An FPGA based Network Interface Card with Query Filter for Storage Nodes of Big Data Systems

Ying Li
University of Electronic Science and Technology of China
e-mail: lylyling_wulala@qq.com

Jinyu Zhan*
University of Electronic Science and Technology of China
Corresponding author: zhanjy@uestc.edu.cn

Wei Jiang
University of Electronic Science and Technology of China
e-mail: weijiang@uestc.edu.cn

Junting Wu
University of Electronic Science and Technology of China
e-mail: 1473692846@qq.com

Jianping Zhu
Tencent Technology Shenzhen Company Ltd
e-mail: felixzhu@tencent.com

Abstract— In this paper, we are interested in improving the data processing of storage and computing separated Big Data systems. We propose an Field Programmable Gate Array (FPGA) based Network Interface Card with Query Filter (NIC-QF) to accelerate the data query efficiency of storage nodes and reduce the workloads of computing nodes and the communication overheads between them. NIC-QF designed with PCIe core, query filter and NIC communication can filter the original data on storage nodes as an implicit coprocessor and directly send the filtered data to computing nodes of Big Data systems. Filter units in query filter can perform multiple SQL tasks in parallel, and each filter unit is internally pipelined, which can further speed up the data processing. Filter units can be designed to support general SQL queries on different data formats and we implement two schemes for TextFile and RCFile separately. Based on TPC-H benchmark and Tencent data set, we conduct extensive experiments to evaluate our design, which can achieve averagely up to 46.91% faster than the traditional approach.

***Index Terms*—Storage and computing separated Big Data systems; Query filter; Network Interface Card; FPGA**

I. Introduction

With the rapid development of big data techniques, the amounts of data in emerging big data applications has exceeded the GB level, but been reaching TB, even PB or EB level. Traditional big data architectures are not suitable for such magnitude of data. These architectures employ mixed role nodes suffers from bad scalability and complex transmission scheduling further aggravate the performance burden caused by the magnitude of data. In traditional architectures, each node may have many different roles and a physical machine may take responsibility for computing tasks under high pressure transmission workloads. This will make computing resources and storage resources tightly coupled with each other. These problems also cause the intention from industries, in recent years, companies like Tencent [1], IBM [2], Facebook [3], and Microsoft [4] have developed storage and computing separated architectures to overcome the shortages in scalability. In these designs, storage nodes and computing nodes work independently with each other and can be easily scaled.

In the existing big data systems, using FPGA as a coprocessor is an effective implementation. The authors of [5] developed an Ibex framework to accelerate the data processing by putting the FPGA between the CPU and Solid State Disk(SSD), and performing filter operations such as selection and projection. However, SSD with low capacity are not suitable for storage nodes to store mass data. And the current FPGA is actually unable to obtain directly data from the hard disk. The authors of [6] [7] proposed a database acceleration method based on CPU and FPGA memory sharing, which can accelerate the processing on computing nodes, but the communication overheads between computing nodes and storage nodes are still very high. Microsoft [8] deploys FPGAs between NICs and switches to accelerate network forwarding and storage virtualization, but the FPGAs cannot deal with SQL tasks. To further improve the query performance, some studies on data processing are proposed. The authors of [9] designed an FPGA-based data filter for RDF triples. The authors of [10] designed a skeleton automaton for data query to filter XML. Both of them do not support general SQL queries. In these designs, the coprocessors can help storage nodes dropping most of those unnecessary data packages before transmission, and reducing the bandwidth pressure. The authors of [11] [12] proposed an FPGA based dynamic configurable plug-in method for querying the database, to perform some general queries. But this method deals with different queries by the FPGA reconfiguration.

Although these design success in increasing the processing speed over the entire system, it still remains many challenges. Using hardware accelerating coprocessors can benefit in amortizing the workloads and also causes a series of problems such as transmission gap and scheduling overheads. For example, the I/O bandwidth between the storage nodes and computing nodes fluctuates greatly and the computing resources are centralized while the calculation cost is very high. Moreover, simply embedding coprocessors into big data architectures may introduce more encumbrances due to the magnitude of data. At last, these designs are all used to speed up very specific SQL tasks. How to support general SQL queries in different storage formats, such as column-based storage, row-based storage and key-value storage, is still an urgent problem. According to the current works, reconfiguring the circuits to adapt to different SQLs will result in a certain amount of time overheads and shorten the FPGA lifetime.

As a supplementary work, we propose an FPGA-based Network Interface Card with query filters (NIC-QF) for storage and computing separated Big Data systems, which can reduce corresponding communication overheads and workloads of the computing node. We use FPGA to design a NIC-QF for the storage nodes, by which the FPGA is integrated as a coprocessor to increase the processing efficiencies of storage nodes. The entire NIC-QF consists of PCIe core, query fiter and NIC communication. Query filter deals with different general SQL queries to filter data, which reduces data size and lowers communication overheads and workloads of computing nodes. Inside the query filter, several filter units are designed to perform multiple SQL tasks in parallel. We implement two schemes for filter units to support TextFile and RCFile. To avoid

978-1-7281-4123-7/20/$31.00

the FPGA reconfiguration for different SQLs, the SQL conditions are parameterized into boolean variables. We conduct extensive experiments to evaluate our approach on TPC-H benchmark and Tencent data set. To the best of our knowledge, this paper is the first work to implement query filter enhanced NIC by FPGA to accelerate the data processing of storage and computation separated big data systems.

The primary contributions of this paper are listed as follows.

- Propose a NIC-QF on storage nodes of big data systems to lower the communication overheads between storage nodes and computing nodes and the workloads of computing nodes.
- Present an FPGA-based implicit coprocessor architecture to reduce the CPU workloads of storage nodes.
- Formulate general SQL conditions of combined predicates into boolean variables to avoid reconfiguring FPGA.
- Implement two schemes inside the filter units for TextFile and RCFile formats.

The rest of this paper is organized as follows. Section II gives the system architecture of storage and computing separated big data systems considered in this paper. Section III and IV presents the designs of NIC-QF and filter unit. We evaluate the proposed design in Section V and draw the conclusion of this paper in Section VI.

II. System Architecture

In this paper we consider to accelerate the data processing of storage and computing separated big data systems. For comparison, we give both the traditional architecture and our improved architecture in Fig.1. In the traditional architecture shown in Fig. 1(a), when the storage node receives SQL query tasks from computing nodes, it requests the query data on the memory. If the data is not in memory, storage node will retrieve all of the original data in table from the hard disk and sent to computing nodes via NIC. The attributes not in the SQL conditions are considered as the unrelated data. Actually, there are a lot of data unrelated to SQL conditions in the original data, which occupies a large amount of communication overheads. The improved architecture is shown in Fig.1(b). Comparing with traditional architecture, the difference is marked in red. The original data is pre-filtered by filtering accelerator of NIC-QF. Only filtered data needs to be transmitted to the computing node, which results in less communication overheads and less workloads in computing nodes.

(a) Traditional architecture

(b) Our improved architecture

Fig. 1: Architecture comparison

III. NIC-QF Design

In this section, we present the design of NIC-QF. To reduce the CPU workloads, the NIC-QF implemented by FPGA is designed as an implicit coprocessor on the storage nodes. As shown in Fig. 2, NIC-QF is composed of PCIe core, query filter and NIC communication, and can speed up data queries and transmissions. The FPGA-based NIC-QF communicates with CPU using Direct Memory Access(DMA) protocol through PCIe interface. When receiving the SQL query requirements, CPU parses the SQL

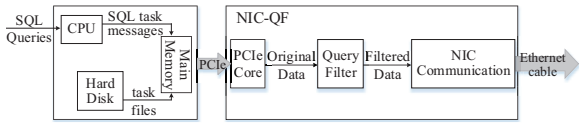


Fig. 2: NIC-QF design

queries into SQL task messages, and indicates the paths to the corresponding files to ensure that data can be taken out from the hard disk. SQL task messages and corresponding data are collected in the main memory and are sent to NIC-QF through PCIe. Data are filtered in query filter according to SQL task messages. Then NIC communication transmits the filtered data as frames by ethernet cable.

A. Query Filter

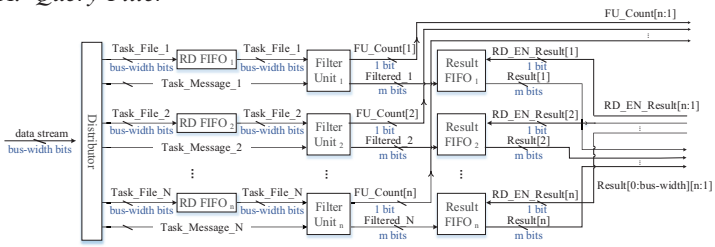


Fig. 3: Query filter

To improve filtering efficiency, query filter performs multiple filtering on the FPGA. As shown in Fig.3, query filter consists of a distributor, several Read FIFOs (RD FIFO), filter units and Result FIFOs, where different filter units deals with different SQL tasks at the same time. The distributor decomposes different task files from the continuous bus-width-bit data streams according to the total length of each file or the symbol of end-of-file and allocates them to corresponding RD FIFOs based on different storage formats. And the distributor also distinguishes SQL task messages from data streams to send to the corresponding filter units. Filter units work in parallel to deal with different SQL tasks, the number of which are restricted by FPGA onboard resources. Filter units can be designed to support different data formats. We have realized the schemes for TextFile and RCFile. Sometimes, it may occur that several result sets of filter units wait to be transmitted at the same time in NIC communication and the next tasks are also blocked. To solve this problem, we set a counter in each filter unit to calculate how many results match the SQL queries. Once the number of results reach the size that needs to be transmitted, $FU\_Count[n{:}1]$ are set to inform NIC communication of transmitting the SQL results stored in Result FIFOs.

B. NIC Communication

The NIC communication takes charge of ethernet transmission. As shown in Fig.4, we implement the encapsulation of the UDP protocol, IP protocol and MAC protocol in the modules of UDF_TX, IP_TX and MAC_TX, respectively. Then the filtered data can be sent to the network as frames through the ethernet cable.

$FU\_Count[n{:}1]$ and $Result[bus\text{-}width{:}0][n{:}1]$ are the inputs whereas $RD\_EN\_Result[n{:}1]$ is the output. To avoid disordering the filtered data of different tasks, $FU\_Count[n{:}1]$ is a n-bit

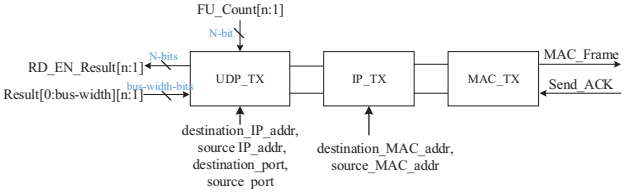


Fig. 4: NIC communication

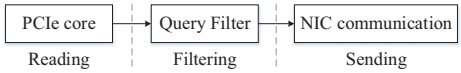


Fig. 5: Pipeline design

variable composed of the outputs of all filter units in Query Filter to inform UDP_TX to encapsulate the results of filter units from the corresponding Result FIFOs. Since UDF_TX module, IP_TX module and MAC_TX module are independent, the protocol encapsulations of the UDP, IP and MAC can be pipelined.

C. Pipeline Design

Since PCIe core, query filter and NIC communication are independent, we design the pipeline of our NIC-QF to improve the overall performance of SQL tasks. There are three major stages in the pipeline of NIC-QF shown in Fig.5.

1) Reading: Reading data from main memory through PCIe.
2) Filtering: Filtering data in filter units.
3) Sending: Encapsulating and transmitting data to the network.

When data arrives through PCIe interface, it will enter the filtering stage. Once the results reach the amount of a packet, the protocol encapsulation and data transmission will occur.

IV. Design of Filter Unit

In this section, we present the design of filter unit in detail. We designed two filtering schemes for TextFile and RCFile.

A. Filter Unit

A filter unit consists of a data projection, a processing unit group and a checker in Fig.6(a). The data projection decomposes the attributes related or unrelated to the SQL conditions from the tuples in data streams and send them to different processing units. One tuple is a record in database tables. The processing unit group is composed of several operator units and a storage unit. The attributes of tuples related to the SQL conditions are filtered in the operator units whereas the attributes unrelate to the SQL conditions are stored in the storage unit. The checker determines whether this tuple meets the SQL conditions.

1) Storage Format: The filter unit can support data in different storage formats by different implementations, such as row-based storage, column-based storage and key-value storage. We have implemented the filtering for the two most common storage formats (TextFile and RCFile) in big data systems. According to the discrepancy of storage formats, the implementations of data projection and processing unit are different.

2) SQL Condition: SQL queries contain multiple atomic conditions, which are combined by logical operators ('AND' and 'OR'), as following.

$$\textbf{WHERE } col_1 > \alpha \textbf{ AND } col_2 = \beta \textbf{ OR } col_3 < \gamma$$

Each atomic condition contains the related attribute (col_1, col_2, col_3), an comparison operator($>$, $=$, $<$) and a target value (α, β, γ), which can be formulated into a boolean parameter to support. Hence, we can formulate the SQL conditions of combined predicates into boolean variables as task messages to the checker.

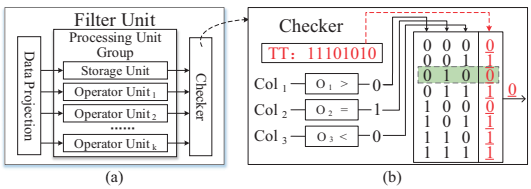


Fig. 6: Filter unit

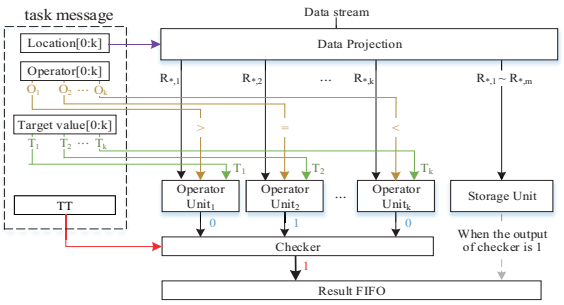


Fig. 7: Filtering Scheme for TextFile

3) Checker: To avoid reconfiguring the circuits according to different SQLs as the reference [13], combined predicates of different SQL conditions are formulated into boolean variables. The truth table of the combined predicates is parameterized into a variable called TT, as a part of task messages and stored in checker. The Fig. 6(b) shows how the checker determines whether a tuple meets the SQL conditions in one clock according to TT. Each attribute related to SQL conditions is filtered by a operator unit. Each operator unit has one boolean result that indicates whether data meets an atomic SQL condition. The results corresponding to n atomic conditions compose a 2^n-bit boolean TT in the checker.

B. Filtering Scheme for TextFile

We present the implementation for TextFile storage format as shown in Fig.7. TextFile is row-based storage format as shown in Fig.8(a). Each tuple is stored contiguously and is distinguished by a tuple terminator, and the attributes in each tuple are distinguished by attribute separators. The length of each data is variable.

In the implementation of filter units for TextFile, data passes in stream. Generally, the width of a tuple is wider than that of the PCIe data bus. Therefore, it is impossible to process one entire tuple in a clock cycle. Thus, one tuple is divided into several bus-width data, which contains one or more attributes. In data projection, the tuples are recognized by the tuple terminator like '/n' while the attributes are recognized by the attribute separator like '/t'. The attributes related to the SQL conditions are sent to the corresponding operator units for comparison, while the attributes unrelated to the SQL conditions are stored in the storage unit.

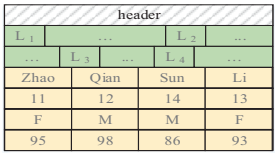
The operators and target values in SQL task messages are assigned to the corresponding operator units. Operator units support various operators like $==$, $!=$, $<$, $>$, $<=$, $>=$. Each operator unit is responsible for each attribute, whose output indicates whether the attribute meets the corresponding atomic condition. Once the checker receives the outputs of all operator units and determines whether the tuple meets the conbined SQL conditions, the storage unit will save or discard the tuple into the result FIFO.

C. Filtering Scheme for RCFile

The Fig.9 shows the implementation for RCFile storage format. RCFile is similar to column-based storage format as shown in Fig.8(b). A file in RCFile format consists of data identification information, metadata information, and data segments. The same attribute is consequent and length of each attribute is stored in metadata information.

Name	Age	Gender	Grade
.....			
Zhao	11	F	95
Qian	12	M	98
Sun	14	M	86
Li	13	F	93
.....			

(a) Format of TextFile



(b) Format of RCFile

Fig. 8: Storage format

TABLE I: SQL QUERIES

Q1	**SELECT** $*$ **FROM** test **WHERE** $L\_PARTKEY > 0$ **AND** $L\_SUPPKEY > 0$ **AND** $L\_LINENUMBER > 0$ **AND** $L\_TAX = 0$ **GROUP BY** $L\_SUPPKEY$
Q2	**SELECT** $*$ **FROM** test **WHERE** $L\_PARTKEY > 0$ **AND** $L\_SUPPKEY > 5000$ **AND** $L\_LINENUMBER > 0$ **AND** $L\_TAX >= 0$ **GROUP BY** $L\_SUPPKEY$
Q3	**SELECT** $*$ **FROM** test **WHERE** $L\_PARTKEY > 0$ **AND** $L\_SUPPKEY > 0$ **AND** $L\_LINENUMBER > 0$ **AND** $L\_TAX > 0$ **GROUP BY** $L\_SUPPKEY$
Q4	**SELECT** $*$ **FROM** test **WHERE** $USER\_IP =' 192.168.0.1'$ **AND** $PORT > 31600$ **AND** $SEND\_BYTES >= 0$ **AND** $CDN >= 0$ **GROUP BY** CDN
Q5	**SELECT** $*$ **FROM** test **WHERE** $USER\_IP =' 192.168.0.1'$ **AND** $PORT = 31600$ **AND** $SEND\_BYTES > 10000$ **AND** $CDN >= 0$ **GROUP BY** CDN
Q6	**SELECT** $*$ **FROM** test **WHERE** $USER\_IP =' 192.168.0.1'$ **AND** $PORT <= 31600$ **AND** $SEND\_BYTES > 0$ **AND** $CDN >= 0$ **GROUP BY** CDN

Data of each column passes in stream through the filter units for RCFile and data projection is simpler than that of TextFile filter units. Data projection can not only decompose the data segments of the file but also the length information in the metadata information of RCFile. Unlike the filter units for TextFile, RCFile filter units can obtain the total length of each columns in metadata information that indicates different columns. Therefore, the corresponding attributes can be continuously sent and stored to the corresponding FIFOs. A FIFO delivers one-bus-width data to a separator, which contains several attributes of a continuous storage column. In separator, each attribute is taken out from one-bus-width columns. Data is processed completely in parallel. The designs of operator units, storage units and the checker for RCFile filter units are the same as those for TextFile filter units. All the filtered data are stored in Result FIFO.

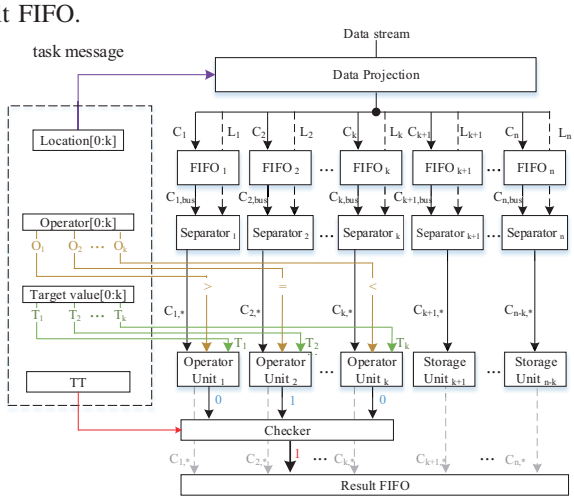


Fig. 9: Processing of RCFile

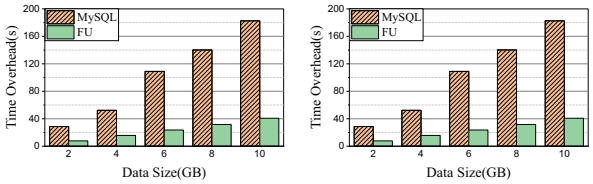
V. Experiment Evaluation

We conducted extensive experiments to evaluate the efficiency of our NIC-QF design. We used a Ubuntu desktop machine with Intel i5 CPU at 3.2 GHz to simulate a storage node. Our NIC-QF is implemented on XILINX XC7K325TFFG900I FPGA board with 326080 Logic Cells, 407600 FF and 16020Kb BRAM, which supports PCIe 2.0 *4 interface and 2G DDR2.

For evaluation, we used TPC-H [14] benchmark whose table has 16 columns and the log data set from Tencent company whose table has 49 columns to test our design. We conducted 6 SQL queries to evaluate our design as representative examples, shown in TABLE.I. The Q1, Q2 and Q3 queries are for TPC-H benchmark and Q4, Q5 and Q6 queries are aimed at Tencent data set, which request nearly 10%, 50% and 90% of test data as result sets respectively.

A. Filtering Performance Comparison

In this experiment, we deploy one Filter Unit(FU) on our NIC-QF to evaluate the filtering performance by using Q2 and Q5 on TPC-H benchmark and Tencent data set in TextFile format and RCFile format respectively. For the filtering performance comparison in TextFile storage format, we chose MySQL [15] as the candidate. And we chose Hive [16] and Presto [17] as the candidates for the comparison of RCFile storage format.



(a) TPC-H benchmark (b) Tencent data set

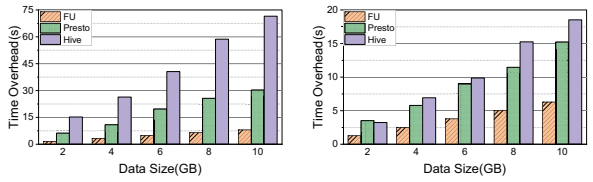
Fig. 10: Filtering performance comparison in TextFile format

1) TextFile: We test the performance of FU on the NIC-QF for TextFile format. Time overheads of our design are shown in Fig.10, using TPC-H benchmark and Tencent data set respectively. The time overhead of our approach includes PCIe transmission and data filtering whereas that of the MySQL is only data filtering. According to Fig.10(a) using TPC-H benchmark, for the case of 2 GB data, the time overhead is 41.49% that of MySQL. But for the case of 10 GB data, the time overhead is only 48.39% of the MySQL. Our approach can reduce 46.23% time overhead of data transmission on average. For the experimental results of Tencent data set shown in Fig.10(b), the minimum time overhead is 21.62% that of MySQL and the maximum time overhead is only 29.99% of MySQL. Our approach can reduce 75.25% time overhead of data transmission on average. According to the observations, we can conclude that the FU on our NIC-QF can effectively reduce the time overheads of filtering data in TextFile format.

2) RCFile: We make efforts to evaluate the performance of FU for RCFile format. Time overheads of three approaches are shown in Fig.11, using TPC-H benchmark and Tencent data set. According to Fig.11(a), we can see that the time overheads are linearly to the amount of data for TPC-H benchmark. For the case of 6 GB data, the time overhead is 24.53% of Presto and 11.91% of Hive. Our approach can reduce 73.61% and 88.59% time overhead on average for Presto and Hive respectively. The experimental results of Tencent data set shown in Fig.11(b). For the case of 6 GB data, the time overhead is 42.03% of the Presto and is 38.24% of Hive. Compared with Presto and Hive, our approach can reduce 58.69% and 63.92% time overhead of data filtering respectively on average. Although the transmission between CPU and FPGA results in certain time overheads, the results suggest that our design can effectively reduce the time overheads compared with Presto and Hive.

B. Impact of Filter Unit Number

We analyzed the resource occupancies of our NIC-QF design with different filter units for different formats on the FPGA board and how many filter units can be optimal. From TABLE.II, besides the resources of the PCIe and NIC communication modules on board, there are still many remaining resources, and the resource



(a) TPC-H benchmark (b) Tencent data set

Fig. 11: Filtering performance comparison in RCFile format

TABLE II: Resource occupancy of each module on FPGA

RESOURCE	LUT	FF	BRAM	BUFG	IO
Total on board	203800	407600	445	32	500
PCIe core	20313	22870	59	6	5
NIC Communication	3215	3397	6	4	23
TextFile	470	262	15	1	-
RCFile	639	840	4	1	-

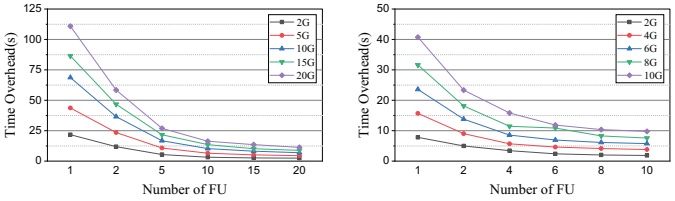
usage of each filter unit is very small whatever RCFile or TextFile. We also clearly find that the number of filter units is actually restricted by the on-board resource BRAM. In order to maximize resource utilization, we conduct a series of experiments at different data size(i.e. 2 GB, 4 GB, 6 GB, 8 GB and 10 GB) to analyze the relationship between the filtering performance and resource usage, shown in Fig. 12. Since the filtering speed of one TextFile filter unit is much lower than the transmission speed of PCIe, it is possible to increase the efficiency by adding the number of filter units. However, the filtering speed of the RCFile filter unit is much higher than that of PCIe, so one filter unit has already reached the maximum speed.

As shown in the Fig.12, time overheads decrease with the increasing number of filter units and tend to be stable at a certain amount of filter units on both TPC-H benchmark or Tencent data set. The number of filter unit is determined by the speeds of single filter unit and PCIe. From the observations, we find that the filtering speed of the TPC-H benchmark is 0.096 GB/s, and that of Tencent data set is 0.297 GB/s, while the transmission speed of PCIe 2.0 is nealy 1.7 GB/s. It can be seen from Fig.12(a) and Fig.12(b) that data filtering of TPC-H benchmark basically reaches the bottleneck when the number of filter units reaches 10, whereas the bottleneck occurs when the number is 6 for Tencent data set. Therefore, we can conclude that more filter units can help to accelerate the filtering. If a faster PCIe interface is used, more filtering units can be added to get better performance.

C. Overall Performance Comparison

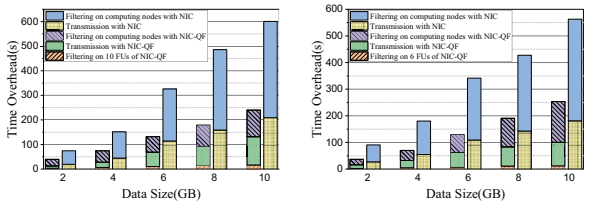
In this section, we conduct a group experiments to evaluate the overall performance of the proposed design. We conduct Hadoop Distributed File System (HDFS) [18] with version of 2.7.4 as a storage node and Hive as a computing node. For comparison, we take the traditional NIC as the candidate. In our design, the overall time overhead consists of the filtering time of FU on our NIC-QF (i.e. *WHERE* clause), the network transmission time of the filtered data, and the filtering execution time of the subsequent complex queries(i.e. *GROUP BY* clause) on computing nodes. While the time overhead of original method includes the network transmission time of original data and the filtering execution time of original data(i.e. complete SQL) on the computing nodes. We test the impact of different data sizes on the overall performance in both TextFile and RCFile formats.

1) TextFile: We deploy 10 filter units for TPC-H benchmark in TestFile format to test overall efficiency using Q2 and Q5 in TABLE.I, shown in Fig.13. Meantime, the overall performances are compared using 6 different SQLs of TABLE.I in Fig.14. For



(a) TPC-H benchmark (b) Tencent data set

Fig. 12: Performance comparison under different Number of FU



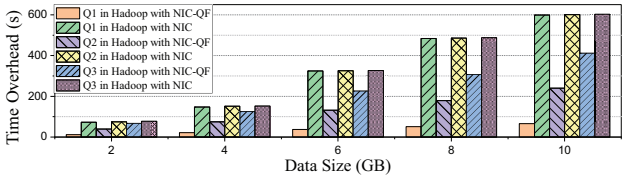
(a) TPC-H benchmark (b) Tencent data set

Fig. 13: Overall time overhead comparison in TextFile format

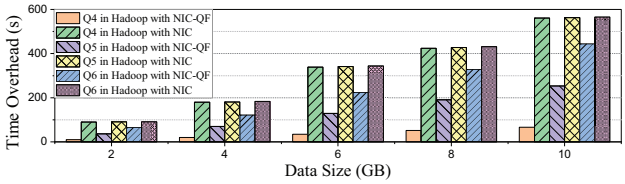
TPC-H data set shown in Fig.13(a), in the case of 2 GB data, the overall performance of our design is 53.01% of that of Hadoop with traditional NIC, while the overall performance of our design has only 39.94% of that of Hadoop with traditional NIC for the case of 10 GB data. Specifically, filtering time on NIC-QF accounts for averagely 7.27% of the total time overheads within our design. Comparing with Hadoop with traditional NIC, our design can greatly reduce the transmission time and computing time in computing nodes. For Tencent benchmark in TestFile format shown in Fig.13(b), we deploy 6 filter units on NIC-QF. We observe that filtering time on computing nodes takes the most overheads on the original Hadoop. In our design the FPGA filter only accounts for 5.78% of the total time overheads, which helps reduce filtering time on computing nodes. The minimum overall time overhead is 37.68% of that of Hadoop with traditional NIC. The maximum overall time overhead is only 45.03% of that of Hadoop with traditional NIC. Our approach can reduce 58.76% overall time overhead on average. Therefore, our design can efficiently accelerate the speed of TextFile data query processing of Big Data systems.

From Fig.14, the overall performances of the 6 SQLs in TABLE I are almost the same in original Hadoop with traditional NIC whereas the overall performances of the 6 SQLs are improved by our NIC-QF. For TPC-H data set using Q1, Q2 and Q3, our approach can reduce 87.30%, 56.12% and 26.17% overall time overhead on average, shown in Fig.14(a). For Tencent data set using Q4, Q5 and Q6 in Fig.14(b), the overall time overheads of our design are 11.27%, 41.24% and 72.64% of that of original Hadoop with traditional NIC, respectively.

2) RCFile: We only deploy 1 filter unit on FPGA board using Q2 and Q5 firstly. Network transmission accounts for the most time overheads in original Hadoop, as shown in Fig.15. Numerical results on TPC-H benchmark are shown in Fig.15(a). Specifically, in the case of 2 GB and 10 GB data, the overall performances of our design are 59.54% and 59.68% of that of Hadoop with traditional

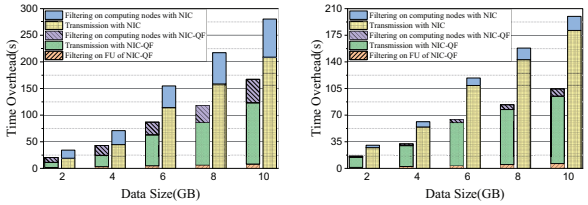


(a) TPC-H benchmark



(b) Tencent data set

Fig. 14: Overall time overhead comparison in TextFile format with different SQL queries

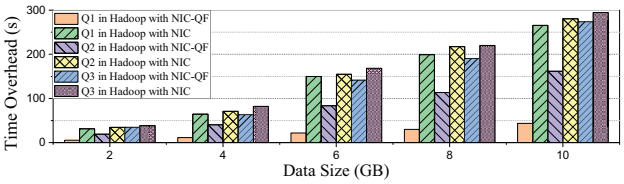


(a) TPC-H benchmark (b) Tencent data set

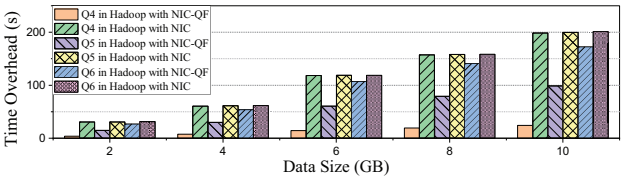
Fig. 15: Overall time overhead comparison in RCFile format

NIC. We can observe that the improvements of our approach can reduce 41.96% of that of traditional NIC on average. The results on Tencent benchmark are shown in Fig.15(b). The minimum overall time overhead is 57.13% of that of Hadoop with traditional NIC. The maximum time overhead is only 58.37% of that of Hadoop with traditional NIC. Our approach can reduce 42.28% overall time overhead on average. According to the observations, we can conclude our NIC-QF can accelerate the data filtering in RCFile format of Big Data systems.

We also use 6 different SQLs to test overall performance in Fig.16. For TPC-H data set using Q1, Q2 and Q3 in Fig.16(a), the overall time overheads of our design are 16.34%, 55.39% and 86.39% of that of original Hadoop with traditional NIC, respectively. For Tencent data set using Q4, Q5 and Q6, our approach can reduce 87.89%, 50.45% and 22.58% overall time overheads on average, shown in Fig.16(b). Therefore, the higher the proportion of filtered data is, the better performance improvements are obtained by our design.



(a) TPC-H benchmark



(b) Tencent data set

Fig. 16: Overall time overhead comparison in RCFile format with different SQL queries

Based on these experiments, we conclude that the design of our NIC-QF can efficiently reduce communication overhead and filtering workload of computing nodes to accelerate data query processing in Big Data systems.

VI. Summary and Conclusions

In this paper, we have proposed an FPGA-based NIC-QF design on storage nodes to improve the performance of storage and computing separated big data systems. The main idea is to add a query filter in the traditional NIC, which can lower the communication overheads and the workloads of computing nodes. The query filter can support the general SQL queries, inside which several filter units perform multiple SQL tasks in parallel. Two filter unit schemes are implemented to support TextFile and RCFile storage formats. Based on TPC-H benchmark and Tencent data set, we evaluated our design by extensive experiments. The experimental results demonstrate the efficiency of our approach, which can significant beat the traditional method. At present, our experiments are based on PCIe 2.0 transmission. If using PCIe 3.0 transmission, the performance will be further improved. Our approach is more suitable for filtering a large amount of data. For future work, we will support more filter unit schemes for different storage formats, besides TextFile and RCFile.

Acknowledgements

This work was partly supported by the Research Fund of National Key Laboratory of Computer Architecture under Grant No.CARCH201811, the Fund of Science and Technology Department of Sichuan Province under Grant No. 2018CC0136, and the Fundamental Research Funds for the Central Universities of China under Grant No. ZYGX2018J077.

References

[1] "Tencent:cloud file storage." [Online]. Available: https://cloud.tencent.com/product/cfs
[2] IBM, "Ibm netezza data warehouse appliances," 2014. [Online]. Available: http://www01.ibm.com/software/data/netezza/.
[3] Y. He, R. Lee, Y. Huai, Z. Shao, N. Jain, X. Zhang, and Z. Xu, "Rcfile: A fast and space-efficient data placement structure in mapreduce-based warehouse systems," 2011.
[4] D. Lomet, "Microsoft sql server's integrated database approach for modern applications and hardware," *Proc. VLDB Endow.*, vol. 6, no. 11, pp. 1178–1179, Aug. 2013. [Online]. Available: http://dx.doi.org/10.14778/2536222.2536248
[5] L. Woods, Z. István, and G. Alonso, "Ibex: An intelligent storage engine with support for advanced sql offloading," *Proc. VLDB Endow.*, vol. 7, no. 11, pp. 963–974, Jul. 2014. [Online]. Available: http://dx.doi.org/10.14778/2732967.2732972
[6] D. Sidler, Z. István, M. Owaida, and G. Alonso, "Accelerating pattern matching queries in hybrid cpu-fpga architectures," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: ACM, 2017, pp. 403–415. [Online]. Available: http://doi.acm.org/10.1145/3035918.3035954
[7] D. Sidler, Z. Istvan, M. Owaida, K. Kara, and G. Alonso, "doppiodb: A hardware accelerated database," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: ACM, 2017, pp. 1659–1662. [Online]. Available: http://doi.acm.org/10.1145/3035918.3058746
[8] A. Caulfield, E. Chung, A. Putnam, H. Angepat, J. Fowers, S. Heil, J.-Y. Kim, D. Lo, M. Papamichael, and T. Massengill, "A cloud-scale acceleration architecture," in *Ieee/acm International Symposium on Microarchitecture*, 2016, pp. 1–1.
[9] S. Werner, D. Heinrich, M. Stelzner, S. Groppe, R. Backasch, and T. Pionteck, "Parallel and pipelined filter operator for hardware-accelerated operator graphs in semantic web databases," in *Proceedings of the 2014 IEEE International Conference on Computer and Information Technology*, ser. CIT '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 539–546. [Online]. Available: https://doi.org/10.1109/CIT.2014.162
[10] J. Teubner, L. Woods, and C. Nie, "Skeleton automata for fpgas: Reconfiguring without reconstructing," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '12. New York, NY, USA: ACM, 2012, pp. 229–240. [Online]. Available: http://doi.acm.org/10.1145/2213836.2213863
[11] D. Ziener, F. Bauer, A. Becher, C. Dennl, K. Meyer-Wegener, U. Schürfeld, J. Teich, J.-S. Vogt, and H. Weber, "Fpga-based dynamically reconfigurable sql query processing," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 9, no. 4, pp. 25:1–25:24, Aug. 2016. [Online]. Available: http://doi.acm.org/10.1145/2845087
[12] D. Koch, J. Tørresen, C. Beckhoff, D. Ziener, C. Dennl, V. Breuer, J. Teich, M. Feilen, and W. Stechele, "Partial reconfiguration on fpgas in practice ł tools and applications," *ARCS 2012*, pp. 1–12, 2012.
[13] C. Dennl, D. Ziener, and J. Teich, "On-the-fly composition of fpga-based sql query accelerators using a partially reconfigurable module library," in *Proceedings of the 2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, ser. FCCM '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 45–52. [Online]. Available: https://doi.org/10.1109/FCCM.2012.18
[14] "Tpc-h:benchmark." [Online]. Available: http://www.tpc.org/tpc_documents_current_versions
[15] "Oracle mysql." [Online]. Available: https://www.mysql.com/
[16] "Apache hive." [Online]. Available: https://hive.apache.org/
[17] "Facebook presto." [Online]. Available: https://prestodb.github.io
[18] "Apache hadoop." [Online]. Available: http://hadoop.apache.org