nativeNDP: Processing Big Data Analytics on Native Storage Nodes

Tobias Vinçon[1], Sergey Hardock[1,2], Christian Riegger[1], Andreas Koch[3], and Ilia Petrov[1]

[1] Data Management Lab, Reutlingen University, Germany
{firstname,surname}@reutlingen-university.de
[2] Databases and Distributed Systems Group, TU Darmstadt, Germany
{firstname,surname}@dvs.tu-darmstadt.de
[3] Embedded Systems and Applications Group, TU Darmstadt, Germany
{firstname,surname}@esa.informatik.tu-darmstadt.de

Abstract. Data analytics tasks on large datasets are computationally-intensive and often demand the compute power of cluster environments. Yet, data cleansing, preparation, dataset characterization and statistics or metrics computation steps are frequent. These are mostly performed ad hoc, in an explorative manner and mandate low response times. But, such steps are I/O intensive and typically very slow due to low data locality, inadequate interfaces and abstractions along the stack. These typically result in prohibitively expensive scans of the full dataset and transformations on interface boundaries.
In this paper we examine *R* as analytical tool, managing large persistent datasets in *Ceph*, a wide-spread cluster file-system. We propose *nativeNDP* – a framework for *Near-Data Processing* that pushes down primitive *R* tasks and executes them in-situ, directly within the storage device of a cluster-node. Across a range of data sizes, we show that *nativeNDP* is more than an order of magnitude faster than other pushdown alternatives.

Keywords: Near-Data Processing, In-Storage Processing, Cluster, Native Storage

1 Introduction

Modern datasets are large, with near-linear growth, driven by developments in IoT, social media, cloud or mobile platforms. Analytical operations and ML workloads result therefore in massive and sometimes repetitive scans of the entire dataset. Furthermore, data preparation and cleansing cause expensive transformations, due to varying abstractions along the analytical stack. For example, our experiments show that computing a simple *sum* on a scientific dataset in *R* takes 1% of the total time, while the remaining 99% are spent for I/O and *csv* format conversion.

Such data transfers, shuffling data across the memory hierarchy, have a negative impact on performance and scalability, and incur low resource efficiency and high energy consumption. The root cause for this phenomenon lies in the typically low data locality as well as in traditional system architectures and algorithms, designed according to the *data-to-code* principle. It requires data to be transferred to the computing units to be processed, which is inherently bounded by the *von Neumann bottleneck*. The negative impact is amplified by the slowdown of *Moore's Law* and the end of *Dennard Scaling*. The limited performance and scalability is especially painful for nodes of high-performance cluster environments with sufficient processing power to support computationally-intensive analytics.

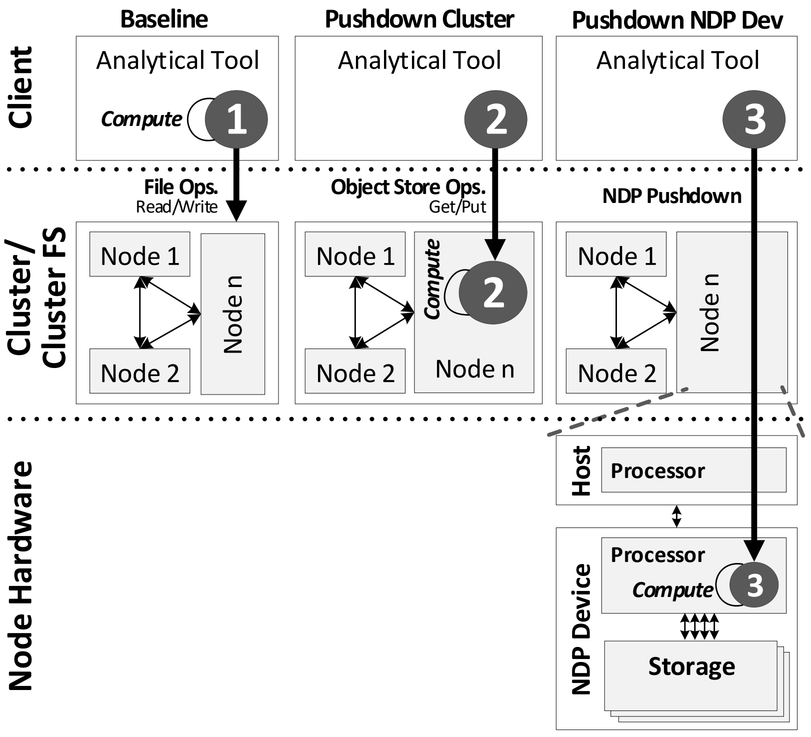

Fig. 1: Three different options to execute analytical operations on a cluster environment. (1) Baseline: Execute on the client; (2) Pushdown Cluster: Execute on a cluster's node; (3) Pushdown NDP Device: Execute on the NDP Device of a cluster's node

Luckily, recent technological developments help to counter these drawbacks. Firstly, hardware vendors can *fabricate combinations of storage and compute elements at reasonable costs*. Secondly, this trend covers virtually all levels of the memory hierarchy (e.g. IBM's AMC for Processing-in-Memory, or Micron's HMC). Thirdly, the device-internal bandwidth and parallelism significantly exceed the external ones (Device-To-Host), for non-volatile semiconductor (NVM, Flash) storage devices.

Such *intelligent storage* allows for *Near Data Processing (NDP)* of analytics operations, i.e. such operations are executed in-situ, close to where data is physically stored and transfer just the result sets, without moving the raw data. This results in a *code-to-data* architecture.

Analytical operations are diverse and range from complex algorithms to basic mathematical, statistical or algebraic operations. In this paper we present execution options for basic operations in nodes of clustered environments as shown in Figure 1: (1) The computation is within the client and the cluster node is used as part of a traditional distributed file system; (2) The operation is transmitted to the cluster and processed within the cluster node itself; (3) The operation is executed in-situ, within the NDP devices of the cluster's node. The investigated operations are simple, yet they clearly give evidence for the NDP effects on internal bandwidth and the ease of system and network buses. The execution of more extensive operations like betweenness centrality within graphs or clustering and k-nearest neighbour searches are planed for future work.

The main contributions of this paper are:
- End-to-end integration of NDP interfaces throughout the entire system stack
- The performance evaluation shows improvements of NDP operation push-down of at least 10x
- Analysis of the impact of and necessity for NDP-based abstractions and interfaces.
- We identify the following aspects as the main drawbacks to implementing NDP: Interfaces; Abstractions; ResultSet consumption semantics; DataLayout and NDP toolchain

The rest of the paper is structured as follows. Section 3 presents the architecture of *nativeNDP*. In Section 4 we discuss the experimental design and performance evaluation. We conclude in Section 5.

2 Related Work

The concept of *Near-Data Processing* is not new. Historically it is deeply rooted in *database machines* [6, 3], developed in the 1970 and 1980s. [3] discuss approaches such as processor-per-track or processor-per-head as an early attempt to combine magneto-mechanical storage and simple computing elements to process data directly on mass storage and to reduce data transfers. Besides reliance on proprietary and costly hardware, the I/O bandwidth and parallelism are claimed to be the limiting factor to justify parallel DBMS [3]. While this conclusion is not surprising, given the characteristics of magnetic/mechanical storage combined with Amdahl's balanced systems law [8], it is revised with modern technologies. Modern semi-conductor storage technologies (NVM, Flash) are offering high raw bandwidth and high levels of parallelism. [3] also raises the issue of temporal locality in database applications, which has already been questioned earlier and is considered to be low in modern workloads, causing unnecessary data transfers. Near-Data Processing presents an opportunity to address it.

The concept of *Active Disk* emerged towards the end of the 1990s. It is most prominently represented by systems such as: Active Disk [2], IDISK [12], and Active storage/disk [15]. While database machines attempted to execute fixed primitive access operations, *Active Disk* targets executing application-specific code on the drive. Active storage [15] relies on processor-per-disk architecture. It yields significant performance benefits for I/O bound scans in terms of bandwidth, parallelism and reduction of data transfers. IDISK [12], assume a higher complexity of data processing operations compared to [15] and targets mainly analytical workloads and business intelligence and DSS systems. Active Disc [2] targets an architecture based on on-device processors and pushdown of custom data-processing operations. [2] focuses on programming models and explores a streaming-based programming model, expressing data intensive operations, as so called *disklets*, which are pushed down and executed on the disk processor.

With the latest trend of applying different compute units, besides CPUs, to accelerate database workloads, a more intelligent FPGA-based storage engine for databases has been demonstrated with Ibex [19]. It focuses mainly on the implementation of classical database operations on reprogrammable compute units to satisfy their characteristics, such as parallelism and bandwidth. A completely distributed storage layer, targeting NDP on DRAM over the network, is presented by Caribou [11]. Its shared-data model is replicated from the master to the respective replica nodes using Zookeeper's atomic broadcast. Utilising bitmaps, Caribou is able to scan datasets with FPGAs only by the limiting factor of the selection itself (low selectivity) or the network (high selectivity). Moreover, [5, 4, 14, 9] investigate further host-to-device interfaces for general-purpose applications or specific workloads.

However, previous research focused mainly either on the concrete implementation of the reconfigurable hardware, or on single device instances. In this paper, we attempt to combine both topics and focus on the abstraction and interfaces necessary to complete an efficient NDP pushdown.

3 nativeNDP Framework

The architecture shown in Figure 2 presents a bird's eye view of the essential components, interfaces, and abstractions of the nativeNDP framework. An analytical client executes an R script, triggering an analytical operation (filtering, simple computation - SUM, AVG, STDDEV, or a clustering algorithm). It can be processing can be processed on different levels of the system stack:

– directly in R (Figure 1–*baseline*). This is a classical approach, which can be done with out-of-the-box software, requiring little overhead. The downside is that the complete dataset needs to be transferred through the stack causing excessive data transfers and posing significant memory pressure on the client.

- within a *Cluster node* (Figure 1–*pushdwon cluster*). The same function can be offloaded to the HPC cluster system and distributed across nodes. Hence the compute and data transfer load can be reduced, but not eliminated as such data transfers are preformed locally on a node.
- on the *Storage Device* (Figure 1–*pushdown NDP dev*). With NDP the operations are offloaded directly on the device, utilising the internal bandwidth, parallelism and compute resources to reduce data transfers and improve latency.

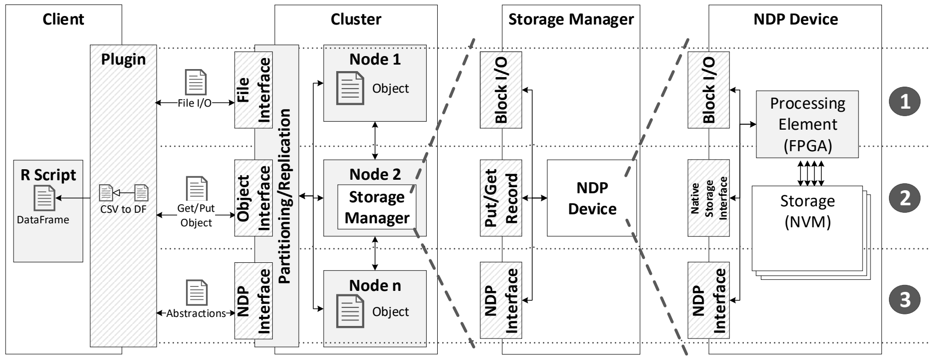

Fig. 2: The high-level architecture showing the applied interfaces and data abstractions along the access path for the three compared experiments: baseline, pushdown cluster node, and pushdown device

3.1 System Stack

In the following we describe the layers of the analytical stack in more detail.

Client: We utilise R as one of the most popular client software for analytical and statistical computation. To interact with the Ceph cluster and the underlaying layers, we designed a custom R plugin, RCeph. It uses the RADOS API [18] to connect against the cluster and is able to issue specific commands with following features:

Put/Get of Files/Objects: To facilitate the first scenario, presented in figure 1, the dataset file has to be retrievable from the cluster. Therefore, the standard file I/O API is reused. However, the transfer of results from the second and third scenario necessitates further interfaces such as RADOS's provided Object API as explained in Section 3.2.

Pushdown of Domain-specific Operations: This feature is mainly addressed with the second and third scenario, where domain-specific operations, usually executed within the client, are push down to either a cluster's node or even throughout the node's storage engine to the NDP Device. I.e. such domain-specific operations comprise R-native operations on their storage abstraction *DataFrame* or could even be extended to small algorithmic expressions.

Format Conversion: As interfaces and abstractions of lower levels often rely on backwards-compatibility in nowadays complex systems, format conversions of the results or CSV-encoded files and objects into the R-specific abstraction *DataFrame* are necessary.

The RCeph is complied using Rcpp [7] to a plugin package and can be installed, loaded, and applied within the R runtime environment of the client.

Cluster: To process todays datasets with analytical or statistical workloads in an acceptable time, both data and calculation are distributed over a cluster environment. This becomes even more crucial with focus on high performance in particular. To simplify low latency data accesses distributed file systems are applied in such environments nowadays. Therefore, Ceph [17], which is a wide-spread solution for clustered environments, builds the foundation of the nativeNDP framework. Its purpose is to efficiently manage a variety of nodes within a cluster environment. Thereby, stored files are striped across small objects, grouped into placement groups and distributed on these nodes to ensure scalability and high reliability. Its flexible architecture comprises various components and provides interfaces for object, block and file I/O. Internally, exchangeable storage engines are responsible to manage the read and writes to secondary storage. One of its most recent storage backends is called BlueStore and utilises RocksDB as an internal KV-Store.

Storage Manager: We replaced the internal KV-Store of BlueStore with our own native storage engine NoFTL-KV [16]. Hereby, hardware characteristics, like in-parallel accessible flash chips of the storage device, are known by NoFTL-KV, which in turn is able to efficiently leverage those. Consequently, the physical location of persisted data is defined by the KV-Store itself rather than any Flash-Translation-Layer (FTL) of a conventional stack. This opens the opportunity to issue commands directly on the physical locations throughout NoFTL-KV and to streamline low-level interfaces along the entire access path.

NDP Device: Devices are emulated by our own storage-type SCM Simulator, based on [10]. Running as a kernel module it provides the ability to delay read and write request depending on its emulated physical locations by utilising the accurate kernel timer functions. As a consequence, reads or writes across physical page borders claim respectively multiple I/O latencies. In the experimental evaluation the simulator is instrumented with realistic storage-type SCM latencies from [1]. Moreover, by its flexible design it allows us to extend it with the necessary NDP interface.

3.2 Inferfaces and Abstractions

The first, most commonly applied interface is the traditional file I/O (Figure 2.1). It abstracts the cluster as a large file system, storing its data distributed on multiple nodes. A partitioning and/or replication layer takes care of the internal data placement on various nodes. Instead of the KV-Store the conventional Block I/O is used to issue reads and writes to the NDP Device. This also involves any kind of Flash-Translation-Layer on the device itself to reduce the wear on a single storage cell and consequently ensure longevity of the entire device.

Secondly, a modern object interface offered by RADOS [18] (Figure 2.2) can be utilized to put/get objects on the cluster. This abstraction might comprise single or multiple records of a file, or the result set of a pushed down user defined function executed on the respective node. Since the cluster handles data placement, it can transparently execute such algorithms in parallel with the full processing power of the node's servers if the operations are data independent. Within the lower levels, depending on the storage manager, one can either exploit the conventional Block I/O to access the NDP Device or leverage NoFTL-KV's *Native Storage Interface.*

Thirdly, an NDP pushdown necessitates a different kind of interface definition (Figure 2.3). The NDP execution of application-specific operations requires open interfaces. These should support NDP of application-specific abstractions such as *DataFrame* for R. Consequently, these interfaces and abstractions mandate flexibility, since various result types of the application logic on the device must be transferred back to the client. Expensive format conversion along the system stack can be avoided almost entirely. Yet, an extensive toolchain and NDP framework support is required, beginning from the analytical tool to the employed hardware devices in the cluster. Utilising the processing elements near-storage (e.g. FPGA), the internal, on-device parallelism and bandwidth. For instance, [13] projects of up to 50 GB/s, while the workload on slower buses (e.g. PCIe 2.0 $\approx$ 6.4 GB/s) in the system is eased by reducing transfer volumes (i.e. $resultset \ll rawdata$).

4 Experimental Evaluation

To compare the different execution options on the presented system stack and evaluate their bottlenecks, we conduct three experiments aligned to the scenarios of Figure 1.

4.1 Datasets and Operations

To ensure the comparability of the scenarios, datasets and operations are predefined. The datasets are created synthetically as CSV files with random numbers, with varying rows and columns from 1k to 10k. When stored in the KV-Store, each cell of the CSV File is identifiable by an auto-generated key with the structure:

$$[object\_name].[column\_index].[row\_index]$$

Inevitably, this is bloating out the raw file size by approximately 16x-17x but enables to access cells by this unique id. Alternatively, depending on the workload, an arrangement pre row or per column is likewise feasible. Table 1 summarizes the properties of each dataset for the present experiments.

Dataset	**KV Pairs**	**CSV Size** [MB]	**KV Size** [MB]	**Bloating Ratio**
1k/1k	1 000 000	2.8	44	15.9
2k/2k	4 000 000	12	182	15.2
4k/4k	16 000 000	45	738	16.4
6k/6k	36 000 000	101	1 668	16.5
8k/8k	64 000 000	178	2 971	16.7
10k/10k	100 000 000	278	4 649	16.7

Table 1: Synthetically generated datasets for the experiments. The raw CSV file size is according the Key-Value format bloated out.

The operations performed in all experiments is independent of the data distribution and constitutes a typical data science application - calculation of the sum or the average over a given column (Because of the marginal differences only sum is shown further on). The final result set comprises a 32 byte integer value and some additional status data. We leave the implementation of further analytical and/or statistical operations open for future work.

4.2 Experimental Setup

The server, *nativeNDP* is evaluated on, is equipped with four Intel Xeon x7560 8-core CPUs clocked at 2.26 GHz, 1TB DRAM running Debian 4.9, kernel 4.9.0. The NDP storage device is emulated by our real-time NVM Simulator, extended with an NDP interface and functionality. I/O and pushdown operations are handled internally with the storage-type SCM latencies [1].

Since the main target is to evaluate the streamlining of NDP interfaces and abstractions, interferences caused by data distribution or multi-node communication have to be avoided. Therefore, the Ceph cluster is set up with a single object store node. This allows conducting experiments along a clean stack and measuring execution and transfer size for each architectural layer individually.

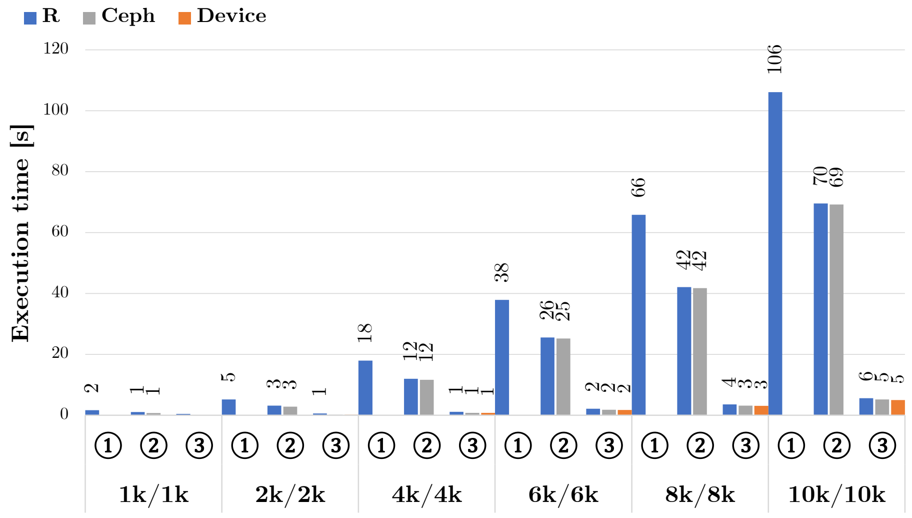

Fig. 3: Execution time for varying dataset sizes shows the performance impact of data transfers/volume, and the improvement through NDP.

4.3 Experiment 1 – Baseline

The first experiment utilises the Ceph cluster in the most common and conventional way - as a file system (Figure 2.1). Therefore, the file abstractions, interfaces, and subsequently Block I/O are used to retrieve the entire file. The sum over the 10th column is calculated in R by calling *readCSVDataFrame* of RCeph and caching the resulting DataFrame into the R runtime environment. Here, R's capabilities can be used to filter the DataFrame on the respective column and perform the arithmetic operation.

```
sum <- sum(RCeph::readCSVDataFrame(o_name)[col_id])
```

This experiment defines the baseline for any improvements of nativeNDP. However, it exemplifies multiple drawbacks yielding in a significant performance degradation. Firstly, the entire file has to be read via block I/O, even though only a small portion of it, the 10th column, is necessary to be processed by the operation (Figure 5). Secondly, the latency and bandwidth limitations of the network interconnect between the R host and the Ceph cluster, contribute to additional delays to the R processing. The significantly higher transfer size of Host-To-Client, illustrated in Figure 5, leads inevitably to a slower request duration. Additionally, as R DataFrames do not support any streaming algorithmic, the processing has to idle until the entire dataset is retrieved from Ceph. Thirdly, additional compute-intensive format conversions along multiple interface boundaries are necessary to create R *DataFrames*, which increase delays even further. E.g. the "R - parse_time" is 95% of the total time as shown in Figure 4. Moreover, such format conversion are directly depending on the data size, which is subsequently affected by the large Host-To-Client transfer size. Lastly, client systems often comprise limited hardware (e.g. notebook or workstation), while typical working sets can range from tens to hundreds of gigabytes. Thus, processing the whole dataset is not always possible without any performance degrading swapping to disk.

These drawbacks lead to a significantly higher total execution time for the calculation in general, as shown in Figure 3 (at least 10x).

In total, the baseline experiment results in the lowest performance for all datasets, which is mainly caused by the time spent in transfer and conversion of the CSV object into the R specific data type *DataFrame* ("R - parse_time" Figure 4).

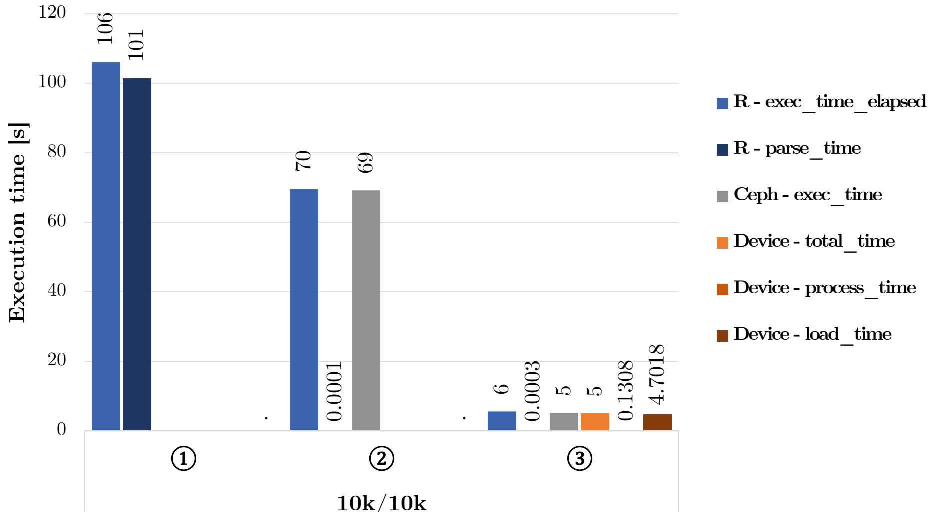

Fig. 4: A detailed execution time analysis shows the main bottlenecks along the analytical stack.

4.4 Experiment 2 – Pushdown Cluster

For the second experiment, Ceph's advanced object interface is extended to execute a user defined function. It queries the KV-Pairs of the respective dataset from NoFTL-KV of the Storage Manager by filtering on the 10th column. Thereby, the retrieved values are cumulated (Figure 2.2). In a full-fledged cluster scenario, Ceph will automatically distribute this algorithm on the respective nodes within the cluster and aggregate their results afterwards. Obviously, the result size after the operation pushdown is dramatically smaller than the raw data, which relieves the network and accelerates subsequent expensive data format conversions. Hence, the almost non-existing "R - parse_time" (Figure 4) and the respective transfer size from *Host-To-Client* (Figure 5). Both result in an overall performance improvement of up to 30% in comparison to the baseline (Figure 3).

```
sum <- RCeph::execCmd(o_name, "NDP_CEPH SELECT SUM COLUMN col_id")
```

Nonetheless, the I/O overhead of reading the entire data from the storage subsystem, as shown in Figure 5 by *Device-To-Host*, represents a major bottleneck. Therefore, the time spent in format conversions within Ceph increases as well. For the largest dataset it takes more than 99% of the time. However, it can be avoided by applying NDP.

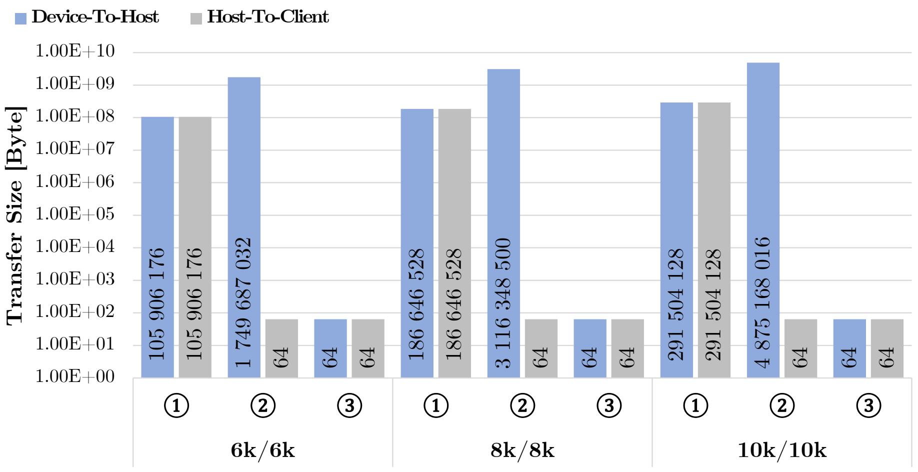

Fig. 5: Transfer sizes from Device-To-Host and Host-To-Client of varying datasets shows the counteraction of NDP to the *von Neumann bottleneck*

4.5 Experiment 3 – Pushdown NDP Device

Our last experiment relies on Near-Data Processing (Figure 2.3). Abstractions and interfaces are statically created for the purpose of filtering on a given column and computing sums to enable a device pushdown.

```
sum <- RCeph::execCmd(obj_name,"NDP_DEV SELECT SUM COLUMN col_id")
```

The NDP pushdown leverages the much higher levels of compute and I/O parallelism supported by the on-device processing elements (FPGA, GPU) to compute the sum an order of magnitude faster (Figure 3). Thereby, transferring data from the storage chips takes most of the time (Figure 4 "Device - load_time"), while the processing is only about 3% of the total time (Figure 4

"Device - process_time"). Not only is the network relieved by this early reduction of volume, but also the system-wide number of data transfers is significantly reduced. This is mainly driven by the on-device computation and result size reduction as shown in Figure 5. As this is only possible with the application-specific abstractions, a push down command must compulsorily comprise those to apply computation on the device, *in-situ*. In R, for instance, *DataFrame* may be a suitable application-specific abstraction.

5 Conclusion

We present *nativeNDP*, a NDP approach to effectively pushdown analytical operations to a native storage node of a clustered environment. The evaluation shows improvements of at least 10x over the baseline. Besides the known issues with todays computer architectures, we identify ill-suited interfaces and abstractions along the analytical stack as major drawbacks of current solutions. Moreover, the necessity to push down application-specific abstractions, and data layouts interpretable by the NDP device is considered a key aspect for a true *in-situ* processing in complex system stacks. To mitigate format conversions along interface boundaries of such stacks, a comprehensive but flexible NDP toolchain is required.

References

1. Itrs - international technology roadmap for semiconductors reports (2014), http://www.itrs2.net/itrs-reports.html
2. Acharya, A., Uysal, M., Saltz, J.H.: Active disks: Programming model, algorithms and evaluation. In: ASPLOS (1998)
3. Boral, H., DeWitt, D.J.: Parallel architectures for database systems. chap. Database Machines: An Idea Whose Time Has Passed? A Critique of the Future of Database Machines (1989)
4. Cho, S., Park, C., Oh, H., Kim, S., Yi, Y., Ganger, G.R.: Active disk meets flash. In: Proc. 27th Int. ACM Conf. Int. Conf. Supercomput. - ICS. p. 91. ACM Press (2013)
5. De, A., Gokhale, M., Gupta, R., Swanson, S.: Minerva: Accelerating Data Analysis in Next-Generation SSDs. In: 2013 IEEE 21st Annu. Int. Symp. Field-Programmable Cust. Comput. Mach. pp. 9–16. IEEE (apr 2013)
6. DeWitt, D., Gray, J.: Parallel database systems: The future of high performance database systems. Commun. ACM
7. Eddelbuettel, D.: Seamless R and C++ integration with Rcpp. Springer (2013)
8. Gray, J., Shenoy, P.J.: Rules of thumb in data engineering. In: Proc. ICDE. p. 3 (2000)
9. Gu, B., Yoon, A.S., Bae, D.H., Jo, I., Lee, J., Yoon, J., Kang, J.U., Kwon, M., Yoon, C., Cho, S., Jeong, J., Chang, D.: Biscuit: A Framework for Near-Data Processing of Big Data Workloads. In: ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. vol. 8, pp. 153–165. IEEE (jun 2016)
10. Hardock, S., Petrov, I., Gottstein, R., Buchmann, A.: Noftl: Database systems on ftl-less flash storage. Proc. VLDB Endow. (2013)

11. István, Z., Sidler, D., Alonso, G.: Caribou. Proc. VLDB Endow. **10**(11), 1202–1213 (aug 2017)
12. Keeton, K., Patterson, D.A., Hellerstein, J.M.: A case for intelligent disks (idisks). SIGMOD Rec. **27**(3), 42–52 (Sep 1998)
13. Kim, S., Oh, H., Park, C., Cho, S., Lee, S.W., Moon, B.: In-storage processing of database scans and joins. Inf. Sci. (Ny). **327**, 183–200 (jan 2016)
14. Minutoli, M., Kuntz, S.K., Tumeo, A., Kogge, P.M.: Implementing Radix Sort on Emu 1. Work. Near-Data Process. pp. 1–6 (2015)
15. Riedel, E., Gibson, G.A., Faloutsos, C.: Active storage for large-scale data mining and multimedia. In: Proceedings of the 24rd International Conference on Very Large Data Bases. pp. 62–73. VLDB, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1998)
16. Vinçon, T., Hardock, S., Riegger, C., Oppermann, J., Koch, A., Petrov, I.: Noftl-kv: Tacklingwrite-amplification on kv-stores with native storage management. In: EDBT (2018)
17. Weil, S.A., Brandt, S.A., Miller, E.L., Long, D.D.E., Maltzahn, C.: Ceph: A scalable, high-performance distributed file system. In: OSDI
18. Weil, S.A., Leung, A.W., Brandt, S.A., Maltzahn, C.: Rados: a scalable, reliable storage service for petabyte-scale storage clusters. In: PDSW (2007)
19. Woods, L., Teubner, J., Alonso, G.: Less watts, more performance. In: Proc. 2013 Int. Conf. Manag. data - SIGMOD. p. 1073. ACM Press, New York, New York, USA (2013)