A cost model for NDP-aware query optimization for KV-stores

Christian Knödler, Tobias Vinçon, Arthur Bernhardt, Ilia Petrov
[firstname].[surname]@reutlingen-university.de
Data Management Lab,
Reutlingen University

Leonardo Solis-Vasquez, Lukas Weber, Andreas Koch
[surname]@esa.tu-darmstadt.de
Embedded Systems and Applications Group,
TU Darmstadt

ABSTRACT

Many modern DBMS architectures require transferring data from storage to process it afterwards. Given the continuously increasing amounts of data, data transfers quickly become a scalability limiting factor. Near-Data Processing and smart/computational storage emerge as promising trends allowing for decoupled in-situ operation execution, data transfer reduction and better bandwidth utilization. However, not every operation is suitable for an in-situ execution and a careful placement and optimization is needed.

In this paper we present an NDP-aware cost model. It has been implemented in MySQL and evaluated with nKV. We make several observations underscoring the need for optimization.

ACM Reference Format:
Christian Knödler, Tobias Vinçon, Arthur Bernhardt, Ilia Petrov and Leonardo Solis-Vasquez, Lukas Weber, Andreas Koch. 2021. A cost model for NDP-aware query optimization for KV-stores. In *International Workshop on Data Management on New Hardware(DAMON'21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3465998.3466013

1 INTRODUCTION

Regardless of the increasing data-sizes and the evolution of storage technology, modern DBMS employ traditional *data-to-code* architectures, which require growing amounts of data to be transferred to the DBMS host. This quickly saturates the available I/O bandwidth and ultimately limits the scalability. Currently, *computational storage* emerges as a trend. *Combined compute and semiconductor storage* elements can be manufactured economically and packaged in the same storage device. As a result, the *device-internal* bandwidth, parallelism, and latencies are much better than the external ones (device-to-host). This lifts major limitations to prior work e.g. *database machines* [4] or *active disks* [14]. Furthermore, it enables Near-Data Processing (NDP) architectures and *code-to-data* paradigms that execute DB-operations close to where data is physically stored.

Background. Although, the concept of NDP and in-storage processing is deeply rooted in [4] or [14], it has recently reemerged due to the advances in the semiconductor industry. Lately, numerous systems [1–3, 7, 9, 10, 12, 15, 16] were proposed and utilize on-device processing capabilities to speedup database operations like selections and joins by avoiding costly data transfers between host and device but *resort to in-situ executions.* In contrast, we observe that NDP executions are not always best and therefore there is a need for execution placement and optimization.

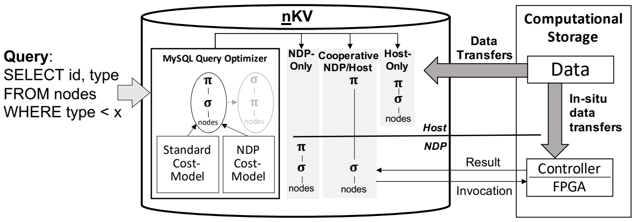


Figure 1: Optimization strategy for computational storage.

Thus choosing the right optimization strategy gains importance. Making the optimizer aware of NDP, yields more possibilities for query plan alternatives and increases the complexity of finding the optimal plan. Current optimization strategies do not consider the characteristics of the underlying hardware resulting in choosing possible non-optimal query plan. [8] is closest to our work however it does not address KV-Stores.

Motivation. With computational storage many DBMS operations like scans, selections, projections or joins, to name a few, have a host-based and a NDP-implementation. This brings up the question of when to select which one. Clearly, existing DBMS query optimizers can be leveraged if they are extended with an *NDP-aware cost model.* The core intuition is that the DBMS optimizer can decide what plans or even parts of a plan can be executed in-situ, and which ones on the host (Figure 1). We also envision cooperative plans, where parts of the query plan are executed on-device, while other parts consume their results and are processed on the host. This becomes an interesting problem considering the different on-device characteristics in terms of I/O, compute, and transfer properties. For example: (i) full scans and version visibility checks are faster in-situ given the higher on-device bandwidth and the lower latencies; in addition (ii) selections (coupled to scans) benefit from on-device compute elements and possibly from scalable HW-SW co-design implementations; (iii) results (intermediary or final) can be temporarily (fully or partially) materialized on-device to be transferred efficiently with a few DMA-transfers back to DBMS/host.

In this paper we present an initial NDP-aware cost model. It has been implemented as an extension of the MySQL optimizer and based on the cost-models it decides whether a purely host-based or an NDP-only execution should be preferred. We use nKV [15] as the underlying NDP-engine, which is a Key/Value-store based on

DAMON'21, June 20–25, 2021, Virtual Event,China

ACM ISBN 978-1-4503-8556-5/21/06…$15.00
https://doi.org/10.1145/3465998.3466013

RocksDB [6] that has been integrated into MySQL with MyRocks [11]. Our *contributions* are: (i) we extend MySQL with a NDP-aware cost-model allowing the optimizer to distinguish between NDP-only and host-only executions. (ii) it shows the value of *early selection and projections* as well as *NPD result handling*. (iii) it is implemented in nKV [15] and tested on real computational storage hardware: COSMOS+ NVMe SSD [13].

2 NDP-AWARE COST MODEL

Cost-based optimization is a classical technique, where the query optimizer enumerates possible query plans, estimating their execution costs according to a cost-model and an abstract hardware model. Ideally, it selects the plan with the lowest total cost. In MySQL/MyRocks we introduce a new *NDP cost-model* and costs of each plan are estimated both by the standard and the NDP cost-model. Furthermore, we emphasize typical heuristics such as *early selection* or *early projection* because of their NDP relevance. While both reduce data transfers and sizes, the former yields better utilization of the on-device I/O properties and thus higher selectivity tolerance, whereas the latter reduces the result sizes and yields efficient transfers. *Early selection* (or selection push-down) is an optimization heuristic, aiming to evaluate selection predicates as early as possible (ideally as part of a scan) and possibly swapping the order of joins and selections. In terms of NDP it effectively reduces the result-set size through in-situ tuple-filtering. In contrast, *early projection* is an optimization heuristic aiming to perform projections first to reduce the size of the resultant records. Like early selection it is not always possible, for instance due to commutativity constraints. In NDP settings, early projections enables in-situ attribute-filtering, which decreases the result-set sizes even further and minimizes expensive `memcpy` operations on device.

Moreover, we focus on NDP-operations on the *value*, as *key*-operations are supported by the LSM storage organization, and by auxiliary structures, i.e. *fence pointers* or *Bloom-filters*. Fence pointers are a kind of Min/Max key index per SST and block that can support range filtering and estimation. Bloom-filters, typically created on the key and for each SST, efficiently support point-filtering.

The total execution time for a value-scan is intuitively expressed as: $c_{total} = c_{scan} + c_{cpu} + c_{transfer}$, where $c_{transfer}$ is the cost for transferring the data from device to the host; c_{scan} is the (engine-specific) cost of scanning the table; and c_{cpu} accounts for the CPU processing costs. We define the individual terms in two different cost models: (a) *host-only execution* – equations (2), (3) and (4); and (b) *NDP-execution* – equations (5), (6) and (7).

$$c_{total} = c_{scan} + c_{cpu} + c_{transfer} \tag{1}$$

$$c_{transfer\_host} = \frac{(n_{records} \cdot MRL)}{BS} \cdot BTT \tag{2}$$

$$c_{scan\_host} = RDBST \tag{3}$$

$$c_{cpu\_host} = n_{records} \cdot REC \tag{4}$$

$$c_{transfer\_NDP} = \frac{(SEL \cdot n_{records} \cdot PB)}{BS} \cdot BTT \tag{5}$$

$$c_{scan\_NDP} = RDBST \cdot DEV_{type} \tag{6}$$

$$c_{cpu\_NDP} = \frac{(n_{records} \cdot REC_{NDP} \cdot PB)}{MRL} \tag{7}$$

Table 1: Flash Latencies and Bandwidth (BW) of the *COSMOS+ OpenSSD* for different levels of parallelism.

		Read Latency per 4 KB [μs]		
	Host	486		
	Device	200 - 400 (depending on LUNs and Channels)		
Access	**Pages**	**Parallelism**	**BW [MB/s]**	**IOPS**
Random	1500	1 Ch. 1 LUN	52	3000
	1500	2 Ch. 1 LUN	102	6000
	1500	1 Ch. 8 LUN	108	6000
	1500	*2 Ch. 8 LUN*	*213*	*13000*
Sequential	*640*	*2 Ch. 8 LUN*	*217*	*13000*

Transfer Costs. The transfer time is expressed by equations (2) and (5) for both models, as the number of blocks multiplied by the transfer latency. Some important differences result from selectivity estimation and from handing *early projections*. We leave the in-situ support for the selectivity and cardinality estimation for future work. Early projections must be preferably handled in-situ, as they are important means for reducing the size of the projected records *PB*, and thus the result set size of NDP executions. Since host executions need to transfer the raw data from device to host, they can still benefit from early projections, however in terms of processing cost, not in terms of reduction of data transfers from storage.

Result-set size estimation is an important factor for estimating the total cost of NDP-executions. For a full scan (without selection predicate), it is estimated in terms of the number of records and the mean-record length (*MRL*). For a key-scan, the estimation is facilitated by fence-pointers to determine the expected number of keys and the *MRL* or the projectivity (i.e. the number of projected attributes). Furthermore, it is possible to pack multiple transfer units together in-situ and transfer them efficiently as a single DMA transfer back to the host. All of these lead to lower transfer costs for the NDP execution, which is accounted by *SEL* (Eq. 5). Alternatively, for host-executions (Eq. 2), we consider the full amount of records for the calculation of the transfer costs, since the current cost model does not differentiate between value- and index-scans.

Scan Costs. During a *full scan* MySQL as well as the NDP-execution need to scan the whole data set. However, c_{scan} costs (Eq. (3) and (6)) may differ significantly depending on the class of computational storage devices. Commodity devices, such as the *COSMOS+* used in this work, exhibit c_{scan_NDP} costs ($DEV_{type} = 1$) equal to the host c_{scan_host}, since the on-device bandwidth is approximately equal to the device-to-host bandwidth. This changes with enterprise devices, which tend to have higher on-device bandwidth and therefore $DEV_{type} = 0.5 \ldots 0.7$ (Eq. 6).

Index scans differ because of the on-device properties and the underlying storage organization. nKV is an LSM-tree based KV-store on top of RocksDB under MyRocks. Hence, the typical auxiliary structures such as fence pointers and Bloom-filters are available and memory resident as they are loaded by MySQL/MyRocks during startup. This significantly reduces the index-scan effort (key predicates) in host-only scenarios.

In contrast, an on device index scan needs to read the index first, however nKV accelerates the process in two stages. Firstly, nKV

Table 2: Expensiveness of memcpy on device compared to host

Size [KB]	memcpy	Device Time [μs]	Host Time [μs]
1	1000	284	20
10	1000	2592	202
1	10000	2831	194
10	10000	25920	1973

also utilizes the fence-pointers in NDP-scenarios as part of the NDP-call preparation, thus reducing the number of SST to be processed on device. Secondly, because SST-organized LSM-trees resemble clustered indices, the respective KV-records (on an SST) can be read directly without any index-to-record I/O overhead. We leave unclustered indices for future work. Overall, the current cost-model does not distinguish between full-scan and clustered index-scan, and consequently calculates the same costs.

Processing/CPU Costs. MySQL uses a fixed cost of $REC = 0.2$ for evaluating a single record (see Eq. (4)). The NDP costs REC_{NDP} vary significantly, depending on the type of computational storage device. On the one hand, newer enterprise devices offer extensive compute performance and parallelism due to higher clock frequencies, larger FPGA area, and optimized heterogeneous processing elements. On the other hand, we still rely on the dual ARM-cores of the *COSMOS+* for NDP. Due to the limited performance of the 667 MHz dual ARM-cores, the end-to-end performance is lower and thus, the cost for evaluating a single record $REC_{NDP} = 0.5$ is higher than REC (Eq. (7)).

This can be demonstrated by comparing the duration of the expensive memcpy on-device against the host as shown in Table 2. Copying 1 KB of data is more than 10× slower on device compared to the host. Therefore, lower projectivities and *early projection* help avoiding unnecessary memcpy-operations on device, and thus improve performance.

3 EVALUATION

The evaluation is performed in MySQL 5.6/MyRocks [11] on a server, equipped with a 3.4 GHz Intel i5, 4GB RAM and Debian 4.9 with ext3. The main computational engine of the *COSMOS+* platform is a Xilinx Zynq-7000 System-on-Chip that combines two ARM Cortex-A9 cores with an FPGA. The *COSMOS+* [13] is exposed as NVMe block device (BLK-baseline), and as an NVMe NDP-device to nKV (NDP-baseline). The block size is 32KB. Its basic I/O characteristics are shown in Table 1.

Our *dataset* is based on the *nodetable* of LinkBench [5], with minor changes to align for the 32-bit *COSMOS+* ARM: *id* field is UINT32, while the *data* field is a fixed size char[144]. The dataset comprises 1M records amounting to 181MB. Due to a hardware limitation in the COSMOS+ DMA-engine, the max. result size of an NDP-call is 1MB. For larger sizes, we extrapolate $c_{transfer\_NDP}$ (Eq. (5)) based on the 1MB latency (*NDP Extrapolated*, Figure 3).

Workload. We execute a query varying its *selectivity* and *projectivity* (i.e. number of projected attributes):

```
SELECT proj FROM nodetable WHERE type < sel.
```

Table 3: Parameters of the NDP-aware cost-model

Variable	Description
MRL	Mean record length [bytes] (MySQL statistics)
PB	Projected record length [bytes]
BS	Block size [bytes]
BTT	Block transfer time
$RDBST$	RocksDB scan time
DEV_{type}	NDP scan acceleration factor w.r.t. device type
REC	Row evaluate cost
REC_{NDP}	NDP row evaluate cost
$n_{records}$	Number of records
SEL	Estimated selectivity (with key) otherwise 1

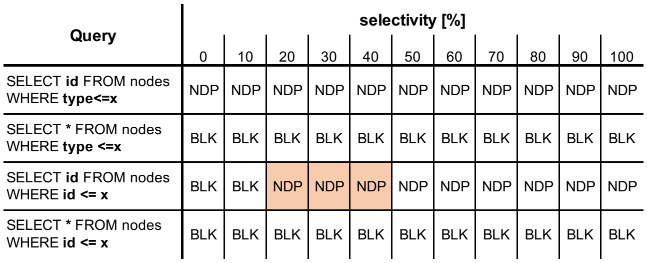
Execution is performed in *host-only* settings – based on cold-data, as well as in *NDP-only* settings – utilizing a single core on the COSMOS+ in an intervention-free manner.

Experiment 1: In an initial experiment (Figure 2) we stress MySQL with different selectivities SEL and projectivities PROJ. The goal is to investigate the parts of the problem space where the sub-optimal plans are chosen due to inexact cost estimation.

We observe that the wrong plan is selected for index-scans at the mid-selectivity range (Figure 2). The root cause for this behaviour is that the selectivity estimation for index-scans yields approximately the same costs and therefore the both cost models (NDP and host) differ in the *CPU* and transfer costs. That said, the cost estimations differ by a small margin and the real execution times for those selectivities are close to each other (Figure 3.b, 20%-40%).

Insight: Given the basic workload, the cost model estimates the costs mostly correctly, yielding appropriate plan selection.

Experiment 2: Selectivity. We continue our experimental evaluation by investigating the impact of varying the selectivity SEL from 0% to 100% (Figure 3.c). For this experiment we limit the projectivity to a single attribute – the ID column. Furthermore, the selection is performed on the *value attribute* TYPE. Consequently, the impact of typical auxiliary structures such as fence pointers and Bloom-filters is avoided, and only the effects of the scan are visible. Interestingly, the optimizer prefers early selection and NDP-execution to minimize the execution costs, which is inline with our



Query	selectivity [%]										
	0	10	20	30	40	50	60	70	80	90	100
SELECT **id** FROM nodes WHERE **type<=x**	NDP	NDP	NDP	NDP	NDP	NDP	NDP	NDP	NDP	NDP	NDP
SELECT * FROM nodes WHERE **type <=x**	BLK	BLK	BLK	BLK	BLK	BLK	BLK	BLK	BLK	BLK	BLK
SELECT **id** FROM nodes WHERE **id <= x**	BLK	BLK	NDP	NDP	NDP	NDP	NDP	NDP	NDP	NDP	NDP
SELECT * FROM nodes WHERE **id <= x**	BLK	BLK	BLK	BLK	BLK	BLK	BLK	BLK	BLK	BLK	BLK

Figure 2: Overall, the cost-model estimates the correct execution plan. Only index-scans with projection are estimated falsely for certain selectivities.

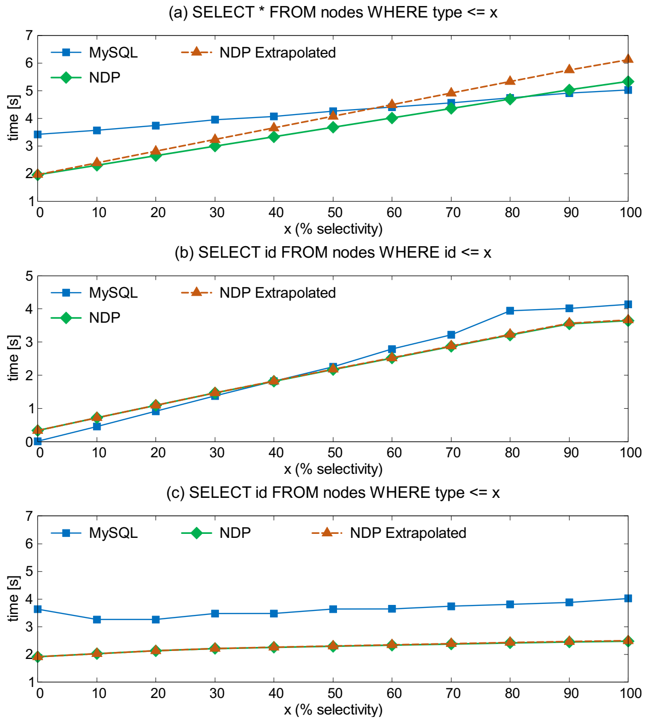


Figure 3: Comparison of MySQL and Extrapolated NDP execution times (lower is better) for different types of projection- and selection- queries showing the benefits and limitations of computational storage execution.

initial hypothesis. As a result the in-situ selection is faster for all selectivities.

Insight: The expected performance impact of early selection and in-situ filtering is correctly preferred by the cost model, yielding 2× performance improvement.

Furthermore, we investigate the impact of varying selectivity SEL on NDP index-scans (Figure 3.b). Because of the underlying LSM-tree storage organization, it is reasonable to claim that the experiment resembles a clustered index. We leave experiments with unclustered indices for future work. Our expectation is that increasing selectivities will have lower impact on the NDP index-scan because of the better I/O and result set transfer properties. We observe that for SEL ≥ 40% the NDP scan is faster. This is because the in-situ random I/O has low overhead and is as fast as sequential (see Table 1).

Another hypothesis is that the NDP index-scan curve (Figure 3.b) should remain essentially flat with increasing selectivities because of the low-overhead in-situ I/O. However, we observe the NDP curve increases (Figure 3.b), indicating rising NDP index-scan costs. This is due to memcpy overhead on ARM (Table 2), which is proportional to the number of records to be processed and the selectivity.

Insight: Considering the better on-device I/O properties, the in-situ index-scan outperforms the host for high selectivities.

Table 4: Execution time delta [ms] to measure the expensiveness of `memcpy`-operations on device

Projectivity (columns)	Selectivity [%]			
	30	50	80	100
4 byte (id)	baseline			
8 byte (id, type)	+41	+67,96	+106.63	+134,87
12 byte (id, type, time)	+84,71	+141,82	+226, 61	+284,06

Experiment 3: Selectivity and Projectivity. In the last experiment we investigate the effect of result set transfer (Figure 3.a). To this end we repeat Experiment 2, Figure 3.c with full PROJ.

Our expectation, based on experiments 1 and 2, is that the performance is sensitive to size of the result-set, which in turn depends on the projectivity (and on the selectivity).

As expected, we observe that with full PROJ the performance decreases (NDP execution time increases, Figure 3.a), yet with SEL ≤ 60% NDP is faster. As already mentioned, we extrapolate the NDP performance (*NDP Extrapolated*) for result-sets ≥ 1MB, to circumvent a hardware limitation in the COSMOS+ DMA-engine.

The limiting factor is the amount of ARM processing and especially the memcpy overhead needed to perform NDP projection, while the DMA transfers themselves incur relatively low overhead. We investigate this effect, by reporting (Table 4) the execution times for different projectivities and selectivities, relative to PROJ of the column ID from Experiment 2 (Figure 3.c). Clearly, the overhead of to preparing the result-set in the expected format in-situ is significant. Interestingly, the portion of DMA transfer times is marginal.

Insight: The result-set processing overhead has a significant performance overhead and necessitates dedicated optimizations.

4 CONCLUSION AND FUTURE WORK

In this work we make the case for NDP-aware cost models for computational storage, as in such settings DBMS operations come in Host- and NDP-flavours and an automated selection is needed. Furthermore, we propose an initial NDP-aware cost model. Given the basic workload, the cost model estimates the costs mostly correctly, yielding appropriate plan selection. We also demonstrate the impact of early selection and early projection as well as in-situ result-set processing.

In future we see much potential and intend to extend the work to cover: (a) cooperative host/NDP plans; (b) other NDP-operations, i.e. JOIN or GROUP BY; (c) auto-tuning model parameters based on Integer-Linear-Programming to address heterogeneous computational storage hardware. Especially targeting cooperative host/NDP executions as well as heterogeneous hardware is a challenge because of its expected impact on DBMS operations in terms of reducing data transfers and targeting hardware/software co-design and configurability.

ACKNOWLEDGMENTS

The authors wish to thank the anonymous reviewers for the valuable comments, which significantly improved the quality of the paper. This work has been partially supported by *BMBF PANDAS – 01IS18081C/D*; *DFG neoDBMS – 419942270*; *HAW Prom, MWK, Baden-Württemberg, Germany*.

REFERENCES

[1] Ian F. Adams, John Keys, and Michael P. Mesnier. 2019. Respecting the Block Interface - Computational Storage Using Virtual Objects. In *Proc. FAST 2019*.

[2] Wei Cao, Yang Liu, and et al. 2020. POLARDB Meets Computational Storage: Efficiently Support Analytical Workloads in Cloud-Native Relational Database. In *USENIX FAST*. 29–41.

[3] Arup De, Maya Gokhale, Steven Swanson, and et. al. 2013. Minerva: Accelerating Data Analysis in Next-Generation SSDs. In *Proc. FCCM*.

[4] David DeWitt and Jim Gray. 1992. Parallel Database Systems: The Future of High Performance Database Systems. *Commun. ACM* 35, 6 (June 1992), 85–98.

[5] Facebook. 2012. Facebook Graph Benchmark. https://github.com/facebookarchive/linkbench.

[6] Facebook. 2020. RocksDB. https://github.com/facebook/rocksdb.

[7] Zsolt István, David Sidler, and Gustavo Alonso. 2017. Caribou: Intelligent Distributed Storage. In *Proc. VLDB 2017*.

[8] Insoon Jo, Duck-ho Bae, and et al. 2016. YourSQL : A High-Performance Database System Leveraging In-Storage Computing. In *Proc. VLDB*.

[9] Sungchan Kim, Hyunok Oh, Chanik Park, Sangyeun Cho, Sang-Won Lee, and Bongki Moon. 2016. In-storage processing of database scans and joins. *Inf. Sci. (Ny)*. 327 (jan 2016), 183–200.

[10] Gunjae Koo, Kiran Kumar Matam, Te I, H. V. Krishna Giri Narra, Jing Li, Hung-Wei Tseng, Steven Swanson, and Murali Annavaram. 2017. Summarizer: Trading Communication with Computing NearStorage. In *Proc. MICRO-50 '17*. 219–231.

[11] Yoshinori Matsunobu, Siying Dong, and Herman Lee. 2020. MyRocks: LSM-Tree Database Storage Engine Serving Facebook's Social Graph. *Proc. VLDB Endow.* 13, 12 (Aug. 2020), 3217–3230.

[12] Sang-woo Jun Ming, Arvind, and et al. 2015. BlueDBM: An Appliance for Big Data Analytics. *Proc. ISCA* (2015).

[13] OpenSSD Project 2021. *COSMOS Project Documentation*. OpenSSD Project. http://www.openssd-project.org/wiki/Cosmos_OpenSSD_Technical_Resources.

[14] Erik Riedel, Christos Faloutsos, Garth A. Gibson, and David Nagle. 2001. Active disks for large-scale data processing. *Computer*. 34, 6 (2001), 68–74.

[15] Tobias Vincon, Lukas Weber, Arthur Bernhardt, Andreas Koch, and Ilia Petrov. 2020. nKV: Near-Data Processing with KV-Stores on Native Comp. Storage. In *Proc. DaMoN 2020*.

[16] Louis Woods, Zsolt István, and Gustavo Alonso. 2014. Ibex: An Intelligent Storage Engine with Support for Advanced SQL Offloading. *Proc. VLDB* (2014).