nKV in Action: Accelerating KV-Stores on Native Computational Storage with Near-Data Processing

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

Lukas Weber, Andreas Koch
[surname]@esa.tu-darmstadt.de
Embedded Systems and Applications Group,
Technische Universität Darmstadt

ABSTRACT

Massive data transfers in modern data-intensive systems resulting from low data-locality and data-to-code system design hurt their performance and scalability. Near-data processing (NDP) designs represent a feasible solution, which although not new, has yet to see widespread use.

In this paper we demonstrate various NDP alternatives in nKV, which is a key/value store utilizing *native computational storage* and *near-data processing*. We showcase the execution of classical operations (*GET*, *SCAN*) and complex graph-processing algorithms (*Betweenness Centrality*) in-situ, with 1.4×-2.7× better performance due to NDP. nKV runs on real hardware - the *COSMOS+* platform.

PVLDB Reference Format:
Tobias Vinçon, Lukas Weber, Arthur Bernhardt, Christian Riegger, Sergey Hardock, Christian Knoedler, Florian Stock, Leonardo Solis-Vasquez, Sajjad Tamimi, Andreas Koch, Ilia Petrov. nKV in Action: Accelerating KV-Stores on Native Computational Storage with Near-Data Processing. *PVLDB*, 13(12): 2981 - 2984, 2020.
DOI: https://doi.org/10.14778/3415478.3415524

1. INTRODUCTION

Besides substantial data ingestion, yielding an exponential increase in data volumes, modern data-intensive systems perform complex analytical tasks. To process them, systems trigger massive *data transfers* that impair performance and scalability, and hurt resource- and energy-efficiency. These are partly caused by the scarce bandwidth in combination with poor data locality, but also result from traditional (*data-to-code*) system architectures.

Near-Data Processing (NDP) is a *code-to-data* paradigm targeting in-situ operation execution, i.e. as close as possible to the physical data location, utilizing the much better on-device I/O performance. NDP leverages several trends. Firstly, hardware manufacturers can fabricate *combinations of storage and compute* elements economically, and package them within the same device – so called NDP-capable *computational storage*. As a result, even commodity storage devices nowadays have compute resources that can be effectively used for NDP, but are executing compatibility firmware (to traditional storage) instead. Secondly, with semiconductor storage technologies (NVM/Flash) the *device-internal* bandwidth, parallelism, and latencies are significantly better than the external ones (device-to-host). Both lift major limitations of prior approaches like ActiveDisks or Database Machines.

In this paper, we demonstrate nKV, which is a RocksDB-based key/value store utilizing *native computational storage* and *near-data processing* (Figure 1). nKV eliminates intermediary layers along the I/O stack (e.g. file system) and operates directly on NVM/Flash storage. nKV directly controls the physical data placement on chips and channels, which is critical for utilizing the on-device I/O properties and compute parallelism. Furthermore, nKV can execute access operations like *GET* or *SCAN*, or more complex graph processing algorithms such as *Betweenness Centrality* as *software NDP* on the ARM cores or with FPGA hardware support (NDP:HW+SW). Under nKV we target *intervention-free* NDP-execution, i.e. the NDP-device has the complete address information, can interpret the *data format*, and access the data in-situ (without any *host interaction*). To reduce data transfers nKV also employs novel *ResultSet*-transfer modes. nKV is resource efficient as it eliminates compatibility layers and utilizes freed compute resources for NDP.

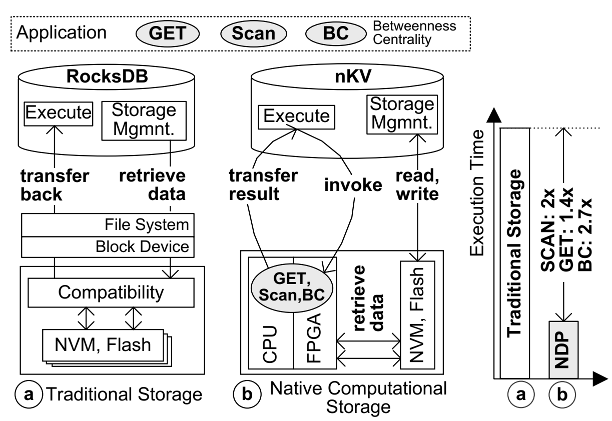

Figure 1: KV-Store transferring data along a traditional I/O stack (a); and (b) nKV executing operations in-situ on native computational storage.

Proceedings of the VLDB Endowment, Vol. 13, No. 12
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3415478.3415524

We demonstrate nKV for the use-case of a database of research papers, and on a *2.4GB* graph dataset with *48 million* KV-pairs. Our demonstration scenarios involve interacting with the paper DB, browsing and analyzing it: (a) *Analysis scenario (BC)*: verifies if the 10-year best paper award was awarded the most prominent paper from 10 years ago and offers some unexpected insights; (b) *Latency-based (GET)*: we let the audience pick a paper from the BC ResultSet and display its details; (c) *Bandwidth-based (SCAN)*: we retrieve other papers from same Venue/Author/Year. nKV performs 1.4×-2× better than RocksDB: *GET latency* – 1.4×; *SCAN bandwidth* – 2×; *Betweenness Centrality* – 2.7×.

2. ARCHITECTURE OF nKV

This section offers a brief overview of the key architectural modules of nKV. More details are provided in [16].

NDP Interface Extensions. nKV defines NDP-Extensions besides the native storage interface. Furthermore, nKV has a dedicated high-performance *in-DBMS* NVMe layer (Figure 2). It does not rely on an NVMe kernel driver, but is deeply integrated in the DBMS and hence runs in *user-space*. The *native NVMe* integration reduces the I/O overhead, as it avoids expensive switches between user and kernel space (drivers), and shortens the I/O even further, as no drivers are needed. *This lean stack improves execution times for I/O and NDP, especially for short-running calls e.g. GET.*

Computation Placement. By using native computational storage, nKV can place computations directly on the heterogeneous on-device compute elements, such as ARM CPUs or the FPGA. nKV can execute various operations such as *GET* or *SCAN*, or more complex graph processing algorithms like *Betweenness Centrality* as *software NDP* on the ARM cores, or with *hardware* support from the FPGA. nKV demonstrates that hardware implementations alone cannot reach the best performance as pure software implementations do not. For its NDP-operations nKV utilizes *hardware/software co-design* to handle the proper separation of concerns and achieve best performance.

In-situ data access and interpretation. Under nKV the NDP-device can interpret the *data format* and *access* the data without *host intervention*. To this end, nKV extracts definitions of the *Key- and Value-formats* [14]. These are then passed as input parameters to NDP-commands. Moreover, the *data format* such as the *Key- and Value-formats* can be automatically extracted from the DB catalogue (*system-defined*), or can be defined by the *application*.

nKV employs a thin on-device *NDP-infrastructure* layer that supports the execution and simplifies the development of NDP-operations (Figures 2). It comprises *data format parsers* and *accessors* that are implemented in both *software* and *hardware* (Figure 3). The in-situ *accessors* are used to traverse and extract the contained sub-entities of the persistent data. Whereas, the in-situ *data format parsers* process the *layout* of each persistent entity, and extract the sub-entities by invoking further accessors (Figure 3).

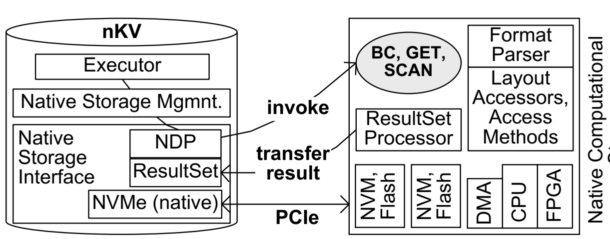

Figure 2: Architecture of nKV

KV-Stores like LevelDB or RocksDB organize the persistent LSM-Tree data in to so called *Sorted String Tables (SST)*. To process a *GET(key)* request, for instance, nKV first identifies the respective *SST* and invokes an *NDP_GET()* command with the physical address ranges (of these SSTs), the respective *Key- and Value-formats* as well as further parameters. First, the *SST layout accessor* is invoked to access data and index blocks. Subsequently, the index block parser is invoked to interpret the data, verify if the *key* is present, and extract its location. If this is the case, the data block accessor and parser are invoked to extract the *Key/Value entry*. In case of an *NDP_SCAN(key_val_condition)* operation, the KV accessor is subsequently invoked to extract it, followed by a *field* parser to extract its value and verify the *condition*. The result are massive I/Os since especially SCANs must retrieve a huge number of data blocks.

Native computational storage. To make efficient use of the on-device I/O nKV extends [15] and employs *native storage* (Figures 1 and 2). This way it eliminates intermediary layers along the critical I/O path like the file system, and can operate directly on NVM/Flash storage using physical addresses. nKV can therefore precisely control physical placement of SST data, which is critical for utilizing the on-device I/O properties and compute parallelism. I.e. nKV physically places *SST data blocks* and *SST index blocks* on different *LUNs* and *Channels* to utilize the on-device parallelism and lower the processing latency (see Figure 3). This accelerates especially the demonstrated I/O-intensive operations SCAN and BC significantly. *Native storage* is essential for reducing read- and write-amplification, and also for executing NDP-operations avoiding information hiding through these layers of abstraction.

ResultSet Handling. nKV aims to bulk-transfer the *ResultSet* of an NDP-Operation to avoid the data transfer overhead caused by a *record-at-a-time* model. Thus nKV materializes the ResultSet, partially or fully, depending on the NDP operation. It is then DMA-transferred with multiples of the COSMOS+'s DMA-engine transfer unit (4KB).

3. DEMONSTRATION WALK-THROUGH

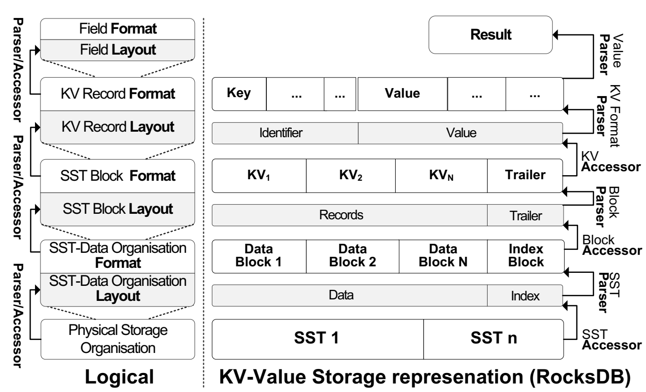

Figure 3: In-situ access and data interpretation in nKV, based on layout accessors and format parsers.

Demo Setup. The demonstration setup comprises a desktop PC as host equipped 3.4 GHz Intel I5 CPU, 4 GB RAM, connected to *COSMOS+* via NVMe over PCIe (Figure 4). The COSMOS+ [11] has a Zynq 7045 SoC with an FPGA, two 667 MHz ARM A9 CPU Cores and an MLC Flash module configured as SLC. We configure both RocksDB and COSMOS+ with 5 MB cache.

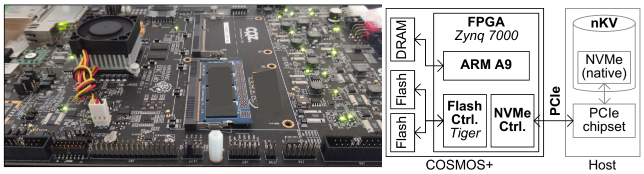

Figure 4: COSMOS+ and the Demonstration Setup

We demonstrate nKV on the use-case of a database of research papers, and on a rather smaller *2.4GB* dataset due to practical runtime constraints of the demo. This graph dataset includes *48 million* Key/Value-pairs, comprising approx. *3.8M* papers, *40M* references, *18K* venues, and *4.2M* authors. *BC* operates on a graph with varying number of relevant edges: from *2.5K* to *2 million*. The audience will browse and analyze the paper set using a GUI (Figures 5), triggering different operations on the paper graph in different scenarios.

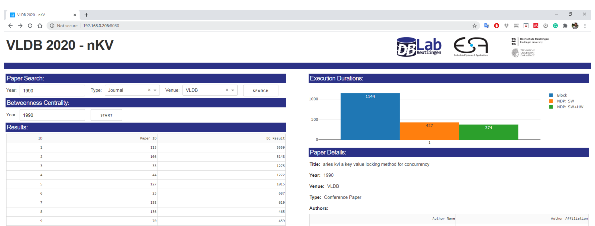

Figure 5: Interactive GUI.

3.1 Demonstration Walk-Through

1. Complex Graph Analysis – ***BC.*** The demo starts by letting the audience pick a *DB conference venue* and an *year* (e.g., *VLDB, 2000*). Subsequently, nKV executes *Betweenness Centrality* to determine the most prominent paper from that year. The audience can then verify if that paper had indeed been awarded the *10-year best paper award* ten years later. Expect some unexpected(!) insights.

Under the hood, nKV executes a complex NDP operation pipeline, comprising a SCAN followed by a BC. Based on the audience selection, nKV first filters out the relevant papers and references by running a *SCAN* and applying *val_condition* on the *values* of all paper KV-pairs. This is only possible since the data formats are available in-situ, and the *format parses* and *layout accessors* execute on-device. The intermediary result is materialized on-device, which is essential for such NDP-pipelines. Subsequently, BC is executed on the intermediary results. nKV switch between software NDP or software/hardware NDP. We demonstrate how the hardware accessors and parsers can be instantiated multiple times, and run in parallel on the FPGA yielding best results.

Observation: nKV executes NDP-pipelines and complex operations in-situ. Given the high parallelism and compute intensity, NDP:SW+HW yields best results.

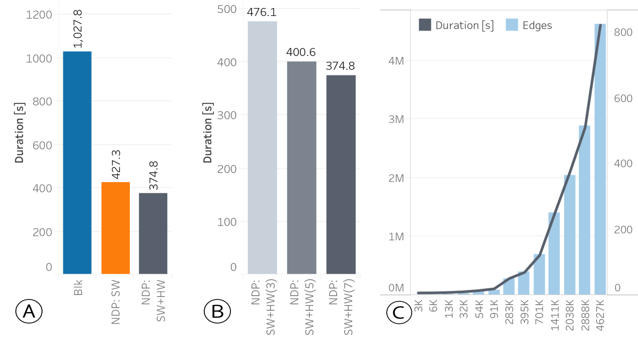

Figure 6: Betweenness Centrality: (A) BC on different stacks; (B) BC with different levels of parallelism; (C) BC execution time vs number of relevant edges (complexity).

2. Latency – ***GET.*** After the *BC* analysis, the audience can interactively pick a paper from the BC ResultSet and have its details displayed.

Under the hood, the NDP execution of GET is performed in SW and in NDP:SW+HW. Since only a single NDP_GET() is executed at a time, nKV utilizes native data placement, but not the on-device parallelism.

Observation: Latency-critical operations are 1.4× faster and best results are achieved with NDP:SW, closely followed by NDP:SW+HW (Figure 7).

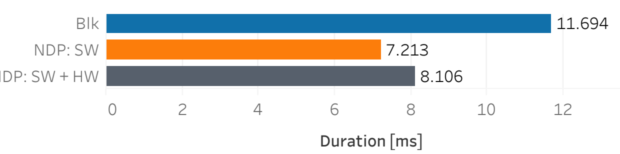

Figure 7: GET Latencies on different stacks.

3. Bandwidth – ***SCAN.*** After the audience has been presented the details of a paper (previous scenario), they can opt for retrieving other papers from the same Venue/Author/Year.

Under the hood, this results in an NDP *SCAN(value_condition)*. The operation is performed with different selectivities and different result set sizes, based on the audience selection (Figure 8a). Importantly, the selection condition is on the value, which requires NDP format parsers and layout accessors to be evaluated in-situ. Conversely, the *Blk* RocksDB stack transfers the entire data to the host, to interpret the values there, apply the *val_condition*, and eventually discard most of the data. Figure 8b shows the extra read volume transferred by the *Blk* to perform the same SCAN.

Observation: Bandwidth-critical scan and selection operations require I/O bandwidth and high hardware parallelism. Hence, NDP:SW+HW is best and outperforms the traditional stack by 2×.

4. Parallelism and Native Computational Storage Last but not least we execute BC again, however this time we demonstrate the effect of *configurable parallelism* in native computational storage, whenever nKV executes a complex operation (Figure 6b).

nKV can configure the degree of parallelism required by each NDP-operation. While the amount of compute paral-

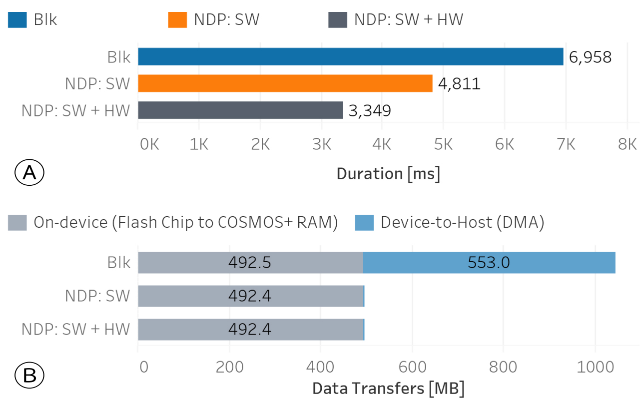

Figure 8: SCAN performance:(A) SCAN on different stacks; (B) Data Transfer Volume

lelism is limited for NDP:SW, as there are few ARM cores, the same does not apply to the FPGA. As described in Section 2, there can be *multiple parallel instances* of the hardware accessors and parsers on the FPGA. These are relatively space-efficient, as 16 instances fit even into the small Zynq 7045 FPGA. Interestingly, operating with the maximum available parallelism does not always yield the the best results (Figure 6c).

Observation: nKV can employ the FPGA for NDP:SW+HW, increasing the level of computational storage parallelism. However, this capability only translates into performance benefits for *complex* operations.

4. RELATED WORK

The Near-Data Processing approach is deeply rooted in well-known techniques such as *database machines* or Active Disk/IDISK. With the advent of Flash technologies and reconfigurable processing elements Smart SSDs [3, 13, 7] were proposed. An FPGA-based intelligent storage engine for databases is introduced with IBEX [17]. JAFAR [18, 1] is one of the first systems to target NDP for Column-stores use, whereas [6, 9] target joins besides scans. Recently, Samsung announced its KV-SSD [12]. The use of NDP in the realm of KV-Stores has been investigated in [8, 2]. Kanzi [4], Caribou [5] and BlueDBM [10] are RDMA-based distributed KV-Stores investigating node-local operations.

Much of the prior work on persistent KV-Stores and NDP focuses on *bandwidth* optimizations. NoFTL-KV [15] addresses the problem of *write-amplification*. The NDP extensions demonstrated by nKV target the *read-amplification*, *latency improvements* and *computational storage*.

5. CONCLUSION

We demonstrate nKV, which is a key/value store utilizing *native computational storage* and *near-data processing*. We showcase the execution of classical operations (*GET*, *SCAN*) and complex graph-processing algorithms (*Betweenness Centrality*) in-situ, with 1.4×-2.7× better performance due to NDP. nKV runs on real hardware - the *COSMOS+* platform. nKV utilizes the the available I/O and compute parallelism on the native computational storage through direct data and operation placement. Complex operations (BC, SCAN) benefit from it, whereas others (GET) benefit from software NDP.

Acknowledgments. This work has been partially supported by *BMBF PANDAS – 01IS18081C/D*; *DFG Grant neoDBMS – 419942270*; *HAW Promotion* and *KPK Service Computing MWK, Baden-Württemberg, Germany.*

6. REFERENCES

[1] O. O. Babarinsa and S. Idreos. JAFAR : Near-Data Processing for Databases. 2015.
[2] A. De, M. Gokhale, S. Swanson, and e. al. Minerva: Accelerating data analysis in next-generation ssds. In *Proc. FCCM*, 2013.
[3] J. Do, J. Patel, D. DeWitt, and et al. Query processing on smart ssds: Opportunities and challenges. In *Proc. SIGMOD*, 2013.
[4] M. Hemmatpour, M. Sadoghi, and et al. Kanzi: A distributed, in-memory key-value store. In *Proc. Middleware*, 2016.
[5] Z. István, D. Sidler, and G. Alonso. Caribou: Intelligent distributed storage. *PVLDB*, 10(11):1202 1213, 2017.
[6] I. Jo, D.-H. Bae, A. S. Yoon, J.-U. Kang, S. Cho, D. D. G. Lee, and J. Jeong. Yoursql: A high-performance database system leveraging in-storage computing. *PVLDB*, 9(12):924935, 2016.
[7] Y. Kang, Y.-s. Kee, and et al. Enabling cost-effective data processing with smart SSD. In *Proc MSST*, 2013.
[8] J. Kim and et al. Papyruskv: A high-performance parallel key-value store for distributed nvm architectures. In *Proc. SC*, 2017.
[9] S. Kim, S.-W. Lee, B. Moon, and et al. In-storage processing of database scans and joins. *Inf. Sci.*, 2016.
[10] S.-w. J. Ming, Arvind, and et al. BlueDBM: An Appliance for Big Data Analytics. *Proc. ISCA*, 2015.
[11] OpenSSD Project. *COSMOS Project Documentation*, January 2019. `http://www.openssd-project.org`.
[12] Samsung. *KV-SSD*. `https://github.com/OpenMPDK/KVSSD`.
[13] S. Seshadri, S. Swanson, and et al. Willow: A User-Programmable SSD. *USENIX, OSDI*, 2014.
[14] T. Vincon, A. Bernhardt, L. Weber, A. Koch, and I. Petrov. On the necessity of explicit cross-layer data formats in near-data processing systems. In *Proc. HardBD@ICDE*, 2020.
[15] T. Vincon, S. Hardock, C. Riegger, J. Oppermann, A. Koch, and I. Petrov. Noftl-kv: Tackling write-amplification on kv-stores with native storage management. In *Proc. EDBT*, 2018.
[16] T. Vincon, L. Weber, A. Bernhardt, A. Koch, and I. Petrov. nKV: Near-Data Processing with KV-Stores on Native Computational Storage. In *Proc. DaMoN*, 2020.
[17] L. Woods, J. Teubner, and G. Alonso. Less watts, more performance: An intelligent storage engine for data appliances. In *Proc. SIGMOD*, 2013.
[18] S. Xi, O. Babarinsa, M. Athanassoulis, and S. Idreos. Beyond the Wall: Near-Data Processing for Databases. *Proc. DAMON*, 2015.