fit in memory. Furthermore, previous studies [Kepe et al. 2019] showed that the join operator is one of the four operators contributing to the execution time and memory usage of a columnar database system with the TPC-H benchmark [Council 2020]. Another important fact is the widespread use of hash joins in distributed query processing due to data partitioning and transmission approaches to relations [Valduriez and Gardarin 1984; Huang et al. 2014; Polychroniou et al. 2018].

Hash join is one of the most used algorithms in relational database management systems, which use hash partitioning. As illustrated in Code 1, the hash join algorithm has two phases [Blanas et al. 2011]: the build and the probe phase. The build phase traverses the smaller relation (the outer relation R) and stores the keys of interest into a hash table (HT), applying a hash function $h1$. The probe phase traverses the larger relation (the inner relation S), applying the same hash function $h1$, looking for the matched values mapped in the hash table to generate the resulting relation.

Code 1. Pseudocode of basic Hash Join Algorithm.

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

3.3 SDN processing

The evolution of network devices driven by SD-WAN technologies presents a potential for network-accelerated query processing. For example, an SDN switch can speed up query operations, such as join, where the data used by a query may be stored and processed on the network. Thus, moving data across network devices contributes to query processing [Lerner et al. 2019].

Programmable network devices, such as the switch, have packet processing hardware formed by Match-Action Units or MAUs. MAUs match data quickly using advanced matching techniques. These techniques are similar to package inspections enabled by TCAM or SRAM via hashing.

For network query processing, switches appear not only as packet routing devices. Such devices start to store data and perform the processing of parts of the query plan. For instance, considering the join operation discussed in this paper, the hash table built with the external relation can be stored directly in the switch. The matching happens as data passes through the network. In this scenario, if the user requested only to send the query result, data can be filtered with the same throughput from the network.

However, the capacity of the network device will influence the query processing. For example, the tuples that do not fit on the switch tables cause tuple overflow [Lerner et al. 2020]. Therefore, it is necessary to consider different strategies to deal with overflow. A technique used in Lerner et al. [2019] controls the number of tuples stored in the switch and redirects the remaining tuples for processing at another matching unit. We consider in this paper that the switch processor processes such overflows.

4. METHODOLOGY AND THEORETICAL MODEL

In this section, we describe the methodology used in our analysis. We present the network topology, the evaluated queries, and the hash join processing scenarios. We also introduce details related to the proposed theoretical cost model.

4.1 Methodology

For the evaluation of the distributed hash join processing, we adopted some data transmission scenarios based on a star topology: two database servers and a client connected by a switch, as shown in Figure 1.

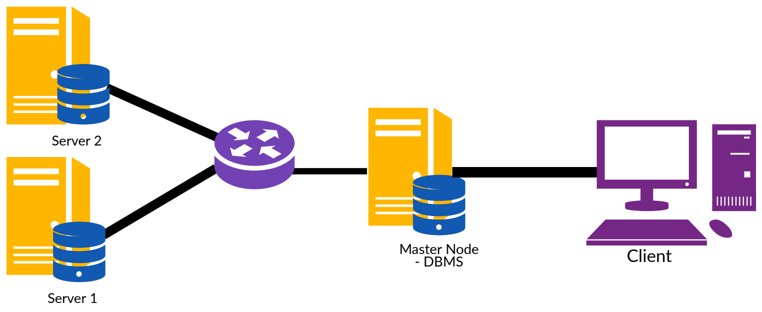


Fig. 1. Star topology adopted for the definition of distributed hash join processing scenarios.

For a straightforward analysis, we simplified the queries. We used versions from query-10 and query-11 from TPC-H [Council 2020], which include only the hash join operation, according to Codes 2 and 3 described in SQL. The TPC-H is a decision support benchmark that encompasses a set of business-oriented ad-hoc queries and concurrent data modifications. Such queries represent decision support systems that examine large volumes of data and execute complex operations.

Code 2. Query-10 from TPC-H.

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

Code 3. Query-11 from TPC-H.

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

We implemented the queries in the C language with a linear algorithm and stored the data with the Decomposition Storage Model (DSM) format. We chose to use the function MurmurHash3 [Appleby 2016] to create the hash tables because it reached excellent performance in terms of data spread and runtime, according to previous work in the literature [Estébanez et al. 2014; Scheidt de Cristo et al. 2019]. We also considered six hash join distributed processing scenarios detailed in Table I, which shows the processing steps divided into initial data storage (S), communication (C), and processing (P) for each proposed scenario.

In our model, we defined M as the size of the hash table (i.e., the total number of entries), and N as the number of keys inserted into the table. The load factor is $\alpha = N/M$. The hashing function might produce collisions when two keys have the same hash, which leads to different values mapped to the same entry in the hash table. In the experiments, the choice criterion for the size of the hash tables (M) was a prime number immediately below a power of two, which would provide a load factor between 5 and 10, to favor a better spread of the data, avoiding collisions [Sedgewick 1998].

We evaluated each scenario with three workloads (TPC-H scales 1 GB, 10 GB, and 100 GB) and four network technologies (Ethernet 100 Gb, Ethernet 200 Gb, Ethernet 400 Gb, and InfiniBand HDR $12\times$), with full-duplex communication. The criterion for choosing network technologies was temporal: all analyzed standards came into use in the last decade (2014, 2017, 2017, and 2021, respectively). In our previous work, we had only considered switch processing – scenarios 3 and 5 – with 100% overflow. Thus, the processing time of the analysis phase from the hash join operation was estimated as if the processor embedded inside the switch entirely executed it.

Table I. Analyzed distributed hash join processing scenarios.

Scenario	(S) Initial Storage, (C) Communication and (P) Processing
1	S: Server 1 stores the larger table. Server 2 stores the smaller table. C: Larger table is transmitted. P: Processing is done on server 2.
2	S: Server 1 stores the larger table. Server 2 stores the smaller table. C: Smaller table is transmitted. P: Processing is done on server 1.
3	S: Server 1 stores the larger table. Server 2 stores the smaller table. C: Both relations are transmitted to the switch. P: **Processing on the network switch.**
4	S: Each server has one half of the two tables. P: Each server handles the build phase using the smaller table. C: Sharing of hash tables between servers. P: Each server makes part of the analysis, using the two hashes and its half of the larger table.
5	S: Each server has one half of the two tables. C: Both relations are transmitted to the switch. P: **Processing on the network switch.**
6	S: Each server has one half of the two tables and co-located relevant data. P: Local processing on the servers of each of the halves.

This work includes new experimental scenarios, with network switch processing emulating advanced matching techniques, such as Ternary Content Addressable Memory (TCAM), with different hypothetical **overflow scenarios: 0%, 25%, 50%, and 75%**. The main difference between such scenarios is that in the **best case** (without overflow, i.e., 0% overflow), data can be filtered with the same throughput from the network. On the other hand, in the **worst case** (with 100% overflow), the switch processor filters all data. For the **other cases** (with overflow between 0% and 100%), we consider the extra processing time according to the overflow percentage in our cost model.

We applied actual run data to compute our estimates. We used a machine with two sockets in the experiments, each with an Intel Xeon Silver 4114 (with Skylake microarchitecture). The Xeon socket has ten cores with a private L1 (I+D) cache (32 KB), a private L2 cache (1 MB), and a shared L3 cache (14 MB). This machine used has 128 GB of DDR-4 main memory and 14 TB of disk storage, running the Ubuntu operating system, version 18.04.01 LTS.

We gathered the elapsed time of 20× execution for each of the 12 base experiments, totaling 240 runs. The experiments combined the different factors at the following levels: (i) query-10 and query-11; (ii) workload size (1 GB, 10 GB, and 100 GB of data); (iii) the standard algorithm and the algorithm simulating network parallelism. Then, we applied the coefficient of variation (CV) to express the variability of the data taking the influence of the variables' magnitude order. CV is the division of the standard deviation by the data mean, resulting in a pure number between 0 and 1. The smaller the CV, the more homogeneous the dataset. Usually, datasets with CV between 0 and 0,30 present low variability.

We computed the CVs from the execution times in the build and probe phases for each base experiment, using the averages in our cost model. We realized the absence of CV above 0.30 in the run batteries of each base experiment for these parameters. For the build time, the minimum value of CV was 0.001, and the maximum value was 0.206. In the probe phase, the lowest CV was 0.003, and the highest was 0.075. Thus, the samples from the results using those parameters in our analysis proved to be highly homogeneous in the battery of tests. Therefore, we understand that the 20 executions of each base experiment are justified.

4.2 Theoretical Model

Our proposed cost model is inspired by the LogP theoretical machine model from [Culler et al. 1993] (a traditional model in parallel programming). In this paper, we considered that the external relation would be sent for the network switch to build the hash table, and the inner relation filtering happens at the same network transfer rate. Besides, in the cost model, we varied tuples overflow in percentage. It is important to note that there is an initial cost of configuring the switch tables to utilize the potential of TCAMs or SRAM to speed up query processing on network devices. However, this cost was not considered in our research experiments as it heavily depends on the TCAM technology adopted.

Table II describes, mathematically, what were the calculations made to estimate the total execution times in the six scenarios of the distributed hash join processing. For example, to simulate scenarios 4 and 6 that use parallel processing distributed on two servers, the algorithms store two relations with half the size of the regular relations, generating output relations with half the size of the regular ones used initially.

Table II. Cost calculation used in each scenario.

Scenario	**Total runtime**
1	$(DAB + DPB + DAP + DPP) + ((N + O)/BW)$
2	$(DAB + DPB + DAP + DPP) + ((n + O)/BW)$
3	$(DAB + DPB + DAP + (OvFP * DPP)) + (((max(N; n)) + O)/BW)$
4	$(DAB + DPB + DAP + DPP) + (((H/2) + O)/BW)$
5	$(DAB + DPB + DAP + (OvFP * DPP)) + ((N/2 + n/2 + O)/BW)$
6	$(DAB + DPB + DAP + DPP) + (O/BW)$

DAB: data access build phase. **DPB:** data processing build phase.
DAP: data access probe phase. **DPP:** data processing probe phase.
About DPB and DPP:
- scenarios 1, 2, 3 and 5, sequential processing in build and probe phases;
- scenarios 4 e 6, parallel build phase and parallel probe phase.

OvFP: overflow percentage, used in scenarios 3 and 5.
N: larger table size. **n:** smaller table size.
H: hash table size - perfect distribution. **O:** output ratio size. **BW:** bandwidth.

Note: "size" refers to the distinct number of tuples (cardinality) in the table multiplied by the sum of the sizes of the columns relevant to the query.

There are two independent loops in the build phase of these algorithms that emulate network/server parallelism. First, these loops traverse one-half of the smaller relation for building the hash tables **A** and **B**. In the probe phase, there are also two independent loops to traverse one-half of the larger relation, looking for matches in both hash tables (**A** and **B**) and generating the output. We collected the execution time averages from the maximum value between the two build phases in each battery of tests for these algorithms. We used this exact computation in the probe phase.

In scenarios 3 and 5, we build the hash table with the external relation, storing it in the network device tables (for instance, using TCAMs). Nevertheless, the hash table storage on the switch on the router depends on the available space. If the number of tuples exceeds the storage capacity of the switch, there may be an overflow [Lerner et al. 2019]. Thus, we consider in our calculations some overflow sub-scenarios for scenarios 3 and 5: 0%, 25%, 50%, 75%, and 100%.

In the scenario with no overflow (0%), the analysis phase only considers the data access time, completely disregarding the hash table processing time. In this case, the tuples from the larger table would reach the network device and be filtered by the hash table stored in the TCAM with the same network throughput. The process works analogously to routing, with the hash table as the addressing table and the tuples being treated as packets. To simulate the different overflow percentages, we consider the corresponding percentage value of the probe phase processing time plus the data access time to calculate the probe phase runtime. With 100% overflow, we consider that all the processing

(100%) of the probe phase must be done in the switch and added to the data access time to obtain the runtime of the probe phase.

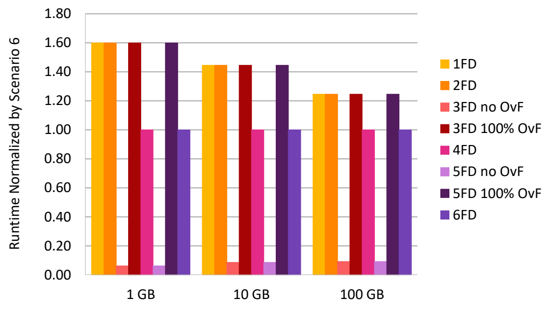
Finally, we use some simplifications to define our cost model. For example, we consider only each technology's theoretical maximum bandwidth capacities for data transmission calculations (100 Gb/s, 200 Gb/s, 400 Gb/s, and 600 Gb/s, respectively, in half-duplex). Our model analyzes only the raw data without considering headers and dividing the messages into packets (which would result in an average variation of less than 2% of the total volume of sent data). Also, we do not take the network latencies and distances among nodes in the calculations. In our estimates, the initial latency of communication is negligible since we send large volumes of data in a few packages. We considered the perfect scattering of data in hash tables to measure the volume of data transferred in scenario 4.

5. RESULTS ANALYSIS

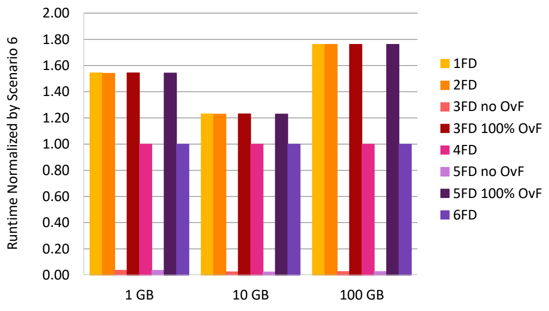
In this section, we present the results of the query-10 and query-11 queries in the scenarios described in Table I. We assess the runtime according to the theoretical modeling presented and analyze the results for each query. We present three distinct assessments to distinguish the relevancy of the workload size (Subsection 5.1), the switch overflow variation (Subsection 5.2), and network capacity and technology (Subsection 5.3). Based on the results, we unravel the best-cases and worst-cases scenarios in Subsection 5.4.

5.1 Varying the Workload Size

In this subsection, we analyze the variability of the workload size starting from 1 GB, 10 GB up to 100 GB. This preliminary evaluation is required as the workload size directly pressures the network capacity and might cause switch overflows. Figures 2(a) and 2(b) show the "best-case scenario": InfiniBand HDR 12×, with runtime normalized by scenario 6.



(a) *Query-10*



(b) *Query-11*

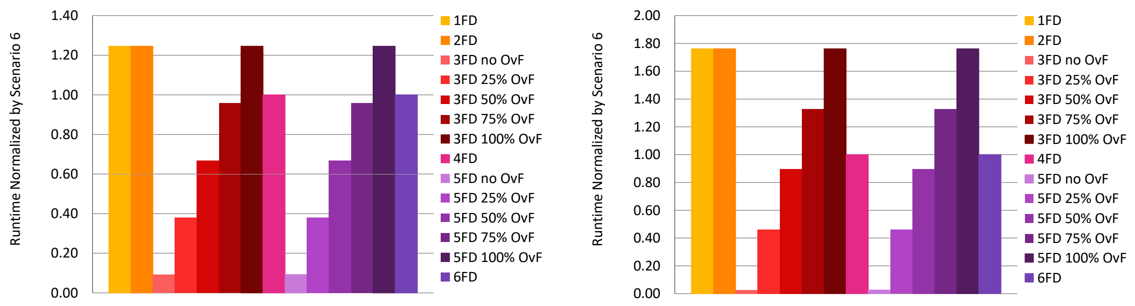
Fig. 2. Runtime values according to workload variation, using InfiniBand HDR 12× with MurmurHash3 function. Values normalized by scenario 6. **Numbers from 1 to 6** represent the scenarios described in Table I. **FD:** means full-duplex. **OvF:** means overflow.

In all experiments from our previous work [Franco et al. 2021], scenario 6 presented the best performance in terms of queries runtime, including the different workload sizes. Thus, we keep scenario 6 as a reference for all other scenarios in this paper. Scenario 6 exploits parallelism with co-located data and is the best CPU processing scenario of query-10 and query-11. Nevertheless, the new results show an impressive performance gain in processing on network devices emulating the use of TCAM without overflow. In this case, the performance improvements depend on the network speed. For instance, when executing query-10 with 100 Gb and InfiniBand HDR 12×, scenarios 3 and 5 without overflow reduced the runtime by 90.79% compared to scenario 6, achieving a speedup of 10.86. The gains were even better in query-11 with the same parameters: achieving an execution time reduction of 97.33% and speedup of 37.46.

For CPU processing, we notice the performance improvement reached while executing the queries in parallel on two servers and the significant impact of data transfer on the total cost of distributed hash join processing. Scenario 6 is the one that transfers the least amount of data: just the query output relation data. At the same time, it has the most storing overhead and synchronizing data on servers, as the data are replicated in the servers. In both queries, scenario 4 reached the second-best performance for CPU processing. This scenario also performs parallel processing on the servers and carries the second smallest amount of data: the hash table and output relation. In query-11, the third-best CPU processing performance was obtained by scenario 2. While scenarios 1 and 2 are similar in query-10, as more data (columns) are requested from the smaller relation – whose data are transferred in scenario 2. Scenarios 3 and 5 with 100% overflow perform similarly to scenarios 1 and 2 in both queries.

5.2 Switch Overflow Evaluation

In this subsection, we assess the switch capacity by varying the overflow percentage from 0% up to 100%. We vary the overflow for scenarios 3 and 5 that use the switch to process the tables, and we compare such variations against the other scenarios. Figures 3(a) and 3(b) show the performance of different overflow percentages for scenarios 3 and 5 compared to the other scenarios.



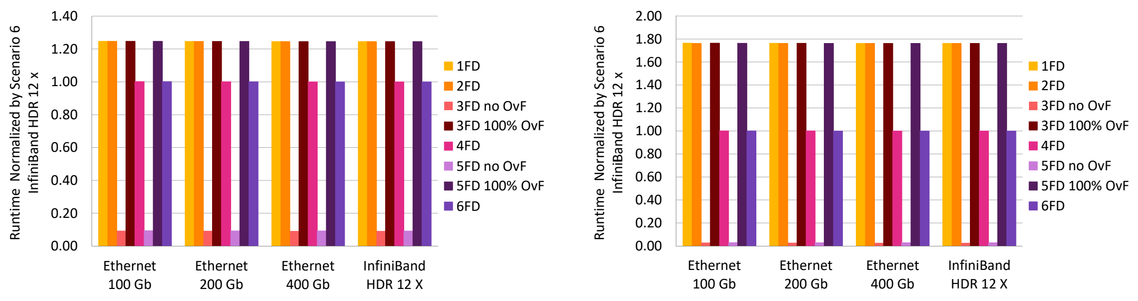
(a) *Query-10* (b) *Query-11*

Fig. 3. Runtime values of different overflow percentages for scenarios 3 and 5 compared to the other scenarios, using InfiniBand HDR with 100 GB workload and MurmurHash3 function. Values normalized by the respective scenario 6.**Numbers from 1 to 6** represent the scenarios described in Table I. **FD:** means full-duplex. **OvF:** means overflow.

In query-10, scenarios 3 and 5 perform better than scenarios 4 and 6, even with 75% overflow. However, in query-11, scenarios 3 and 5 only perform better than scenarios 4 and 6 up to a maximum of 50% overflow. These results reflect the weight of the build and probe phases – and, consequently, the weight of the cardinality of the tables – in the total runtime. In both queries on all analyzed scenarios, we can see the strong influence of processing time on the final runtime. We observed that the query execution times in scenarios 3 and 5 with 100% overflow are similar to scenarios 1 and 2. That happens because the amount of data transferred is the same, and there is no parallelism.

5.3 Varying Network Capacity and Technology

The network capacity and technology are crucial factors for distributed data processing. Therefore, Figures 4(a) and 4(b) show the impact of varying these network aspects. The performance of different network technologies is very close because our cost model does not consider the network delays and the costs of packaging/unpacking messages. Also, the build and probe phases are the most relevant elements in the runtime composition.



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

Fig. 4. Runtime values according to the variation in the flow capacity of the network, using InfiniBand HDR with 100 GB workload and MurmurHash3 function. Values normalized by the respective InfiniBand HDR scenario 6. **Numbers from 1 to 6 represent the scenarios described in Table I. FD: means full-duplex. OvF: means overflow.**

5.4 Comparisons Between Best-cases and Worst-cases Scenarios

Based on the assessments of Subsections 5.1, 5.2, and 5.3, we chose the best-cases and worst-cases scenarios to compare the performances whose analyzes we present in this subsection. Table III brings comparison of the cost model dimensions with the total query runtime, in the worst-case scenario (1) and the best-case scenario (6) for CPU Processing, using Ethernet 100 Gb. We used the slower network technology to make that comparison as the network speed increases, and the data transfer becomes less significant.

Table III. Percentage of each step's time for query-10 and query-11. Considering the worst case (1) and best case (6), with Ethernet 100 Gb.

Scenario	**Query**	**Workload (GB)**	**Build phase (ms)**		**Probe phase (ms)**		**Data transfer (ms)**
			Data access	**Data processing**	**Data access**	**Data processing**	
1	10	1	0.00675	24.55	0.06455	632.01	5.28
			0.001020%	3.71%	0.01%	95.48%	0.80%
6	10	1	0.00675	10.64	0.06455	399.58	4.80
			0.001626%	2.56%	0.015551%	96.26%	1.16%
1	10	10	0.00760	1310.16	0.56465	20905.56	52.80
			0.000034%	5.88%	0.002536%	93.88%	0.237100%
6	10	10	0.00760	307.44	0.56465	15065.42	48.00
			0.000049%	1.99%	0.003661%	97.69%	0.31%
1	10	100	0.00740	29010.80	5.16	364648.54	528.00
			0.000002%	7.36%	0.001310%	92.51%	0.133945%
6	10	100	0.00740	14986.89	5.16	301092.29	480.00
			0.000002%	4.73%	0.001631%	95.11%	0.15%
1	11	1	0.00715	0.73760	0.00450	35.71	0.76800
			0.019204%	1.98%	0.012086%	95.92%	2.06%
6	11	1	0.00715	0.36255	0.00450	23.28	0.25600
			0.03%	1.52%	0.02%	97.36%	1.07%
1	11	10	0.00670	10.13	0.04935	561.19	7.68
			0.001157%	1.75%	0.008522%	96.91%	1.33%
6	11	10	0.0067	4.51635	0.04935	460.24175	2.56
			0.00%	0.97%	0.01%	98.47%	0.55%
1	11	100	0.00720	839.03	0.36650	55381.58	76.80
			0.000013%	1.49%	0.000651%	98.37%	0.136417%
6	11	100	0.00720	138.76	0.36650	31776.91	25.60
			0.000023%	0.43%	0.001147%	99.48%	0.08%

The experiments reveal that the weight of the data transfer diminishes as the workload size increases, considering the same query and scenario. For example, scenario 6 has a distinct behavior because it moves fewer data around the network. When analyzing each stretch of the query execution, the high query performance is due to the local parallel processing of both phases (build and probe) on the two servers.

Table IV brings comparison of the cost model dimensions with the total query runtime in the worst-case scenario (3 100% OvF) and the best-case scenario (3 no OvF) for CPU Processing, using Ethernet 100 Gb. One more time, we used the slower network technology to make that comparison, as the network speed increases as the data transfer become less significant.

Table IV. Percentage of each step's time for query-10 and query-11. Considering the worst case in switch (3 without overflow - 3 no OvF) and best case in switch (3 100% of overflow - 3 100% OvF), with Ethernet 100 Gb.

Scenario	**Query**	**Workload (GB)**	**Build phase (ms)**		**Probe phase (ms)**		**Data transfer (ms)**
			Data access	**Data processing**	**Data access**	**Data processing**	
3 no OvF	10	1	0.00675	24.55	0.06455	0	5.28
			0.022576%	82.10%	0.22%	-	17.66%
3 100% OvF	10	1	0.00675	24.55	0.06455	632.01	5.28
			0.001020%	3.71%	0.01%	95.48%	0.80%
3 no OvF	10	10	0.00760	1310.16	0.56465	0	52.80
			0.000557%	96.09%	0.041411%	-	3.872306%
3 100% OvF	10	10	0.00760	1310.16	0.56465	20905.56	52.80
			0.000034%	5.88%	0.002536%	93.88%	0.237100%
3 no OvF	10	100	0.00740	29010.80	5.16	0	528.00
			0.000025%	98.20%	0.017479%	-	1.787167%
3 100% OvF	10	100	0.00740	29010.80	5.16	364648.54	528.00
			0.000002%	7.36%	0.001310%	92.51%	0.133945%
3 no OvF	11	1	0.00715	0.73760	0.00450	0	0.76800
			0.471247%	48.61%	0.296589%	-	50.62%
3 100% OvF	11	1	0.00715	0.7376	0.00450	35.71	0.76800
			0.019204%	1.98%	0.012086%	95.92%	2.06%
3 no OvF	11	10	0.00670	10.13	0.04935	0	7.68
			0.037499%	56.70%	0.276208%	-	42.98%
3 100% OvF	11	10	0.0067	10.1309	0.04935	561.1933	7.68
			0.001157%	1.75%	0.008522%	96.91%	1.33%
3 no OvF	11	100	0.00720	839.03	0.36650	0	76.80
			0.000786%	91.58%	0.040002%	-	8.382436%
3 100% OvF	11	100	0.00720	839.03	0.36650	55381.58	76.80
			0.000013%	1.49%	0.000651%	98.37%	0.136417%

Table V brings a summary of performance gains (time reduction) and speedup of scenario 3 without overflow versus scenario 6. We observe an average time reduction of 91.82% (achieving a speedup of 12.38), and a maximum time reduction of 93.80% (achieving a speedup of 16.12).

Table V. Summary of performance gains (time reduction) and speedup of scenario 3 without overflow versus scenario 6 – Query-10.

Query-10					
Workload (GB)	**Network technology**	**Total runtime (ms)**		**Time reduction**	**S(p)**
		3 no OvF	**6**		
1	Ethernet 100 Gb	29.90	415.09	92.80%	13.88
10	Ethernet 100 Gb	1363.53	15421.43	91.16%	11.31
100	Ethernet 100 Gb	29543.97	316564.35	90.67%	10.72
1	Ethernet 200 Gb	27.26	412.69	93.39%	15.14
10	Ethernet 200 Gb	1337.13	15397.43	91.32%	11.52
100	Ethernet 200 Gb	29279.97	316324.35	90.74%	10.80
1	Ethernet 400 Gb	25.94	411.49	93.70%	15.86
10	Ethernet 400 Gb	1323.93	15385.43	91.39%	11.62
100	Ethernet 400 Gb	29147.97	316204.35	90.78%	10.85
1	InfiniBand HDR 12 X	25.50	411.09	93.80%	16.12
10	InfiniBand HDR 12 X	1319.53	15381.43	91.42%	11.66
100	InfiniBand HDR 12 X	29103.97	316164.35	90.79%	10.86
		Geometric mean		91.82%	12.38

The performance gains obtained by scenario 3 without overflow, in relation to scenario 6, in query-11 can be seen in Table VI and were even more expressive. Scenario 3 without overflow has an average time reduction of 96.52% (achieving a speedup of 30.03), with maximum reduction of 97.54% (achieving a speedup of 40.57).

Table VI. Summary of performance gains (time reduction) and speedup of scenario 3 without overflow versus scenario 6 – Query-11.

Query-11					
Workload (GB)	**Network technology**	**Total runtime (ms)**		**Time reduction**	**S(p)**
		3 no OvF	**6**		
1	Ethernet 100 Gb	1.52	23.91	93.65%	15.76
10	Ethernet 100 Gb	17.87	467.37	96.18%	26.16
100	Ethernet 100 Gb	916.20	31941.64	97.13%	34.86
1	Ethernet 200 Gb	1.13	23.78	95.23%	20.98
10	Ethernet 200 Gb	14.03	466.09	96.99%	33.23
100	Ethernet 200 Gb	877.80	31928.84	97.25%	36.37
1	Ethernet 400 Gb	0.94	23.71	96.03%	25.19
10	Ethernet 400 Gb	12.11	465.45	97.40%	38.45
100	Ethernet 400 Gb	858.60	31922.44	97.31%	37.18
1	InfiniBand HDR 12 X	0.88	23.69	96.30%	27.01
10	InfiniBand HDR 12 X	11.47	465.24	97.54%	40.57
100	InfiniBand HDR 12 X	852.20	31920.31	97.33%	37.46
		Geometric mean		96.52%	30.03

6. CONCLUSIONS AND FUTURE WORK

This paper presents an analysis through a cost model of a distributed hash join in six distinct scenarios, including the processing in a network switch emulating advanced matching techniques, such as TCAM with different hypothetical overflow scenarios. Our choice for the star topology (with two servers, a switch, and a client) proved successful because basic variations in data distribution and processing were able to differentiate performance and indicate the best scenarios to be explored to advance the research.

In all experiments from our previous work [Franco et al. 2021], scenario 6 achieved the best performance in both evaluated queries. Moreover, it exploits parallelism with co-located data and is still the best CPU processing scenario for query-10 and query-11. In that scenario, each server has one-half of the two relationships with a strategic data placement, performing local data processing on both servers, diminishing data transfer, and enabling parallel execution. It is worth mentioning that partitioning the tables on the servers to maintain hot data placement comes with a cost. However, we did not investigate this factor in our analysis. In addition, parallel execution may impair the system performance when there is a high demand for distinct queries.

However, our new analyzes have shown us the great potential of switch processing using advanced matching techniques, such as TCAMs. Furthermore, the new results have shown an impressive performance gain in processing on network devices emulating TCAM without overflow (scenarios 3 and 5, in which processing is done on the switch). As we said previously, the performance improvements depends on the network speed in this case.

Our results show that the data processing using SDN switches presents higher performance than traditional processing in servers. Our comparisons with similar data traffic achieved more than 90% of runtime reduction on average in all experiments – and with an average speedup greater than 12.

Given that, historically, the transmission speeds of network technologies do not grow at the same rate as the volume of data transferred annually, data transfer latency remains an essential bottleneck in distributed systems. Moreover, the present research demonstrates that parallelism in processing is a highly relevant point of query performance that involves distributed processing of the join operation in databases.

In the study scenarios, we consider a private and local network to mitigate the risks and disadvantages of using programmable networks for the distributed processing of hash join operations. However, we still need to analyze fault tolerance and memory limitations issues. Besides, in this work, we could notice that exclusive processing in network devices might lead to centralization, creating a bottleneck in processing. On the other hand, recurring query processing can benefit from programmable networks.

Considering the results observed emulating advanced matching techniques in network devices, it is evident the importance of avoiding overflow for a more significant gain in performance. We understand that our results will contribute more profound knowledge about different strategies of distributed data processing and the potential of data processing in network devices to gain performance, leading to new and further studies.

In this work, we did not consider the initial cost of reconfiguring the switch tables to utilize the potential of TCAMs or SRAM to speed up query processing on network devices. However, this poses a limitation in our model that could be explored in future work comparing multiple TCAM implementations.

REFERENCES

Appleby, A. Smhasher. `https://github.com/aappleby/smhasher/`, 2016. Accessed: 2020-01-26.

Binnig, C., Crotty, A., Galakatos, A., Kraska, T., and Zamanian, E. The end of slow networks: It's time for a redesign. *Proceedings of the VLDB Endowment* 9 (7): 528–539, 2016.

Blanas, S., Li, Y., and Patel, J. M. Design and evaluation of main memory hash join algorithms for multi-core cpus. In *Proceedings of the ACM SIGMOD International Conference on Management of Data Conference*. Athens, Greece, pp. 37–48, 2011.

Council, T. P. P. Tpc benchmark h. `http://www.tpc.org/tpch/`, 2020. Accessed: 2020-11-15.

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

Estébanez, C., Sáez, Y., Recio, G., and Isasi, P. Performance of the most common non-cryptographic hash functions. *Software: Practice and Experience* vol. 44, pp. 681–698, 2014.

Franco, M. S., Dominico, S., Kepe, T., Albini, L. C., de Almeida, E. C., and Alves, M. Z. Processamento distribuído de operações hash join em switches programáveis: Uma análise via modelo de custo. In *Proceedings of the Brazilian Symposium on Databases*. Porto Alegre, RS, Brazil, pp. 253–264, 2021.

Holst, A. V. S. Amount of data created, consumed, and stored 2010-2025. `https://www.statista.com/statistics/871513/worldwide-data-created/`, 2021. Accessed: 2021-06-25.

Huang, J., Venkatraman, K., and Abadi, D. J. Query optimization of distributed pattern matching. In *Proceedings of the IEEE International Conference on Data Engineering*. Chicago, IL, USA, pp. 64–75, 2014.

Jin, X., Li, X., Zhang, H., Foster, N., Lee, J., Soulé, R., Kim, C., and Stoica, I. Netchain: Scale-free sub-rtt coordination. In *Conference on Networked Systems Design and Implementation*. Renton, WA, USA, pp. 35–49, 2018.

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

Kossmann, D. The state of the art in distributed query processing. *ACM Computing Surveys* 32 (4): 422–469, 2000.

Lerner, A., Hussein, R., Cudre-Mauroux, P., and eXascale Infolab, U. The case for network accelerated query processing. In *Conference on Innovative Data Systems Research*. Asilomar, CA, USA, 2019.

Lerner, A., Hussein, R., Lee, A. R. S., and Cudré-Mauroux, P. Networking and storage: The next computing elements in exascale systems? *IEEE Data Engineering Bulletin* vol. 43, pp. 60–71, 2020.

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

Polychroniou, O., Sen, R., and Ross, K. A. Track join: Distributed joins with minimal network traffic. In *Proceedings of the ACM SIGMOD International Conference on Management of Data Conference*. New York, NY, USA, pp. 1483–1494, 2014.

Polychroniou, O., Zhang, W., and Ross, K. A. Distributed joins and data placement for minimal network traffic. *ACM Transactions on Database Systems* 43 (3): 1–45, 2018.

Qiu, K., Yuan, J., Zhao, J., Wang, X., Secci, S., and Fu, X. Fastrule: Efficient flow entry updates for tcam-based openflow switches. *IEEE Journal on Selected Areas in Communications* 37 (3): 484–498, 2019.

Salama, A., Binnig, C., Kraska, T., Scherp, A., and Ziegler, T. Rethinking distributed query execution on high-speed networks. *IEEE Data Engineering Bulletin* 40 (1): 27–37, 2017.

Scheidt de Cristo, F., Almeida, E., and Alves, M. Vivid cuckoo hash: Fast cuckoo table building in simd. In *High Performance Computing Systems Symposium*. Porto Alegre, RS, Brazil, 2019.

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

Shin, M., Nam, K., and Kim, H. Software-defined networking (sdn): A reference architecture and open apis. In *International Conference on ICT Convergence*. Jeju, Korea (South), pp. 360–361, 2012.

Valduriez, P. and Gardarin, G. Join and semijoin algorithms for a multiprocessor database machine. *ACM Transactions on Database Systems* 9 (1): 133–161, 1984.

Wan, Y., Song, H., and Liu, B. Greedyjump: A fast tcam update algorithm. *IEEE Networking Letters* 4 (1): 25–29, 2022.

Xiong, P., Hacigumus, H., and Naughton, J. F. A software-defined networking based approach for performance management of analytical queries on distributed data stores. In *Proceedings of the ACM SIGMOD International Conference on Management of Data Conference*. Snowbird, Utah, USA, pp. 955–966, 2014.

Yang, Z., Cui, Y., Li, B., Liu, Y., and Xu, Y. Software-defined wide area network (sd-wan): Architecture, advances and opportunities. In *International . Conference on Computer Communication and Networks*. Valencia, Spain, pp. 1–9, 2019.