DASH: Asynchronous Hardware Data Processing Services

Norman May
Daniel Ritter
Andre Dossinger
SAP SE
Germany
firstname.lastname@sap.com

Christian Färber
Intel Corporation
Germany
christian.faerber@intel.com

Suleyman S. Demirsoy
Intel Corporation (UK) Ltd.
United Kingdom
suleyman.demirsoy@intel.com

Abstract

As databases need to strike a good balance of high performance at low costs, hardware-accelerators like FPGAs are increasingly considered to improve core database operations. In this paper we complement previous research, where hardware accelerators are tightly integrated with the host CPU, and expose the FPGA as a hardware data processing service to a cluster of databases to delegate the asynchronous execution of compute-intensive database operations. We apply our novel architecture, called DASH, to the use case of string dictionary compression using the compute-intensive RePair compression and demonstrate its benefit in this distributed architecture. We also explain how DASH can be applied to other database scenarios and propose a research agenda.

1 Introduction

Databases need to strike a good balance between high performance at a low price. Hardware accelerators promise to accelerate or offload compute-intensive operations from general purpose CPUs. As hardware-accelerators like FPGAs become more widely available, e. g., offered by cloud providers as dedicated FPGA-equipped compute nodes [2, 6], researchers investigated how to apply FPGAs in the context of databases [15]. In this research it was demonstrated that FPGAs can help to accelerate database operations, to be more energy efficient or in general more cost-efficient.

However, prior research assumes a setup where the FPGA is deployed with the database processing server where the FPGA communicates with the host CPU via PCIe or UPI interconnects that offer high bandwidth and fairly low latencies as illustrated in Figure 1 (marked green). Other options explored put the FPGA on the I/O path – either to network, disk or memory as a "bump in the wire" as shown in Figure 1 (marked blue). Still, the data transfer between FPGA and host CPU is often considered a bottleneck which is either limited by latency or bandwidth of the interconnect.

In this paper, we approach the use of hardware accelerators from a different perspective. We focus on highly compute-intensive database operations that can be performed asynchronously and where the transfer time of the data and completion time of the tasks relative to the initiation time are less critical. Examples of such operations include reorganizing or compressing the persistent data, collecting statistics or consistency checks. With the service

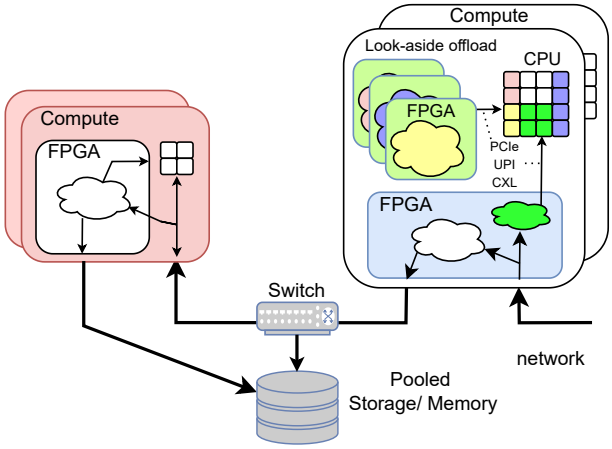

Figure 1: Compute topologies in the cloud and data centers with FPGAs. Focus of interest of this paper is marked red.

oriented architecture of the cloud native database services, such tasks could be deployed on accelerators that are declared to the system as accelerator-based services. As marked red in Figure 1, these accelerator attached nodes are connected to other parts of the database engines via network links. Exposing the FPGA operations as a service enables sharing the FPGA between multiple database instances and scaling the FPGAs based services elastically based on the demand.

As we discuss in more detail in Section 2, this setup is consistent with the trend to build cloud-native databases that factor database operations into independent services, e. g., compute and store nodes, optimizer services etc. [4, 5, 24]. Database implementers seek to minimize the hardware costs to support database workloads through this separation of components, and at the same time, it allows them to scale components independently. As a logical consequence of this development, we argue that certain components can be deployed on specialized hardware. While, in principle, such asynchronous operations could also be executed on CPUs, FPGAs – or depending on the specific task GPUs or TPUs – may offer better performance and energy efficiency than a CPU making them attractive for the service-based architecture proposed in this paper. The service-oriented approach also allows for choosing operations that fit well to the processing model of the FPGA, i. e., compute-intensive operations that can utilize the inherent pipelining and parallelization of the FPGA. With this we also recognize the difficulties implementing a full-fledged database on an accelerator where prior work had to rely on simplifying assumptions related to the transaction processing model or supported workload [7, 19, 23].

 13th Annual Conference on Innovative Data Systems Research (CIDR '23). January 8-11, 2023, Amsterdam, The Netherlands.

In this paper, we present DASH which exposes compute intensive data processing services to a cluster of databases and that runs on FPGA-accelerated compute nodes, in Sect. 3. The architecture we present for DASH can accommodate a broad set of use cases that are not latency-critical, e. g., including machine learning or non-relational data processing [12] like graph [10, 11] or JSON processing [13]. We then exemplify the detailed use model of this generic architecture in Sect. 4 through one use case, string dictionary compression. String compression is known to be compute intensive when the goal is to achieve high compression ratios for arbitrary collections of strings [21]. In our experiments we quantify the benefits of using the DASH architecture from a performance and cost perspective. We then explain how the FPGA accelerators are exposed to a cluster of database instances and how this setup allows to accelerate different functions in parallel supporting elastic scaling in response to fluctuating user demand. From our findings with DASH we propose a research agenda in Sect. 5 as starting point for further research in this area, before we conclude in Sect. 6.

2 Background and Related Work

In this section we provide some background on recent trends in FPGA hardware and especially how they are used in cloud environments in general. We also discuss related work in the field of hardware acceleration using FPGAs in the database context.

2.1 Accelerator Hardware in the Cloud

Hardware accelerators in the cloud exist in different deployment variants and form factors. All major hyperscalers either employ FPGA internally in their infrastructure [5, 17] or expose them to customers as compute instances [2, 6] for offload accelerators, which can be extended to a FPGA-as-a-Service to accelerate distributed micro services in the cloud [5]. Tightly packed server instances are available with multiple accelerator cards[1] that offer large amounts of accelerator compute capacity to be exposed as a service.

PCIe-based cards are the most widely used form factor for FPGAs or GPUs. In addition, tightly integrated CPU+FPGA systems that utilize both the PCIe and UPI interfaces offere higher bandwidth for the data exchange between CPU and FPGA tasks as well as reduced communication latency [8, 20] between different end points.

Prior work showed the benefits of pushing FPGA-accelerated computation on the I/O path [28]. Another major use case of FPGA in the cloud is to keep the infrastructure flexible with smartNICs and to use these for inline processing on the network interface [18].

The above-mentioned scenarios might become more attractive in the future as Compute Express Link (CXL)[2] offers coherency for the cache (CXL.cache) and memory expansion (CXL.memory) on the accelerator nodes on top of the usual IO exchanges (CXL.io) over the PCIe interface. PCIe- or CXL-attached FPGA boards may be equipped with on-board memory. With CXL, such memory is visible to the host CPU as additional NUMA node. Moreover, FPGAs can provide abstraction for different memory and storage technologies to get used seamlessly within the server application.

[1]NVidia DGX servers, visited 12/22: https://www.nvidia.com/en-us/data-center/dgx-systems/ or Bittware Terabox servers https://www.bittware.com/fpga/servers-systems/
[2]https://www.computeexpresslink.org/about-cxl

2.2 Related Work: Hardware Accelerators for Databases

Abadi [1] and Narasayya [24] confirm the trend that databases are increasingly offered as managed services in the cloud. In this context, the cloud service provider may have more control over the deployed hardware than in an on-premise setup where the lifecycle of deployed hardware and software is completely under control of the customer. Hence, the availability of new hardware, in particular hardware accelerators like FPGA or GPUs, in cloud environments opens the opportunity to use accelerators for databases in these managed service environments [5, 25, 26].

Various options were explored to integrate FPGAs with database system, and most of these approaches follow an architecture where the FPGA attached via fast interconnects like UPI or PCIe to the host where the main database executes following a "bump-in-the-wire" approach to accelerate database operations [15]. FPGA were deployed on the storage path, e. g., [9, 28] to reduce the amount of data to transfer to the host, or to compress and encrypt data on the I/O path. FPGA were also used to accelerate core database operations, e.g. the join operation [8, 20] where the FPGA was either tightly coupled to the host CPU via UPI interconnect or deployed on an discrete card with on-board memory available during query processing. Additional work regarding database query processing with FPGAs shows competitive performance of the database sort-merge operator with a morphing operator approach to save resources on the FPGA and be able to implement pipeline breaking operators in an single FPGA [22]. The authors describe how they accelerated the sort-merge operation with an FPGA on average up to 5x compared to MonetDB using a modern CPU. Similarly, FPGA were deployed on the network path [27] of data processing systems. Finally, FPGAs served as accelerators to offload work from the host CPU, e. g., [3]. In most of these architectures, the latency of data transfer from and to the FPGA is an important factor.

According to Eryilmaz et al. [14] FPGAs become more widely offered by hyperscalers as another accelerator option next to GPUs and others. But they also raise several challenges which have to be considered and overcome to increase the adoption rate in the database community. These are resource ceiling, reprogramming time, available interface bandwidth on-board and to the host and also difficulty of programming. These challenges are well-known in the community, and later in the description of DASH we discuss how to overcome some of these system-level limitations. In general a lot of experience was already collected with the programming of FPGA without RTL design knowledge, e. g., with high level languages like Intel® oneAPI™. Therefore we argue that options exist for programmers capable in C++ to also address FPGAs as target platform for acceleration. We also expect that new generations of FPGAs will lift important hardware limits, e. g., the bandwidth limitation to the host with CXL.

3 DASH – Hardware Data Processing Services

In this section, we present our vision on asynchronous hardware data processing services, short DASH. DASH leverages cloud computing and next generation re-configurable hardware technology for complementing current cloud data processing off the critical

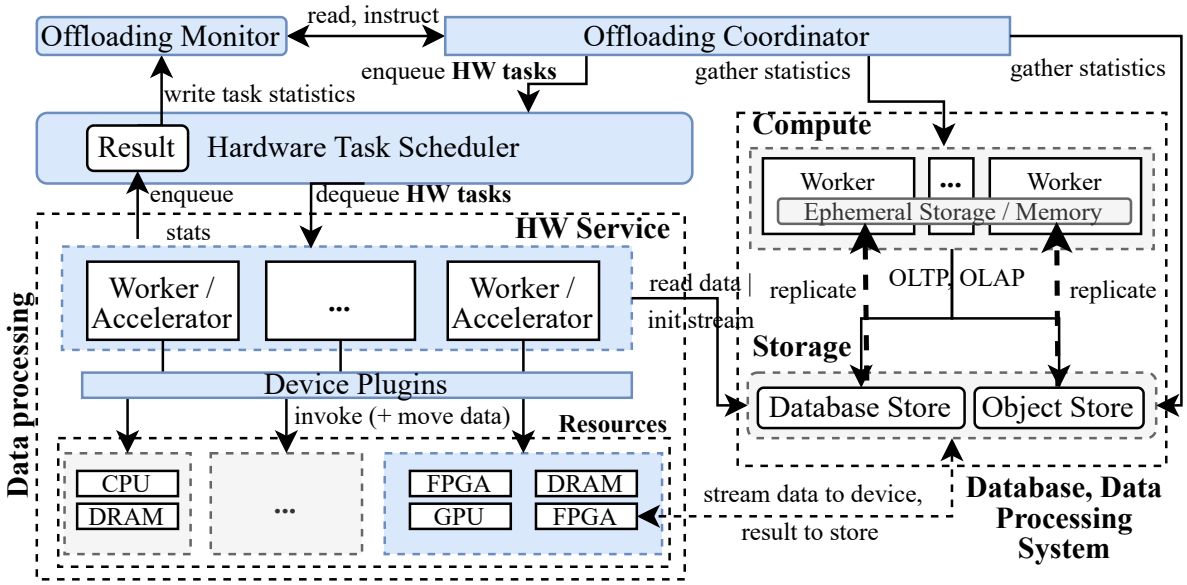

Figure 2: Asynchronous data processing with hardware offloading services (DASH components in blue)

path (i. e., offloading). Based on our vision, we describe a prototypical implementation with current cloud computing technology that we use in our experimental evaluation.

3.1 DASH Architecture

For efficient data processing off the critical path, we present DASH's architecture and high-level components (in blue), as shown in Fig. 2. We assume cloud databases or data processing systems with separate compute and storage components (e. g., [4, 5]), in which the query processing is done by scalable workers that get their data from the storage as ephemeral storage in memory. The storage component could be database storage or cloud object stores.

Asynchronous Data Processing. To keep the hardware accelerators off the critical path of data processing systems, the `offloading coordinator` is an (intelligent) component that observes the systems' internal states (e. g., metrics, statistics) and automatically decides on actions that improve their efficiency or health (e. g., reduce storage, memory through stronger compression of string dictionaries). The coordinator enqueues identified actions as hardware tasks for asynchronous processing and informs itself about the result of the task execution by checking the `offloading monitor` component to check and adapt for future actions (e. g., redo previous task on failure). The changes to the data resulting from the processing within the hardware services are re-distributed to the storage and eventually to the compute instances. One coordinator is sufficient for observing data processing instances of a data center.

Scheduling and Observing Hardware Services. For scheduling the hardware tasks while allowing for prioritization and flow control, we envision a topic-based `hardware task scheduler`. The scheduler gets hardware task specifications which it offers as topics to the hardware service components. The internal state of the hardware service is observed by the scheduler and forwarded to a `offloading monitor` component. The monitor allows users for checking the efficiency (e. g., compression times) and health of the service (e. g., failure rates, security attacks). Based on the observations, the coordinator can be instructed to apply countermeasures (e. g., take malfunctioning FPGAs out of the scheduling). The coordinator – without the need for user interaction – checks the success and efficiency of scheduled hardware tasks and reacts as discussed before.

Hardware Services / Compute. A `hardware service` denotes the compute component with attached hardware `resources` (e. g., FPGAs, GPUs). Each of the services has several software `workers / accelerator` (i. e., hosts), which use cloud-specific drivers, called `device plugins`, to attach and access hardware resources. Several resources can be bound to one worker. The resources have specific functions, whose bit-stream is configured on the hardware (e. g., Re-Pair compression [21], JSON parsing [13]) during the provisioning (not shown). Through the task specification and the pre-provisioned functions, the workers match and find tasks they can handle with the attached resources. The input data (e. g., string dictionary, table partition) can either be fetched by the workers or is directly streamed into the available resources. The processed data is routed in the same way. The workers collect and forward statistics of the tasks as well as device information through the device plugins to the scheduler.

Disaggregated Compute. While co-location is possible, DASH denotes elastic compute, that is separate of database / data processing compute (i. e., no co-location needed) and can be combined with multiple distributed compute clusters within the same data center. To address concerns like security and cost of transferred data, existing technologies for virtual networks or virtual private clouds like Private Link[3] could be used. Economic feasibility of disaggregated compute depends on data transfer pricing of the given platform or solution used.

[3]VPC technology, visited 12/22: AWS https://aws.amazon.com/de/privatelink/, MS Azure https://azure.microsoft.com/de-de/products/private-link/

3.2 Using DASH: Compression-as-a-Service

We implemented Compression-as-a-Service (CaaS) on a DASH prototype as part of SAP HANA Cloud[4] to illustrate and experimentally evaluate DASH in practice. As compression technique we selected the RePair algorithm, which we implemented in previous work [21], and use it to compress string dictionaries used in HANA Cloud. By default SAP HANA [16] workers compresses string dictionaries using front coding on the critical path. We assume that data can be compressed asynchronously with RePair directly using the image of the compressed string dictionary from storage. To illustrate the inner workings of the `hardware service workers`, as shown in Fig. 3, we subsequently explain its components in more detail and describe its realization with current cloud concepts as well as the execution flow.

CaaS Hardware Service Worker. We recall that hardware tasks are available in the `hardware task scheduler` with task specifications, and the data is provided by the database / data processing system, as shown in Fig. 3. A specification $\langle id, func, sid, tid \rangle$ contains a unique task identifier id, a function identifier $func$, which corresponds to available logic on attached resources, a handle for source data sid and one for target data tid (sink), among others.

For our prototype, we assume enterprise-ready, multi-cloud infrastructure such as Gardener[5] and realize the hardware service workers in Fig. 3 as kubernetes[6] (short `k8s`) pod that runs in a k8s cluster. A pod binds resources that are available via k8s nodes, using device plugins, for which we adapted the Intel IOFS[7] FPGA driver to run with k8s. The worker has a local `function scheduler` to manage a collection of `hardware functions`, which represent specific logic resources that denote a function on an FPGA (e. g., compression like RePair, K^2). To run a function, optional $\langle \{config\}, data \rangle$ information might be required, with a collection $config$ of configuration parameters and $data$ (in a non-streaming case). The scheduler reports at least $\langle id, exc, \{stats\} \rangle$ including the task identifier id, a status exc, and a name-value pair collection of statistics $stats$. The required data sources and targets are managed by a `storage gateway`, which is able to resolve $\langle sid \rangle$ and $\langle tid \rangle$, when reading data or initializing streams into the FPGA resources via on-board NICs. The overall architecture scales in three dimensions. Depending on their size, current FPGAs are able to configure few functions that correspond 1:1 to hardware functions within the workers. Tasks like RePair compression can be scaled by partitioning the data and distributing them to separate functions within one FPGA (only limited by bandwidth to the pod). Additionally, on-chip resources of FPGAs can be re-configured to impersonate another function. For example, if the provisioned K^2 function is not used, the resources could be used to configured another RePair function and update the corresponding hardware function in the pod. The number of functions can be scaled by adding multiple FPGAs as resources into the k8s cluster. With current data center architectures, we see eight to ten FPGAs attached to one hardware service worker. Consequently, scaling several workers might be necessary to fit the various workloads of databases and data processing systems within one data center.

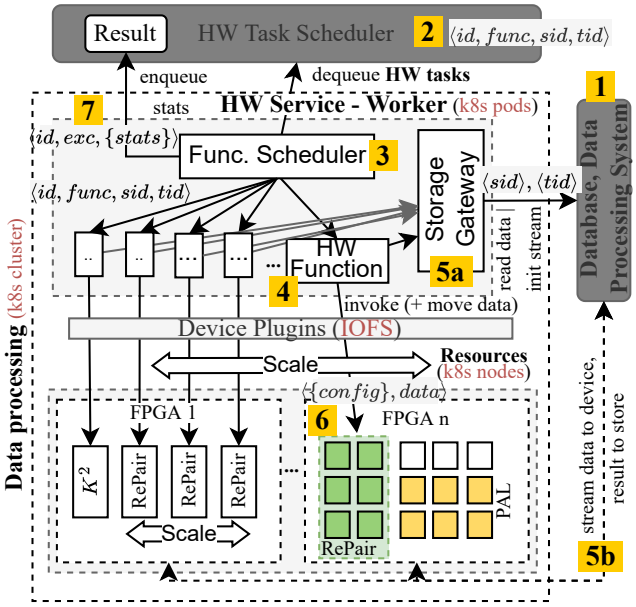

Figure 3: HW service worker and execution flow

CaaS Execution Flow. To understand the interaction between the main components within one hardware service worker, we step through the flow of executions by example of string dictionary compression with our RePair implementation.

First, during the data processing in the compute of the database system, 1 statistical records are gathered about table sizes, number of distinct values per column, sizes of dictionaries and column vectors, and storage requirements, and they are exposed via monitoring views. The coordinator (not shown) analyzes these statistical records an decides to conduct an action 2 and enqueues a hardware task in the hardware task scheduler. In our prototype, we first check if a table was merged because the merge operation creates a new version of a dictionary. We trigger the compression on the FPGA only for sufficiently large dictionaries. As we detail in Section 4 we also check for a minimal average size per dictionary element after applying front coding when scheduling the compression on the FPGA. When a hardware function within a worker is free and the task specification matches the function's abilities, 3, the scheduler dequeues and 4 assigns a task to that function. The function starts coordinating data access through the storage gateway to get the data from a data store 5a or 5b initiates the data stream into the function on the FPGA. During the execution of the task 6, the hardware function cannot work on newly arriving hardware tasks, but will collect statistics, 7 which it reports to the hardware task scheduler together with a status. As soon as the function is free, the hardware service worker can pick up new tasks fitting to the function, thus balancing the load.

[4]SAP HANA Cloud, visited 8/2022: https://bit.ly/3bqsxxm
[5]Gardener, visited 12/22: https://gardener.cloud/
[6]Kubernetes, visited 12/22: https://kubernetes.io/de/
[7]Intel IOFS, visited 12/22: https://github.com/otcshare/ofs-docs

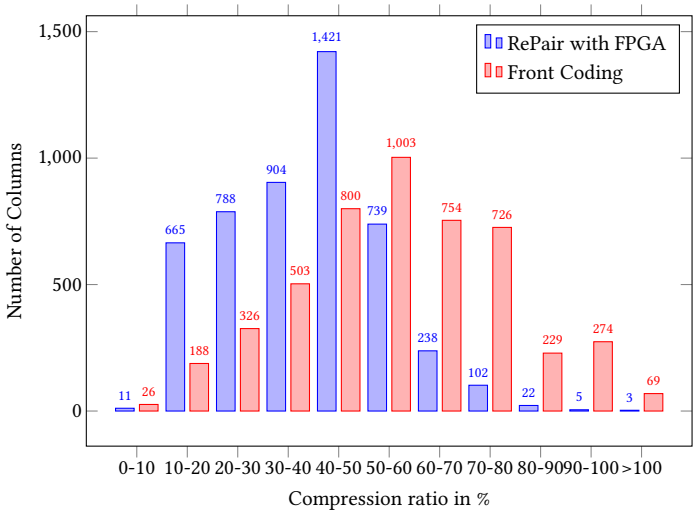

Figure 4: Distribution of compression ratios

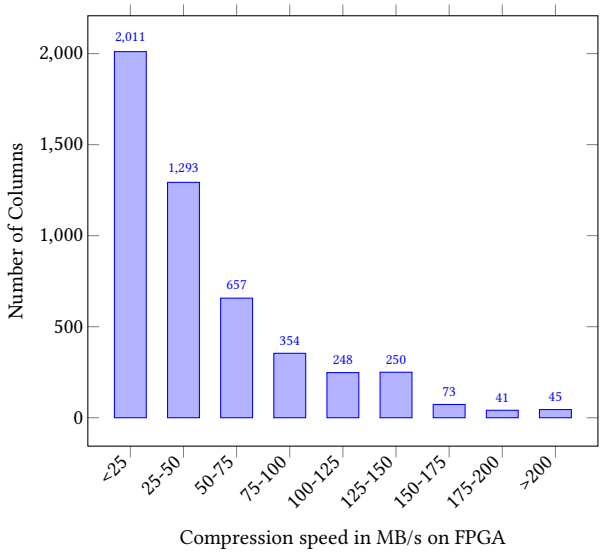

Figure 5: Distribution of RePair speed with FGPA

4 Preliminary Experiments

In this section we present experiments that demonstrate how a disaggregated setup of FPGAs can be highly beneficial for core database workloads. We first confirm earlier experiments [21] on new, real-world datasets that RePair compression indeed can significantly reduce the memory consumption for string dictionaries in SAP HANA. We also show that in the setup of DASH the data transfer from and to the FPGA is not the bottleneck for compute-intensive tasks like RePair compression of string dictionaries. We also identify under which conditions the overheads associated with RePair compression and the DASH architecture do not justify the potential reduction in memory consumption. Finally, we analyze the use of shared FPGAs for a cluster setup where multiple databases delegate string dictionary compression to the FPGA.

Setup. To run the FPGA workload on the server a Docker container was created with CentOs 7.6, IOFS runtime and a FPGA bit-stream for the compression. As programming language this setup uses OpenCL™ version 20.4. In the hardware service worker, a simple function scheduler queues the compression requests (i. e., hardware tasks) of different dictionaries from different HANA instances.

We extracted 4923 columns of productive SAP HANA systems with data type (N)VARCHAR and CLOB (89 columns). Overall, the uncompressed strings of these column dictionaries consume 47GB memory; after front-coding – the default compression technique used for string dictionaries in SAP HANA - these string dictionaries consume 22GB of main memory. In total the data of the analyzed systems consumed 166GB of memory. We performed RePair compression or regular front coding in SAP HANA on every column and collected the runtime. For FPGA, the uncompressed dictionary resides on host memory and was also transferred to the FPGA.

RePair Compression Performance. In our first experiment we confirm our earlier results [21] that RePair compression indeed results in a better compression ratio than plain front coding as it is used in SAP HANA in most cases today. Therefore, we executed both alternatives on string columns extracted from various productive SAP HANA systems for SAP applications like SAP S/4 HANA.

In Fig. 4 we present the compression ratio achieved for these columns grouped by the resulting compression ratio (lower is better). Evidently, for the majority of columns the compression ratio using RePair is between 10% and 60%, and significantly fewer columns achieve this compression ratio for front coding. Also less than 7% of the columns compressed with RePair have compression ratios above 60%, while this is the case for more than 40% of the columns with front coding. In summary, RePair compression reduces the size of the string dictionaries from 22GB to 12.33GB.

In a second experiment we analyze if the network transfer needed in the DASH architecture becomes a performance bottleneck. In Fig. 5 we report the compression throughput as uncompressed size of a column in MB divided by the time required for RePair compression in seconds. The compression time also includes the data transfer from the host to the FPGA and back as well as overheads for invoking the FPGA logic. It can be seen that for ca. 40% of the columns the throughput is below 25 MB/s and for 80% of the columns below 75 MB/s. Only for 45 columns the compression speed exceeds 200 MB/s. From these numbers we can conclude that network transfer is likely not a bottleneck for RePair compression using the DASH architecture. The results also highlight that RePair compression is a heavy, compute-bound operation.

RePair Compression using DASH. Considering that RePair compression in the DASH architecture adds latency and may not even improve the compression ratio for some columns, we analyze in Fig. 6 when RePair compression is expected to be beneficial. The chart shows the compression ratio grouped by the average memory size per element in the string dictionary after applying front coding. For example, the left-most group indicates that an average entry in the front-coded string dictionary consumes less than 2 Bytes. Our results confirm the intuition that applying RePair compression on front-coded string dictionaries as a post-processing

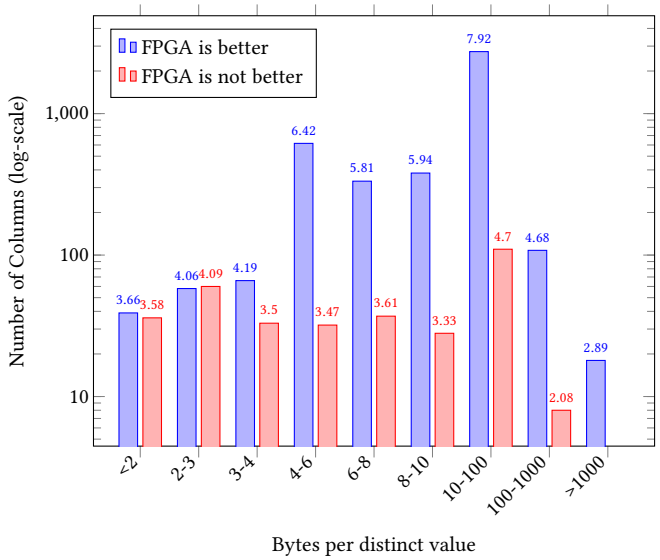

Figure 6: RePair with DASH is beneficial when the average front-coded entry in the string dictionary exceeds three Bytes

step does not improve the compression ratio anymore when the elements in the dictionary are short. Concretely, as soon as the memory size per element exceeds 3 Bytes, the Re-Pair algorithm improves the compression ratio in most cases. Consequently, we suggest to use Re-Pair compression when the average memory size per dictionary element after front coding is greater than 3 Bytes. This can be checked before transferring the data from the HANA instance to the FPGA. Using the heuristic slightly reduces the memory consumption of all string dictionaries from 12.33GB (always using RePair compression) to 12.16GB (using the heuristics).

Cost analysis of DASH. We conclude our experiments with a cost-analysis. In our experiments above we learned that RePair front-coding reduces the memory consumption of string dictionaries by 50%, and RePair compression of the remaining uncompressed suffix improves the compression ratio by another 50%. Furthermore, at 100MB/s we can compress about 8.6TB of string dictionaries per day. Assuming two merge operations per day this corresponds to 4.3TB of uncompressed string dictionary data that could be compressed to 2.3TB of front-coded dictionaries on the CPU and to 1.075TB of RePair-compressed dictionary data using RePair compression and DASH (i. e., 1.075TB main memory savings). Using the public HANA Cloud capacity unit estimator[8] we see that one GB of DRAM is charged at 0.023 CU/GB/hour, i. e., 18050 CU for 1.075TB of DRAM at 730 hours usage per month. At 0.80€/CU this maps to 14,440€/month for provisioning the DRAM of string dictionaries. We compare this to a f1.2xlarge reserved instance at AWS [6] which is charged at 1,06USD/hour (i. e., 671€ per month). The total saving of RePair compression in this setup using an F1 instance at Amazon would be 13.769 € per month per FPGA. We can assume linear scaling of these savings as the data sizes increase.

[8]HANA Cloud capacity unit estimator, visited 12/22: https://bit.ly/3OSyml8

From our experiments we conclude, that technically but also economically the disaggregated accelerator architecture of DASH can be highly beneficial for compute-intensive data management operations.

5 Research Agenda

In this section, we discuss some opportunities for further research and open research challenges in the context of DASH.

5.1 Cloud Infrastructure and Operation

The scenario outlined in this paper assumes a setup within the same data center infrastructure. While some cloud platform vendors start to offer FPGAs in their compute instances [2, 6] it may still be the case that different FPGAs are needed for the targeted acceleration task. For example, in our setup we use Intel FPGAs which are not yet offered as compute instances at AWS, Azure or GCP. However, using secure connections which can be implemented in the FPGA or in the HW Service Worker used to access the FPGA it should also be feasible to use FPGAs in another data center of another vendor or hosted on premises. In our cost analysis this adds the costs for network outgoing traffic from the cloud platform vendor, but there is still a cost benefit. We also emphasize, that in the DASH architecture the additional latencies of this cross-data center access should be acceptable.

Deploying and operating FPGAs in a public cloud offering is still in its infancy. While the internal use of FPGAs in a data center as done in [5, 26] allows for full control of lifecycle operations like upgrading or undeployment, this is more challenging in a more loosely coupled architecture as in DASH. Further research is needed to understand how fundamental software engineering aspects (e. g., security, testing, debugging, monitoring) apply to large-scale deployment of FPGAs.

The deployment of hardware service workers in DASH offers options to scale the number of deployed hardware accelerators or the number of instances of a hardware-accelerated service to meet the elasticity requirements of the databases in the cluster. The setup also allows to implement high-availability setups where failures of accelerators are transparently handled by redundantly deployed instances of the same hardware-accelerated service. We intend to analyze these use cases as part of our future work.

5.2 Heterogeneous Compute

In next generation data management systems, boundaries of storage, compute and network are more fuzzy and compute is distributed to all of those hierarchies in the form of in-line processing near memory or storage and distributed over multiple nodes. A cloud-native application can be assumed to have different services running on different underlying technologies such as CPU, GPU, FPGAs or even smart NICs. We presented the DASH architecture using FPGAs as accelerators for database operations. However the concept can be similarly applied to any other relevant platform. Multiple HW tasks (security, compression, ML etc) can be targeted to execute on the same accelerator device, or with multiple devices.

5.3 Load-Balancing and Data Movement

DASH's architecture offers several options for de-centralized, elastic scaling of hardware components, even scaling them down to

zero. For example, components like the HW Service Worker could also be realized following a Function-as-a-Service pattern. Moreover, the load-balancing dynamics of DASH should be studied in practice, e. g., partial reconfiguration to make hardware tasks available on the FPGA on demand. While in DASH a central scheduler assigns work to the hardware devices, the scheduling could also be delegated further to the accelerators when they communicate directly, e. g., via network interface for FPGA or NVLink for GPUs.

We expect CXL to provide a standardized way for coherent memory transfers and accelerator-to-accelerator connectivity in such heterogeneous systems. More sophisticated scheduler schemes in DASH's HW Task Scheduler could manage an orchestration of hardware services which communicate directly by wiring source and target devices directly, e. g., via CXL, instead of returning data to the storage and the next task reads it from there, while tasks normally report back to the scheduler. This would enable the DASH architecture to further offload work from the CPU because the communication with the host is further reduced.

We assume DASH to be used by multiple databases for which the asynchronous scheduling might result in a large number of enqueued hardware tasks. When having tasks with specified service-level agreements, e. g., scheduling strategies with priorities or identification and handling of long-running tasks should be considered.

6 Conclusion

While previous research has demonstrated the benefit of FPGAs in the context of databases, they have not yet enjoyed wide-spread deployments. We argue that the tight integration of FPGAs with the database engine is one of main limiting factors of a broader adoption of FPGAs in this area.

To address this issue, we propose DASH as a disaggregated architecture that exposes FPGAs as a service for compute-intensive data management operations. While we illustrate the benefits of this architecture using string dictionary compression with RePair compression, we argue that this setup can be applied to a wide range of similar scenarios. For example, in [13] we show how JSON parsing can be accelerated using FPGAs, and this approach can be used as a JSON validation service that can reject invalid JSON documents without investing expensive CPU cycles on the database server. Similarly, FPGA-accelerated machine learning operations [3] deployed in a separate, shared process can be used by multiple database instances. Such shared accelerated database services become a viable deployment option in the cloud to accelerate or offload compute intensive workload. Along this trend, hardware docomposition could complement the effort to decompose database operations into hardware microservices and become a logical next step to schedule those microservices on heterogenous hardware devices.

Acknowledgments

We thank Tao Shen for supporting the experiments, and Andre Russ, Vincent Riesop, Malte Janduda from the SAP Gardner team for the OS support with IOFS. We would like to thank Lenovo for providing the ThinkSystem SR670 V2 server hardware for the FPGA studies and the technical support from Trick Hartman, Arne Wolf, Ajay Dholakia and Pravin Patel.

References

[1] D. Abadi, A. Ailamaki, D. Andersen, P. Bailis, M. Balazinska, P. A. Bernstein, P. Boncz, S. Chaudhuri, A. Cheung, A. Doan, L. Dong, M. J. Franklin, J. Freire, A. Halevy, J. M. Hellerstein, S. Idreos, D. Kossmann, T. Kraska, S. Krishnamurthy, V. Markl, S. Melnik, T. Milo, C. Mohan, T. Neumann, B. C. Ooi, F. Ozcan, J. Patel, A. Pavlo, R. Popa, R. Ramakrishnan, C. Re, M. Stonebraker, and D. Suciu. The seattle report on database research. *Commun. ACM*, 65(8):72–79, 2022.

[2] Alibaba. Github aliyun-faas-agilex. https://github.com/aliyun/aliyun-faas-agilex, 2022. Accessed: 2022-12-07.

[3] G. Alonso, Z. István, K. Kara, M. Owaida, and D. Sidler. doppioDB 1.0: Machine learning inside a relational engine. *IEEE Data Eng. Bull.*, 42(2):19–31, 2019.

[4] P. Antonopoulos, A. Budovski, C. Diaconu, A. H. Saenz, J. Hu, H. Kodavalla, D. Kossmann, S. Lingam, U. F. Minhas, N. Prakash, V. Purohit, H. Qu, C. S. Ravella, K. Reisteter, S. Shrotri, D. Tang, and V. Wakade. Socrates: The new SQL server in the cloud. In *SIGMOD*, pages 1743–1756. ACM, 2019.

[5] N. Armenatzoglou, S. Basu, N. Bhanoori, M. Cai, N. Chainani, K. Chinta, V. Govindaraju, T. J. Green, M. Gupta, S. Hillig, E. Hotinger, Y. Leshinksy, J. Liang, M. McCreedy, F. Nagel, I. Pandis, P. Parchas, R. Pathak, O. Polychroniou, F. Rahman, G. Saxena, G. Soundararajan, S. Subramanian, and D. Terry. Amazon Redshift re-invented. In *SIGMOD*, page 2205–2217. ACM, 2022.

[6] AWS. Amazon EC2 F1-instances. https://aws.amazon.com/de/ec2/instance-types/f1, 2022. Accessed: 2022-08-01.

[7] N. Boeschen and C. Binnig. Gacco - A gpu-accelerated OLTP DBMS. In *SIGMOD Conference*, pages 1003–1016. ACM, 2022.

[8] X. Chen, Y. Chen, R. Bajaj, J. He, B. He, W. Wong, and D. Chen. Is FPGA useful for hash joins? In *CIDR*, 2020.

[9] M. Chiosa, F. Maschi, I. Müller, G. Alonso, and N. May. Hardware acceleration of compression and encryption in SAP HANA. *Proc. VLDB Endow.*, 2022.

[10] J. Dann, D. Ritter, and H. Fröning. Demystifying memory access patterns of FPGA-based graph processing accelerators. In V. Kalavri and N. Yakovets, editors, *GRADES-NDA*, pages 3:1–3:10. ACM, 2021.

[11] J. Dann, D. Ritter, and H. Fröning. GraphScale: Scalable bandwidth-efficient graph processing on FPGAs. *CoRR*, abs/2206.08432, 2022.

[12] J. Dann, D. Ritter, and H. Fröning. Non-relational databases on FPGAs: Survey, design decisions, challenges. *ACM Comput. Surv.*, oct 2022. Just Accepted.

[13] J. Dann, R. Wagner, D. Ritter, C. Faerber, and H. Fröning. Pipejson: Parsing JSON at line speed on FPGAs. In *DaMoN*, pages 3:1–3:7. ACM, 2022.

[14] Z. F. Eryilmaz, A. Kakaraparthy, J. M. Patel, R. Sen, and K. Park. FPGA for aggregate processing: The good, the bad, and the ugly. In *ICDE 2021*, pages 1044–1055. IEEE, 2021.

[15] J. Fang, Y. T. B. Mulder, J. Hidders, J. Lee, and H. P. Hofstee. In-memory database acceleration on FPGAs: A survey. *The VLDB Journal*, 29(1):33–59, 2020.

[16] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The SAP HANA database – an architecture overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012.

[17] G. Huang, X. Cheng, J. Wang, Y. Wang, D. He, T. Zhang, F. Li, S. Wang, W. Cao, and Q. Li. X-Engine: An optimized storage engine for large-scale E-Commerce transaction processing. In *SIGMOD*, page 651–665. ACM, 2019.

[18] Z. István, D. Sidler, and G. Alonso. Building a distributed key-value store with fpga-based microservers. In *FPL*, pages 1–1, 2015.

[19] K. Kim, R. Johnson, and I. Pandis. Bionicdb: Fast and power-efficient OLTP on FPGA. In *EDBT*, pages 301–312. OpenProceedings.org, 2019.

[20] R. Lasch, M. Moghaddamfar, N. May, S. S. Demirsoy, C. Färber, and K. Sattler. Bandwidth-optimal relational joins on FPGAs. In *EDBT*, pages 1:27–1:39, 2022.

[21] R. Lasch, I. Oukid, R. Dementiev, N. May, S. S. Demirsoy, and K. Sattler. Faster & strong: string dictionary compression using sampling and fast vectorized decompression. *VLDB J.*, 29(6):1263–1285, 2020.

[22] M. Moghaddamfar, C. Färber, W. Lehner, N. May, and A. Kumar. Resource-efficient database query processing on FPGAs. In D. Porobic and S. Blanas, editors, *DaMoN 2021*, pages 4:1–4:8. ACM, 2021.

[23] R. Mueller, J. Teubner, and G. Alonso. Streams on wires: a query compiler for FPGAs. *Proceedings of the VLDB Endowment*, 2(1):229–240, 2009.

[24] V. R. Narasayya and S. Chaudhuri. Cloud data services: Workloads, architectures and multi-tenancy. *Foundations and Trends in Databases*, 10(1):1–107, 2021.

[25] J. Paul, S. Lu, and B. He. Database systems on GPUs. *Foundations and Trends in Databases*, 11(1):1–108, 2021.

[26] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, E. Peterson, A. Smith, J. Thong, P. Y. Xiao, D. Burger, J. Larus, G. P. Gopal, and S. Pope. A reconfigurable fabric for accelerating large-scale datacenter services. In *ISCA*, pages 13–24. IEEE Press, 2014.

[27] D. Ritter, J. Dann, N. May, and S. Rinderle-Ma. Hardware accelerated application integration processing: Industry paper. In *DEBS*, page 215–226. ACM, 2017.

[28] L. Woods, Z. István, and G. Alonso. Ibex: An intelligent storage engine with support for advanced SQL offloading. *Proc. VLDB Endow.*, 7(11):963–974, 2014.