Opportunistic Query Execution on SmartNICs for Analyzing In-Transit Data

Jianshen Liu
Computer Science & Engineering
UC Santa Cruz
Santa Cruz, CA, USA
jliu120@ucsc.edu

Carlos Maltzahn
Computer Science & Engineering
UC Santa Cruz
Santa Cruz, CA, USA
carlosm@ucsc.edu

Craig Ulmer
Scalable Modeling & Analysis
Sandia National Laboratories
Livermore, CA, USA
cdulmer@sandia.gov

***Abstract*—High-performance computing (HPC) systems researchers have proposed using current, programmable network interface cards (or SmartNICs) to offload data management services that would otherwise consume host processor cycles in a platform. While this work has successfully mapped data pipelines to a collection of SmartNICs, users require a flexible means of inspecting in-transit data to assess the live state of the system. In this paper, we explore SmartNIC-driven opportunistic query execution, i.e., enabling the SmartNIC to make a decision about whether to execute a query operation locally (i.e., "offload") or defer execution to the client (i.e., "push-back"). Characterizations of different parts of the end-to-end query path allow the decision engine to make complexity predictions that would not be feasible by the client alone.**

***Index Terms*—SmartNICs, HPC data services, BlueField-2, decision engine, data query, data management**

I. Introduction

The availability of low-cost ARM and RISC-V IP cores has motivated several hardware vendors to include reprogrammable resources in hardware products that have traditionally only offered fixed functionality. For example, storage vendors sell computational storage devices (CSDs) that allow users to perform data transformations at the disk to assist in deduplication and error handling [1]. Similarly, network vendors are including user-programmable processing resources in their network cards and switches to offload collective communication operations [2], enhance security [3], [4], and present disaggregated storage to the host [5]. While the embedded processors in these devices may be an order of magnitude slower than host processors, vendors have demonstrated that there is great value in placing small pieces of embedded software in the hardware devices that are distributed throughout a computing platform [6].

Multiple vendors have released information and software development kits that enable end-consumers to implement their own application-specific software for these devices [7], [8]. As high-performance computing (HPC) systems researchers, we are interested in developing software that will enable us to migrate data management services from the host processors in a platform to the embedded processors distributed across its network and storage fabrics. Our work has focused on using Apache Arrow [9] to define a standard for processing in-transit data [10] and constructing data flows that autonomously processes in-transit data as it propagates through SmartNICs [11].

As more workloads are adapted to run in distributed network and storage devices, there is a growing need for query interfaces that allow end users to remotely inspect the dynamic content that individual devices manage. A key challenge in implementing a query interface is determining where in the system a query operation should execute. As illustrated in Figure 1, executing the query on the remote device (i.e., "offload") may minimize the data returned, but the computation may overwhelm its embedded processors and compromise its other duties. Deferring execution to the client (i.e., "push-back") may yield better query performance, but be hindered by the cost of retrieving a large amount of raw data.

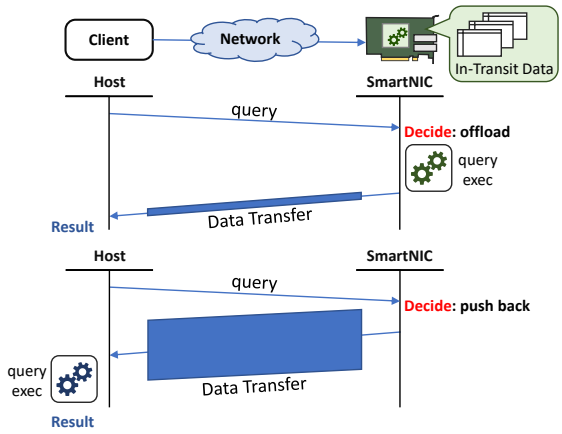

Fig. 1: Query execution can be offloaded or pushed back

In this paper we explore the design of a decision engine that resides on the target SmartNIC and uses predictive scheduling to determine if a query should be offloaded or pushed back to the client. This engine uses the Apache DataSketches library [12] to rapidly characterize in-transit data and makes predictions about different overheads of the two possible paths. Our work demonstrates that (1) embedding the decision login in the SmartNIC adds value as in-transit data content can be leveraged in the decision making process and (2) predictions can be made quickly enough to justify offloading on embedded hardware.

II. Implementing Data Services in SmartNICs

In HPC workflows researchers rely on *composable data service libraries* [13] to route data between parallel jobs in a computing platform. These libraries provide a variety of components that enable users to construct application-specific services for the workflow that can be hosted in other compute nodes in the platform. Examples of data services include staging data in memory for job-to-job handoffs in workflows, transforming data to make it easier to exploit, and converting from in-memory to on-disk formats to insulate jobs from I/O overheads.

A. SmartNICs

Recently, multiple network vendors have produced programmable network interface cards (SmartNICs) that feature embedded processing, memory, and storage resources that can be programmed by end users. For example, the NVIDIA BlueField-2 DPU [7] network card includes 8 ARM A72 cores, 16GB of DRAM, and 60GB of flash storage. While the card was initially designed to serve as an on-path network security device for cloud infrastructure, it can be configured to operate in an off-path mode where the card appears as an embedded compute node in the network fabric. Given that the VPI variant of the BlueField-2 can support 100Gb/s InfiniBand or Ethernet, HPC system researchers are building experimental platforms that use the BlueField-2 as the high-speed NIC for compute nodes [14].

In previous work [11] we have asserted that SmartNICs provide an appealing substrate for hosting low-volume data management services because they offer an isolated environment that has network access and is in close proximity to host applications. As a means of exploring this space we extended the Faodel [15] composable data service library to allow communication endpoints to be placed in either the hosts or SmartNICs of an HPC platform. We leveraged the Apache Arrow [9] library to provide a robust environment for representing and processing in-transit data. Arrow employs a tabular data model and includes a computational framework that automatically maps data-parallel operators to multiple cores and SIMD hardware.

B. Particle-Sifting Example

A particle-sifting example from our previous work provides an example of how we can offload data services into the SmartNICs of compute nodes and implement useful data processing pipelines. As illustrated in Figure 2, a particle simulation tool runs on an array of compute nodes and periodically generates write-optimized data that is converted to an Arrow tabular format, serialized, and injected to the local SmartNICs. Once a SmartNIC has collected a sufficient amount of data, it reorganizes the data into smaller tables with similar particle IDs and then transmits each partition to a corresponding neighbor. As data passes through the system, data becomes more organized until it is in a form that can be written to disk or consumed by other applications. Offloading this work to the SmartNICs allows researchers to dedicate a compute node's resources to solving compute-bound problems instead of the asynchronous behaviors of I/O libraries.

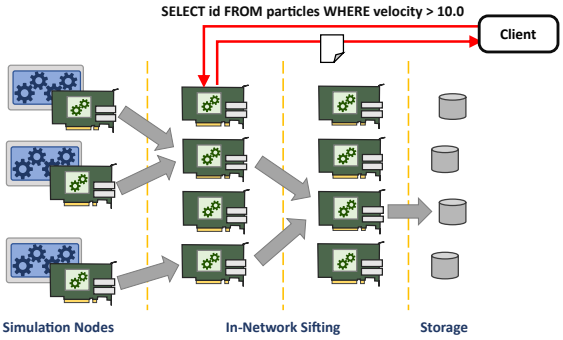

Fig. 2: A query interface provides users with a way to inspect in-transit data

C. Dynamic Query Interface

An important but missing piece of this architecture is the ability for users to dynamically inspect the in-transit data of the SmartNICs distributed throughout the system. In addition to serving as a mechanism for debugging data flows, query interfaces offer an opportunity for users to gain insight from the simulation's live results and make better-informed steering decisions for the overall workflow. The most straightforward way to implement a query interface when using composable data service libraries is through the creation of remote procedure calls (RPCs). While RPCs provide an open environment for interacting with a remote node, we recognize that it can be tedious to define and insert RPC function handlers that can support a wide range of domain-specific queries. As such, we advocate for an approach that allows users to channel SQL queries over a single RPC handler. In addition to simplifying RPC complexity, this approach allows users to interrogate their data in a well-known language and leverage a wide variety of existing tools.

III. Dynamic Query Decision Engine Placement

One of the advantages of a SQL-based query interface for inspecting in-transit data on SmartNICs is that system designers have the freedom to determine where different portions of the query plan execute in the system. In queries that return a small portion of the available data, it may be beneficial to *offload* the query to the SmartNIC to minimize data transfers. Conversely, in queries that are complex or do not reduce the original data substantially, it may be better for the SmartNIC to simply return the raw data and *push-back* execution to the client. A *decision engine* is therefore required to assess current conditions and predict which path is best.

Given the dynamic workloads and fluctuating resource environment within a SmartNIC, an important question arises for system developers: Should the decision to offload or push back be made at the client or the SmartNIC? We assert that the SmartNIC is better poised to make this decision because it has better access to information that affects how a query is fulfilled. Specifically, the SmartNIC has local information

about runtime characteristics that may improve the predicted amount of time required to process a query. More importantly though, the SmartNIC has direct access to the content of the in-transit data it maintains. This rapidly-changing data is valuable for predicting query overheads and is challenging to propagate to clients without incurring additional infrastructure costs.

A. Delivery of Query Workloads

When considering design alternatives for a query interface, it is beneficial to be more specific about the mechanisms by which query workloads are delivered in a distributed system. Database management systems (DBMSs) process queries through a multi-layered approach, converting a query into different representations at different system layers [16], [17]. The first step in this process is to parse the query statement and convert it into a logical plan. This operation may involve significant computational overhead [18], [19] and is therefore best performed by the client. A logical plan is then delivered to a target through a network operation such as an RPC. A target will retrieve relevant tables, execute its portion of the logical plan, and then return results to the client. Finally, client software evaluates any remaining pieces of the logical plan and return results to the user.

The open source community has developed a number of libraries that simplify the process of building query services. In our work we leverage the following stack for logical query declaration, execution, and transmission:

Substrait [20] aims to offer a cross-language specification for data computing operations and a standardized format for query plans. This facilitates the transformation of logical query plans into binary substrait plans in formats like protobuf or JSON, which are suited for network transfer.

Apache Arrow provides a data standard for the environment that allows developers from multiple realms to represent, process, and query their data. It utilizes a tabular data model, which is suitable for many of our applications due to its thriving open-source community, and includes built-in compute operators with SIMD optimizations that can map to parallel processors. Most importantly, the Acero component of Apache Arrow provides semantics for constructing and executing composable logical query declarations in C++.

Faodel provides a key/blob API that enables the exchange of RDMA-transportable objects across a diverse set of communication endpoints. Furthermore, it has the ability to trigger computations on objects located at remote endpoints using user-defined functions. This feature facilitates the transmission of query workloads to SmartNICs and the return of either complete or intermediate results depending on the execution decisions made by the SmartNICs.

B. Related Work

Dynamically offloading functions to SmartNICs can be implemented using different scheduling approaches. One such approach is reactive scheduling, where placement decisions are based on the scheduler's current internal state to handle incoming workloads. An example of this approach is the utilization of the deficit round-robin algorithm [21] for scheduling distributed applications [22]. Another approach is predictive scheduling, which aims to forecast the cost of a workload request relative to the current context. Data stream processing [23], search engines, online data mining [24], [25], and query optimizers in databases [26] commonly adopt this approach to determine beneficial plans for executing queries on host systems. Our focus in this paper is on the second approach for the decision engine's implementation, given its potential to enable dynamic offloading using mature data management techniques in embedded systems.

IV. End-to-End Cost Analysis

Developing an offload/push-back decision engine for SmartNICs requires an understanding of the dominant overheads in the end-to-end query process. As depicted in Figure 3, there are three types of overheads in the data path: serialization, network transfers, and query execution.

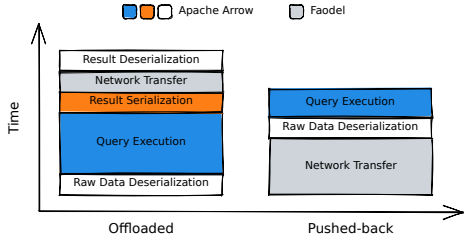

Fig. 3: Conceptual time breakdown for offloaded and pushed-back cases with components handled by different data service libraries

A. Serialization Time

Each SmartNIC in the system manages a collection of in-transit data objects in on-card memory. A data object holds one or more Apache Arrow tables that have been serialized to the Arrow inter-process communication (IPC) format. As such the data must be deserialized into an Arrow table before a query can execute, whether the work is done on the SmartNIC or at the client. Fortunately, Arrow's IPC format is designed to allow zero-copy reconstruction of an Arrow table from an (uncompressed) IPC buffer simply by establishing reference pointers into the buffer. Figure 4 (left) demonstrates that deserialization performance is independent of table size, remaining nearly constant for both hosts and SmartNICs.

In the case where a query is offloaded to a SmartNIC, it is necessary to serialize the results into an IPC buffer in order to allow it to be transmitted to the client. As depicted in Figure 4 (right), serialization time depends on table size.

B. Network Transfer Time

While the offloaded and pushed-back cases both transmit serialized data from the SmartNIC to the client, query selectivity may significantly reduce the amount of data returned in the offloaded case. To create an estimate of network overhead, we measured the round-trip time for a client to request varying sizes of data from the SmartNIC using Faodel. As shown

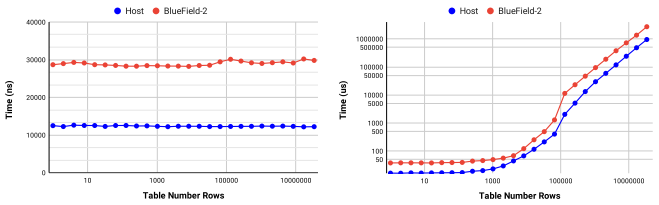

Fig. 4: Arrow table deserialization and serialization times

in Figure 5, overhead is constant until tables are larger than 64KB.

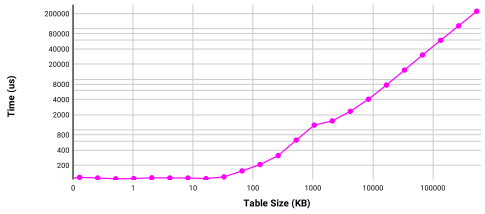

Fig. 5: Time required to request different return sizes from a remote SmartNIC with Faodel

C. Query Execution Time

While we expect query execution time to be the dominant cost in our workload, it is challenging to estimate query overhead because it largely depends on the characteristics of both the input table and the query. The size of the input table can linearly influence data scan time for various operations once the compute engine has sufficient work to occupy available cores. However, users may embed a wide variety of computations in a query using operations that have different overheads. For example, we observed that a simple scan with the "power" operation required substantially more CPU time than another with logical operations such as "and" on a BlueField-2 SmartNIC. Additionally, the relationship between query conditions and input data distribution can impact the computational overhead, as seen when rearranging the order of multiple filtering conditions optimizes the data reduction benefit from one filter to the next.

In database systems, the complexity of a query workload is typically delineated through the process of cardinality estimation [27], where the call count of every operation in a query and the query result size are evaluated based on statistical information (e.g., histogram and distinct counting) gathered on the data without executing the query. Unlike cardinality estimators in mature database systems (e.g., PostgreSQL [28] and Spark SQL [29]) designed to run on host systems, implementing one to run on SmartNICs emphasizes efficiency due to the limited resources these devices have. Additionally, the in-transit nature of the data managed by SmartNICs necessitates the progressive updating and revising of each histogram as new data is received, merged, or migrated.

Once the complexity of a query workload is captured, we can combine the resource availability (e.g., available thread count) of the target system (e.g., SmartNIC or host) to predict the execution time of the workload.

V. Decision Engine Implementation

As a means of exploring performance trade-offs in a function system, we supplemented our current SmartNIC software stack with a query interface. This interface includes a decision engine that employs predictive scheduling to determine if incoming queries should be offloaded to the SmartNIC or pushed back to the client. A key challenge in this work is making accurate predictions of the serialization, network transfer, and query execution overheads associated with processing a query. In this section we discuss the mechanisms by which we predict these overheads and evaluate accuracy for workflows that operate on particle datasets.

A. Predicting Serialization Time

While deserialization operates at a constant rate, serialization overhead depends on the size and content of an Arrow table. To customize serialization overhead prediction to our particle data use case, we constructed a synthetic corpus of particle data tables and measured the amount of time required to serialize each table on a BlueField-2 SmartNIC. Tables employed 9 fields and ranged in length from 1 to 2^{25} rows. Utilizing a random forest regression for training, we achieved a prediction accuracy with an error rate of less than 7% (Figure 6) for particle data serialization.

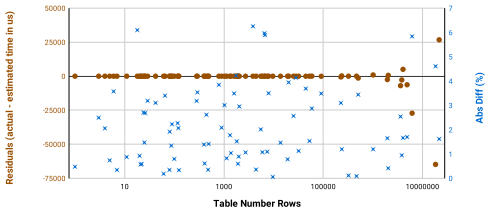

Fig. 6: Differences in actual vs. estimated serialization time for Arrow tables of various row counts on the BlueField-2

B. Predicting Network Transfer Time

We used the Faodel library for data delivery and invoking query handling at the remote SmartNIC. While it is possible to predict the network transfer time required for a query workload based on both the schema of the resultant table object and the number of rows in the table, our approach was to anchor the prediction on the *size* of the table object to be transferred. This approach simplifies network time modeling and insulates it from future changes in the table schema.

In scenarios where the query workload is executed on the SmartNIC, the size of the serialized object can be predicted by considering the number of rows in the resulting table (predicted in the following section), as depicted in Figure 7. Conversely, when the query workload is to be pushed back for execution, the total serialized size of the data objects referenced by the query can be determined by aggregating their respective sizes.

To construct a training dataset for predicting network transfer time based on the data size to be transferred, we measured the round-trip time of requests dispatched to a SmartNIC to

fetch local IPC buffers of varying sizes. We trained our model using the random forest regression algorithm. Performance results for the communication between an HPC compute node and a local SmartNIC for a randomly generated table test set are illustrated in Figure 8. This model consistently maintained error rates within single-digit percentages on the test set.

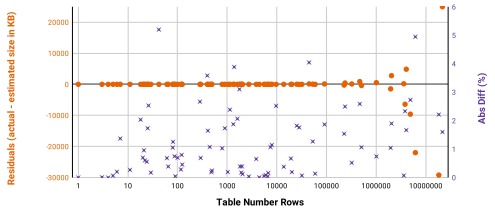

Fig. 7: The residuals and percentage differences between the actual and estimated serialization sizes

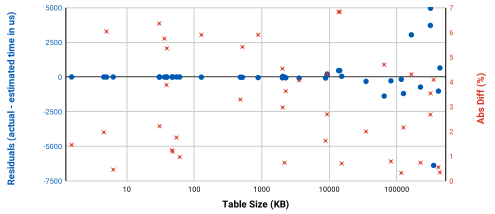

Fig. 8: Residuals and absolute percentage differences between measured and estimated network transfer times against table size in SmartNIC-hosted table retrieval

C. Predicting Query Execution Time

To predict query execution time we implemented an efficient cardinality estimator for use on SmartNICs. It generates an operation vector, representing the call count of every involved query operation, to encapsulate the complexity of each query workload, as illustrated in Figure 9. For maintaining statistical information of in-transit data, we used the Theta Sketch [30] and KLL Sketch [31] algorithms from the Apache DataSketches library [12] to derive the distinct counting and histogram statistics of particle data tables, respectively. One benefit of this library that is relevant for processing in-transit data streams is that it includes an interface for updating created statistics over time. Furthermore, the library provides parameters that enable the fine-tuning of the estimation's accuracy, allowing us to evaluate the trade-offs between estimation performance and system resource consumption.

Our cardinality estimator currently supports queries that involve filtering, projection, aggregation, or any combination of these operations. The estimator is also capable of estimating reducible conditions to maximize the value of statistics generated for individual columns. Moreover, the estimator can estimate queries that depend on multiple data sources. This capability is particularly significant as it allows for the estimation of workloads querying multiple data table partitions simultaneously on a single SmartNIC. Figure 10 illustrates the performance of the cardinality estimator using randomly

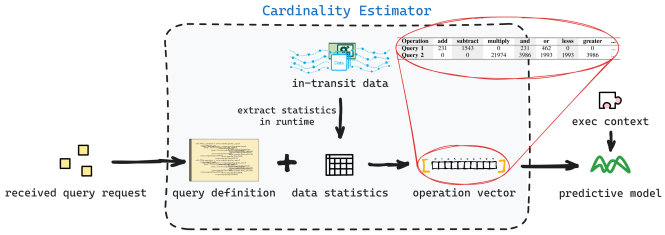

Fig. 9: The process of cardinality estimation to generate vectors of operations as input to the model to predict query execution time

generated test queries on data tables of varying sizes. SQL representations of three queries are listed below.

```
(Q3) SELECT x,
       power(vx, 2) + power(vy, 2) + power(vz, 2)
       AS square_sum
     FROM particles
     WHERE square_sum <= 1310.0

(Q4) SELECT *
     FROM particles
     WHERE 3 + vx * 2 <= 100 OR sqrt(vy) > 20

(Q8) SELECT COUNT(id)
     FROM particles
     WHERE x >= 0.7 AND y < 0.3 AND z <= 0.1
     GROUP BY particle_id
```

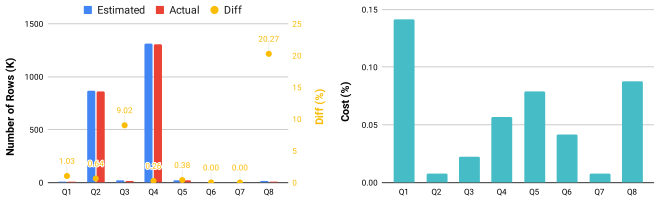

Fig. 10: Performance of cardinality estimation. The left figure shows the absolute percentage difference between the estimated and actual output rows of each test query. The right figure displays the CPU time for estimation as a percentage of the CPU time for query execution on a BlueField-2 SmartNIC.

The performance evaluation shows that the majority of estimations have an error rate within 1% of the actual cardinality. However, some estimations may experience higher errors due to the aggregation of multiple statistics' biases. For instance, in query Q8, the estimator uses histogram statistics for the three filtering conditions and distinct counting statistics for the one aggregation condition. Furthermore, it requires applying the distinct counting statistics to a subset of the table that resulted from filtering, which can introduce significant errors if the data itself is biased. Figure 10 (right) emphasizes the efficiency of conducting cardinality estimation on the BlueField-2 SmartNIC, which is crucial for SmartNICs to handle the functions of the decision engine efficiently.

To convert an operation counts vector, as depicted in Figure 9, into execution time, we again utilized machine learning techniques. We used query templates to generate sufficient data for model training. Each template was a C++ logical query plan with placeholders that could be filled with randomly generated constants to produce concrete logical plans. Each

record in our training dataset included operation counts, the number of rows in the queried table, the thread counts, and the actual time for workload execution. We employed the random forest model to train the data. Figure 11 illustrates the prediction performance on a set of test queries. It is worth noting that our estimator supports operation counting for sub-conditions, which enables fine-grained execution time predictions for complex, composable query workloads.

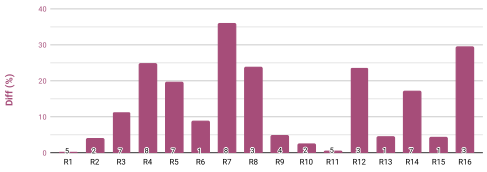

Fig. 11: Difference between the actual and estimated execution times for test queries on the BlueField-2. Thread count for measurement and prediction are indicated at base of each bin.

VI. Opportunistic Query Execution Case Studies

As a means of validating that our approach achieves a benefit for end users, we conducted two query execution case studies that perform a bounding-box threshold of particle data. We used our prediction models to estimate the time consumption associated with each activity in the offload and push-back paths, and then measured the actual time of these activities consumed by end-to-end queries. By comparing these measurements, we can evaluate the effectiveness and accuracy of our decision engine, which makes scheduling decisions based on aggregating the predicted overheads listed in this paper. A host with two Intel Xeon 16-core E5-2698 CPUs running at 2.30GHz and a Bluefield-2 SmartNIC were used for these experiments.

In the first study, we analyze a query applied to a particle dataset comprised of 6,177,731 rows:

```
SELECT * FROM particles
  WHERE x >= 0.7 and y < 0.3 and z <= 0.1
```
(CQ1)

The actual execution of this query results in 55,517 rows, comprising 0.9% of the original data. The cardinality estimator predicts 55,036.7 rows, showing a difference of 0.865% from the actual row count. For this particular query, the operation vector is generated as follows:

TABLE I: The operation vector produced for the case study Query CQ1

and_kleene	filter	greater_equal	less	less_equal	select	table_rows
12355500	6177730	6177730	6177730	6177730	55036.7	6177731

The second study uses a query slightly different from the first one to examine the crossover point where offloading and pushing back result in similar execution costs:

```
SELECT * FROM particles
  WHERE x >= 0.5 and y < 0.55 and z <= 0.67
```
(CQ2)

Executing this query on the same dataset yields 1,136,847 rows, which represents 18.4% of the total row count. The cardinality estimator predicts a return of 1,152,860 rows, indicating a minor discrepancy of 1.41%.

The estimated and actual time consumption, aggregated from the time factors for each of the two scenarios, are depicted in Figure 12. The query execution time for the SmartNIC is both measured and estimated using six threads, whereas, for the host, it is measured and estimated utilizing 32 host threads. This bias is intentionally introduced to account for the host's superior availability of computing resources. Despite this adjustment, the comparison reveals that for the first query, choosing offloaded execution significantly reduces execution latency by 74.64% due to low-percentage selectivity. This outcome can be attributed to the high network transfer cost that dominates the total execution latency in the scenario of pushed-back execution. As for the second query workload, execution latency is comparable whether conducted on the SmartNIC or the host. Specifically, while the estimation slightly leans towards pushing back, keeping execution on the SmartNIC reduces latency by 1.38% due to higher-percentage selectivity.

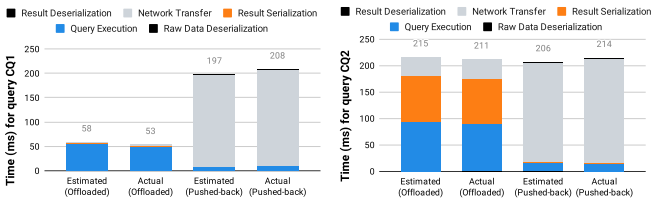

Fig. 12: Analysis of time consumption for offloaded vs. pushed-back execution with case study queries

The case studies show that high-precision cost prediction for query workloads can be achieved efficiently. In our cases, the estimation process uses less than 2% of the CPU time required for executing the corresponding query workload on the BlueField-2. Despite being resource-efficient, our approach exhibits adaptability to dynamic workloads and execution environments on SmartNICs, which is valuable for practical HPC workflows.

VII. Summary and Future Work

Optimizing data query workload execution via dynamic offloading across systems of different architectures is challenging. However, data management techniques open avenues for developing a workload placement decision engine for SmartNICs, tailored to the HPC data processing landscape. Importantly, our decision engine's predictive approach may extend to embedded systems with similar hardware capabilities (e.g., computational storage devices) to exploit data service offloading benefits. It is worth noting that several factors, including network bandwidth variation, system resource fluctuation, and performance interference among different data services, could impact dynamic offloading performance. Addressing these factors remains vital in our future work to enhance the efficiency of dynamic queries on SmartNICs.

ACKNOWLEDGMENT

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research under Field Work Proposal Number 20-023266. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

REFERENCES

[1] S. Bates, "Computational Storage Real World Deployments," https://www.snia.org/educational-library/fms-2020-computational-storage-track-computational-storage-real-world, Nov. 2020, publisher: Flash Memory Summit.

[2] R. L. Graham, D. Bureddy, P. Lui, H. Rosenstock, G. Shainer, G. Bloch, D. Goldenerg, M. Dubman, S. Kotchubievsky, V. Koushnir *et al.*, "Scalable hierarchical aggregation protocol (sharp): A hardware architecture for efficient data reduction," in *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*. IEEE, 2016, pp. 1–10.

[3] S. Grant, A. Yelam, M. Bland, and A. C. Snoeren, "Smartnic performance isolation with fairnic: Programmable networking for the cloud," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 681–693.

[4] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung *et al.*, "Azure accelerated networking:{SmartNICs} in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 51–66.

[5] J. Min, M. Liu, T. Chugh, C. Zhao, A. Wei, I. H. Doh, and A. Krishnamurthy, "Gimbal: enabling multi-tenant storage disaggregation on smartnic jbofs," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 106–122.

[6] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana, "E3:{Energy-Efficient} microservices on {SmartNIC-Accelerated} servers," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 363–378.

[7] I. Burstein, "Nvidia data center processing unit (dpu) architecture," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–20.

[8] J. Dastidar, D. Riddoch, J. Moore, S. Pope, and J. Wesselkamper, "Amd 400g adaptive smartnic soc–technology preview," *IEEE Micro*, 2023.

[9] G. Lentner, "Shared memory high throughput computing with Apache Arrow," in *Proceedings of the Practice and Experience in Advanced Research Computing on Rise of the Machines (learning)*, 2019.

[10] J. Liu, C. Maltzahn, M. L. Curry, and C. Ulmer, "Processing particle data flows with smartnics," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2022, pp. 1–8.

[11] C. Ulmer, J. Liu, C. Maltzahn, and M. L. Curry, "Extending composable data services into smartnics," in *Proceedings of the 2nd Workshop on Composable Systems*, 2023.

[12] L. Rhodes, K. Lang, A. Saydakov, J. Thaler, E. Liberty, and J. Malkin, "Apache DataSketches: A software library of stochastic streaming algorithms," https://datasketches.apache.org/.

[13] S. Ramesh, A. D. Malony, P. Carns, R. B. Ross, M. Dorier, J. Soumagne, and S. Snyder, "Symbiosys: A methodology for performance analysis of composable hpc data services," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2021, pp. 35–45.

[14] T. Kordenbrock, G. Templet, C. Ulmer *et al.*, "Viability of s3 object storage for the asc program at sandia." Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Sandia ..., Tech. Rep., 2022.

[15] C. Ulmer, S. Mukherjee, G. Templet, S. Levy, J. Lofstead, P. Widener, T. Kordenbrock, and M. Lawson, "Faodel: Data management for next-generation application workflows," in *Proceedings of the 9th Workshop on Scientific Cloud Computing*, 2018.

[16] G. Graefe, "Query evaluation techniques for large databases," *ACM Computing Surveys (CSUR)*, vol. 25, no. 2, pp. 73–169, 1993.

[17] A. Deshpande, Z. Ives, V. Raman *et al.*, "Adaptive query processing," *Foundations and Trends® in Databases*, vol. 1, no. 1, pp. 1–140, 2007.

[18] G. Douglas and R. Lawrence, "Improving sql query performance on embedded devices using pre-compilation," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, 2016, pp. 961–964.

[19] F. Waas and C. Galindo-Legaria, "Counting, enumerating, and sampling of execution plans in a cost-based query optimizer," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000, pp. 499–509.

[20] J. Nadeau, "Substrait: Cross-Language Serialization for Relational Algebra," https://substrait.io/.

[21] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," in *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 1995, pp. 231–242.

[22] M. Liu, T. Cui, H. N. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto smartnics using iPipe," *Proceedings of the ACM Special Interest Group on Data Communication*, 2019.

[23] C. C. Aggarwal and P. S. Yu, "A survey of synopsis construction in data streams," *Data streams: models and algorithms*, pp. 169–207, 2007.

[24] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm," in *Proceedings of the 16th International Conference on Extending Database Technology*, 2013, pp. 683–692.

[25] A. Metwally, D. Agrawal, and A. E. Abbadi, "Why go logarithmic if we can go linear? towards effective distinct counting of search traffic," in *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, 2008, pp. 618–629.

[26] Y. E. Ioannidis, "Query optimization," *ACM Computing Surveys (CSUR)*, vol. 28, no. 1, pp. 121–123, 1996.

[27] K. Youssefi and E. Wong, "Query processing in a relational database management system," in *Fifth International Conference on Very Large Data Bases, 1979.* IEEE Computer Society, 1979, pp. 409–410.

[28] J. D. Drake and J. C. Worsley, *Practical PostgreSQL.* " O'Reilly Media, Inc.", 2002.

[29] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, 2015, pp. 1383–1394.

[30] A. Dasgupta, K. Lang, L. Rhodes, and J. Thaler, "A framework for estimating stream expression cardinalities," *arXiv preprint arXiv:1510.01455*, 2015.

[31] Z. Karnin, K. Lang, and E. Liberty, "Optimal quantile approximation in streams," in *2016 ieee 57th annual symposium on foundations of computer science (focs)*. IEEE, 2016, pp. 71–78.