SKV: A SmartNIC-Offloaded Distributed Key-Value Store

Shangyi Sun Rui Zhang Ming Yan Jie Wu*
School of Computer Science
Fudan University
Shanghai, China
{shangyisun21, zhangrui21}@m.fudan.edu.cn, {myan,jwu}@fudan.edu.cn

2022 IEEE International Conference on Cluster Computing (CLUSTER) | 978-1-6654-9856-2/22/$31.00 ©2022 IEEE | DOI: 10.1109/CLUSTER51413.2022.00016

***Abstract*—In data center networks, applications such as distributed key-value stores consume a lot of CPU resources. The performance of the entire system drops significantly under heavy load conditions. In order to improve the performance of key-value stores, many existing studies use RDMA (Remote Direct Memory Access) to reduce the communication overhead. However, RDMA primitives can only offload simple operations to the NIC, such as reading and writing remote memory. With the emergence of new hardware like SmartNICs, we consider whether we can offload more complex operations in distributed key-value stores to SmartNICs to reduce the load on CPU. In this paper we present SKV, a SmartNIC-offloaded distributed key-value store. In order to make full use of the offload ability of the SmartNIC, we make a detailed analysis on the characteristic and architecture of SmartNICs and distributed key-value stores. SKV offloads operations such as data replication to the SmartNIC. We design a new replication mechanism, which enables the server to separate background processing from the interaction with clients in the front. We implement SKV on the Mellanox BlueField SmartNIC. Our evaluations show that SKV improves the overall throughput by 14% and reduces latency by 21% compared with baseline.**

***Index Terms*—SmartNIC, distributed key-value store**

I. Introduction

With the increasing demand for data centers to collect and analyze massive data, NoSQL (Not Only Structured Query Language) database system has been widely used in industry [1]. As a typical NoSQL, key-value store is widely used in network services, big data processing and other fields due to its ease of use and high performance. A key-value store does not need to maintain complex relationships between data tables like the traditional way, so it can store data flexibly. There are many widely used key-value stores, such as Redis [2], Memcached [3] and LevelDB [4]. Some of these key-value stores are based on hash tables and others are based on Log Structured Merge (LSM) trees, and each has its advantages and disadvantages. However, the performance of these key-value stores far exceeds that of traditional relational databases in general [5]. Therefore, key-value stores are becoming more and more popular.

For key-value store clients, the communication between hosts often brings a lot of overhead. Since the read/write speed of key-value pairs in memory is especially fast, the communication process of the traditional network protocol stack becomes the performance bottleneck of the whole system under high load. To improve overall performance, many studies [6]–[10] have used RDMA as the network layer of key-value stores. There are also researches, such as Nova-LSM [11], using RDMA for a distributed key-value store based on LSM tree. By using RDMA instead of TCP, these efforts greatly reduce the communication overhead between hosts, thereby improving overall performance.

However, using RDMA to accelerate network communication can only offload simple operations to the RDMA NIC including directly reading and writing remote memory, which does not allow the NIC to offload more complicated tasks. Many complex distributed key-value store operations still consume a lot of host CPU resources. For example, when a key-value store master server is equipped with multiple slaves, the master needs to replicate each received `SET` command to all slaves. When the number of slaves is large, the replication operation on the master node will consume a lot of CPU resources. To offload these operations to the NIC, other methods should be used.

In recent years, the emergence of SmartNICs has given us new inspiration. At present, many researches focus on FPGA-based SmartNICs, which consolidate the application logic into the programmable logic blocks of the FPGA NIC. This approach is only suitable for offloading parallel and simple tasks. To offload more complex tasks, multi-core system-on-chip (SoC) SmartNICs are required. Nowadays, many manufacturers have released different types of multi-core SoC SmartNICs, such as Mellanox Bluefield [12], Marvell LiquidIO [13], Broadcom Stingray [14] and Agilio Netronome [15]. This new hardware not only supports sending and receiving data packets like traditional NICs but also contains rich computing resources. With the increase of the burden on servers, the computing power of host CPU has gradually become a very scarce resource in the servers in large data centers. SmartNICs are just the thing to mitigate the gap between the increasing network bandwidth and stagnant CPU computing power [16].

Due to these advantages, multi-core SoC SmartNICs have been used in many scenarios. Xenic [17] uses SmartNICs for distributed transaction systems. ipipe [18] designs a framework to dynamically schedule tasks between SmartNICs and hosts according to system load. E3 [19] offloads microservices to the SmartNIC. LineFS [20] uses SmartNICs to accelerate dis-

*Jie Wu is the corresponding author.

2168-9253/22/$31.00 ©2022 IEEE
DOI 10.1109/CLUSTER51413.2022.00016

tributed file systems. However, none of these existing studies focuses on using SmartNIC for distributed key-value stores.

In this paper, we propose SKV, a high-performance distributed key-value store accelerated with SmartNIC. SKV makes full use of the computing power of the SmartNIC and offloads some of the components in a distributed key-value store to the SmartNIC. We use Redis, a very popular key-value storage system, as a building block. As a widely used storage system in industry, Redis uses hash table as a storage structure, which has high insertion and query performance. This paper mainly optimizes the architecture of Redis in its distributed mode.

However, offloading a distributed key-value store to the SmartNIC is a very challenging task. Although the processor cores on the SmartNIC have more functions than the processing unit of the RDMA NIC, their processing speed is much worse than that of the host CPU cores. Therefore, we cannot simply offload the entire key-value store to the SmartNIC. The components of the distributed key-value store must be carefully analyzed and only the appropriate parts should be offloaded to the SmartNIC. In order to make full use of the advantages of the SmartNIC and avoid its shortcomings, we carefully choose the offloading scheme. We offload part of the distributed key-value store to SmartNIC and design a new replication mechanism, allowing the host and the SmartNIC to work in parallel.

In this paper, we make the following contributions:

- We summarize the types of existing SmartNICs, and describe the internal structure of multi-core SoC SmartNICs. In order to take full advantage of SmartNIC resources when offloading a distributed key-value store, we measure the basic characteristics of the SmartNIC. At the same time, we analyze the workflow of the distributed key-value store and point out some problems that need to be solved.
- To address existing problems, we propose SKV, a high-performance distributed key-value store accelerated with SmartNIC. We point out the challenges in offloading the system to SmartNIC and propose how to offload appropriate parts of the distributed key-value store based on our characterization on SmartNICs.
- We implement SKV based on Redis, a popular distributed key-value store. Our implementation uses a Mellanox BlueField SmartNIC. As an off-path SmartNIC, BlueField contains abundant computing resources, which can offload complex tasks flexibly. We offload some components of the distributed key-value store to SmartNIC, including data replication and failure detection.
- We deploy SKV on the host and SmartNIC and compare the performance of SKV with Redis. In our experiments on the overall performance, SKV reduces the latency by 21%, while increasing the throughput by 14%.

The next few sections are organized like this. Section II introduces the background of SKV, including the characteristics of SmartNICs and the architecture of typical distributed key-value stores. Section III puts forward the design goal of SKV, and introduces the design of each part of SKV in detail. Section IV describes the implementation of SKV. Section V shows and analyzes the performance of SKV compared with baseline. Section VI summarizes some existing work on SmartNICs and key-value stores. Section VII comes to the conclusion of this paper and shows our expectation for future work.

II. Background

We first summarize the types and characteristics of SmartNICs and point out the implications these characteristics give us. Then, as a basis for offloading the appropriate task to SmartNIC, we use Redis as an example to analyze the architecture of a distributed key-value store.

A. Multi-Core SoC SmartNICs

SmartNIC is a new type of network device, which can expand the computing power of the server at a low cost. It contains computing units and on-board memory. SmartNICs are either based on FPGA or based on multi-core SoC. Here, we focus on SmartNICs based on multi-core SoC. These SmartNICs can be divided into two types: on-path and off-path.

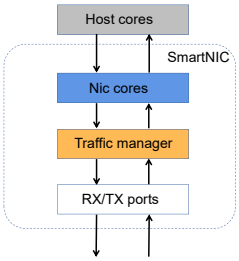


Fig. 1. On-path SmartNIC.

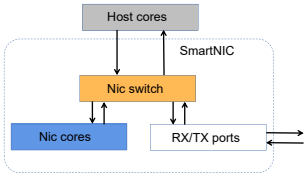


Fig. 2. Off-path SmartNIC.

1) On-path SmartNICs: As shown in Figure 1, on-path SmartNIC refers to the fact that the core of the SmartNIC is located on the path of data transmission, and it can manipulate each incoming and outgoing packet (e.g., Marvell LiquidIO and Netronome Agilio). This type of SmartNIC provides low-level hardware interfaces for packet processing, requiring all network traffic to and from the host to be handled by the NIC core. The main low-level hardware interfaces provided by on-path SmartNICs include Ethernet RX/TX queues, packet buffer management, packet scheduling and ordering modules, and PCIe DMA engines. After specifying packet processing logic, many functions can be offloaded to the SmartNIC. By utilizing on-board DRAM to store data, unnecessary PCIe accesses can be avoided when processing requests from remote hosts. In addition, on-path SmartNICs can access host memory by issuing PCIe DMA requests, which off-path SmartNICs cannot.

The advantage of on-path SmartNIC is that it can directly operate on incoming and outgoing data packets, and it can provide high-throughput and low-latency services. But using on-path SmartNIC can only operate on data packets through

low-level interfaces, which lacks flexibility. Therefore, this kind of SmartNICs are suitable for offloading tasks with relatively simple and parallel logic, but cannot be used for offloading complex tasks.

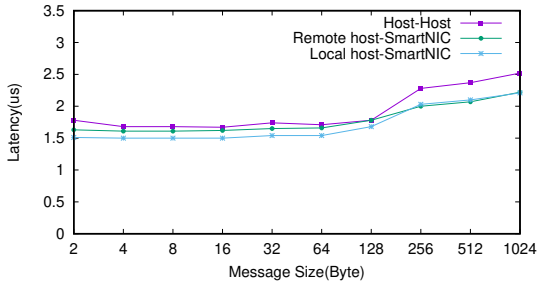


Fig. 3. Latency of RDMA WRITE.

2) Off-path SmartNICs: Off-path SmartNICs include Broadcom StingRay and Mellanox Bluefield. In addition to components such as on-board memory and multi-core processors, this type of SmartNIC contains a NIC switch as shown in Figure 2. The NIC switch can direct traffic to the host or the core on the SmartNIC according to the rules on it. An off-path SmartNIC typically runs with a full network stack, but it lacks low-level interface for manipulating packets directly. Moreover, communication between the SmartNIC and the host is inefficient due to the complete network stack on SmartNIC. We evaluate the `RDMA WRITE` latency between two hosts, from remote host to local SmartNIC, and from local host to local SmartNIC, respectively. Figure 3 illustrates the results of the experiments under these three conditions. The SmartNIC is just like a separated endpoint in the network, as the write latency from host to SmartNIC is only a little lower than the write latency between two hosts. Therefore, this overhead cannot be ignored when using an off-path SmartNIC to offload tasks.

However, the design of off-path architecture has obvious advantages: (1) the data path is loosely coupled with the processors on the SmartNIC, and data flow which does not need special processing can bypass the cores on SmartNIC with the help of NIC switch; (2) processors under this architecture tend to have higher programming flexibility, which makes them more suited for offloading complex tasks. Therefore, this paper uses Mellanox Bluefield as an off-path SmartNIC to offload a distributed key-value store.

B. Redis

Redis is an open-source, high-performance key-value database. Redis supports data persistence by constantly saving data in memory to disk, so the data can be reloaded for use upon restart. Redis not only supports simple string data, but also provides the storage of data structures such as list, set, and hash. In addition, Redis supports redundant storage of data, that is, data backup in master-slave mode.

Redis is an event-driven storage system and needs to handle both time events and file events, as shown in Figure 4. A Redis server connects to clients (or other Redis servers) through sockets and file events are the abstraction of socket operations of the server. Some operations in the Redis server (such as cleaning up expired keys) need to be performed at a given point in time, and time events are the server's abstraction of these operations.

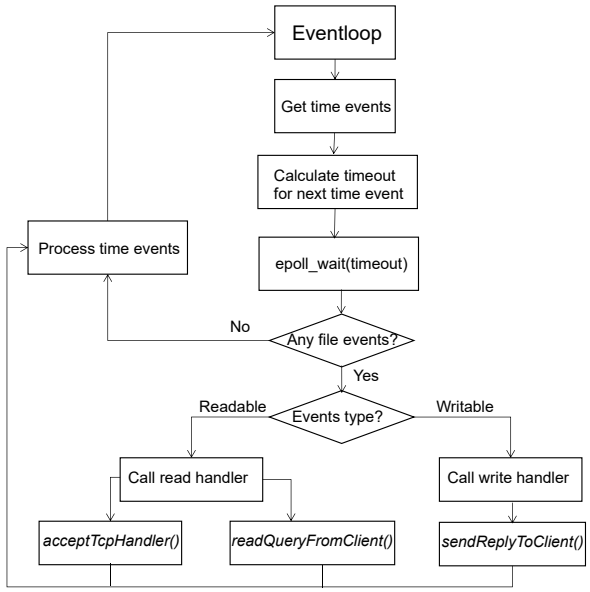


Fig. 4. Redis workflow.

1) Redis Basic Workflow: A Redis server handles requests from clients like this.

Server initialization. At startup, the server needs to create a server object and set default values for its properties. Then it starts an event loop and listens on the specified port. When a new client tries to connect with the server, the corresponding callback function is called to accept it. The server then gets a file descriptor from the accepted connection and listens on it. Then the listened port will be wrapped as a file event object and registered with the event loop, so the event loop can capture each read or write event and call the corresponding callback function. For each client connection, the server creates a corresponding client object for it, which contains the input/output buffer, the state of the client and other information.

Accepts and processes the commands. After the client sends the command, the server socket becomes readable, and the previously registered callback function `readQueryFromClient()` is triggered. The server reads the command from the socket and puts it in the query buffer of the client object. The number of command bytes that can be read is limited. If the upper limit is exceeded, the server will refuse the connection and release the client object. After the command is read, it needs to be parsed according to Redis communication protocol. After parsing, the server does some regular checks, such as whether the command exists and whether the number of parameters is valid. Then it calls the

function that actually executes the command.

Replies to the client. If the command is executed successfully, an "OK" string is written to the reply buffer of the client through the `addReply()` method. At the same time, the client is added to a list that contains all the clients with pending replies. When the file descriptor becomes writable, the write callback function `sendReplyToClient()` is triggered. All the replies in the reply buffer of the clients in the list will be written to the socket. Finally, all the clients receive their replies.

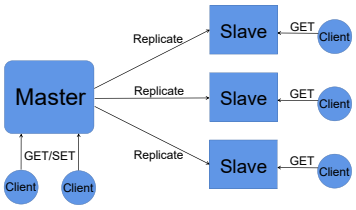


Fig. 5. Master-slave mode of Redis.

2) Master-slave Replication: In order to increase the scalability and availability of the system, Redis provides a master-slave mode [21], which is now widely used in industry. Master-slave mode provides multiple slave servers for one master server as shown in Figure 5. In this mode, the master node can accept both read and write requests, while the slave node can only be used for reading. In order to keep all slave nodes consistent with the state of the master node, Redis needs to replicate data from the master node to the slave nodes. The main process of replicating is as follows.

1) A slave sends a SYNC(synchronization) command to the master.
2) After the master receives the SYNC command, it starts a child process to persist all the data and generate an RDB file.
3) When the master finishes executing the persisting process, it immediately sends the RDB file to the slave, and the slave will receive and load the file.
4) All write commands received by the master during the persisting process are stored in a backlog buffer. The master sends these commands to the slave for execution.
5) After the above processing, each write command executed by the master is sent to all of its slaves.

Such a mechanism makes it possible to prepare multiple copies of a single piece of data and the server's ability to accept requests is extended, but it often brings huge burden to the server CPU.

C. Challenges of Offloading

The emergence of SmartNIC gives us the inspiration to use it to improve the performance of distributed key-value stores. However, offloading a distributed key-value store to SmartNIC is a very challenging task. Here we summarize several main challenges.

- **The complexity of distributed key-value stores.** A distributed key-value store is a very complex system, which includes many functions such as data reading and writing, persistence, and master-slave replication. Different functional modules have different requirements for CPU and host memory resources, and many components are closely coupled with each other. We must sort out these relationships before proceeding with the subsequent design.
- **Limited resources on SmartNIC.** The performance of the cores on the SmartNIC is much weaker than that of the host cores. When executing the same task, the SmartNIC will perform much slower than a host [22]. What's more, the size of DRAM on the SmartNIC is much smaller than that of the host, so it can't store as much data in memory as the host. Therefore, it is not feasible to simply put all the key-value store operations onto the SmartNIC.
- **High communication overhead between SmartNIC and host.** The off-path SmartNIC can dynamically decide whether to forward the received data packet directly to the host or hand it over to the core of the SmartNIC for further processing. However, the complete network stack on the SmartNIC introduces a huge overhead to the processing of data packets. This results in a very large communication latency between the host and the SmartNIC. Therefore, we have to carefully design the offloading strategy to avoid this huge overhead.

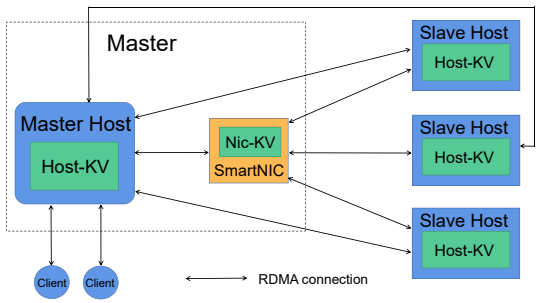


Fig. 6. Overview of SKV.

III. Design

Figure 6 shows the overview of SKV. SKV is a distributed key-value store, which equips the master server with multiple slaves. A SmartNIC is installed on the master node. SKV carefully spreads the components of the distributed key-value store across the host and the SmartNIC. SKV nodes consist of two components: Host-KV and Nic-KV. Host-KV is located on the host while Nic-KV is located on SmartNIC, which contains the offloaded components.

A. Design Goals

We hope SKV to achieve the following goals by leveraging the SmartNIC.

- **Low latency and high throughput.** In data center networks, various upper-layer applications often have strict requirements for the latency of key-value store. For example, some tasks may remain blocked until the value of a key is retrieved or written. Therefore, the latency of the key-value store can significantly affect the overall performance of the upper-layer application. In addition, throughput is also important. The greater throughput means more requests can be processed in the same time, increasing the efficiency of the system. Therefore, we try to optimize both latency and throughput in the following design.
- **Low CPU consumption.** If a distributed key-value store uses multi-threading to improve its performance, it often consumes a lot of host CPU cores. Especially when the CPU competition caused by multi-threading occurs, the performance of the whole application will drop sharply. Therefore, we urgently need an offloading method that can use the CPU efficiently. We hope to use single thread on host to reduce the number of occupied cores while maintaining high performance.
- **Make full use of the SmartNIC.** To reduce the load on the host as much as possible, SKV needs to maximize the use of data path processing capability of the cores on SmartNIC. Hence, we should choose the most appropriate tasks to offload. At the same time, care must be taken not to overload the SmartNIC, and the overall performance of the entire distributed key-value store must be considered.

B. Utilizing RDMA

In a distributed key-value store, a large portion of the overhead of read and write operations comes from communication. In addition to the communication between the client and the server, the data replication between the server nodes also brings huge network overhead. The original Redis uses TCP for inter-host communication. Kernel network stacks such as TCP typically have end-to-end latency of hundreds of microseconds. This is because in such a transmission process, each data packet undergoes multiple memory copies between user space, kernel space and the NIC. In addition, the overhead of context switching between the application process and the kernel thread, and the encapsulation and parsing of the packet header caused by the protocol stack also introduces non-negligible CPU overhead. Therefore, we redesign the network layer of Redis in order to reduce the overhead caused by network transmission. We replace TCP with RDMA. In SKV, all the hosts and the SmartNIC communicate with each other using RDMA.

Due to the complexity of programming with native RDMA primitives, there are many RDMA frameworks, including rsocket, MPI and SDP. Although these libraries bring great convenience to programming, they fail to explore the full advantages of the high performance of RDMA. Therefore, we use native RDMA verbs to implement the network communication module of SKV.

SKV uses RDMA_CM to establish the connection between two nodes. First, the server listens on the specified RDMA port. When a client initiates a connection request, the server calls the registered callback function to accept the request from the client. After that, the client and the server exchange their Memory Region (MR) information using `SEND/RECV` primitives. Then, the SKV server can get prepared to respond to the request from the client. The client sends commands and the server replies using `WRITE_WITH_IMM`, which notifies the receiver when its memory is written. When the receive buffer is full, the MR needs to be registered again. After sending the MR information to the other node with the `SEND` operation, the previous communication process continues.

To reduce CPU utilization, we avoid constantly polling the Completion Queue (CQ). Instead, we use the completion event channel to pass notifications for upcoming work completion in CQ. When the `ibv_get_cq_event()` method attached to the event channel is called, it will keep blocked until upcoming work completion events are triggered. We use the `ibv_ack_cq_event()` method to acknowledge the events and the notification of the following work completion is requested by the `ibv_req_notify_cq()` method.

C. Replication Offloading

As Redis master replicates data asynchronously, master-slave replication can be separated from other Redis components. Therefore, replication may be an appropriate object to offload. In master-slave mode, after the replication process has reached a steady state, the master node in the distributed key-value store needs to send all received write executed to all slaves. If there are a large number of slave nodes, replicating commands to all slave nodes will greatly affect the performance of the entire system. We tested the situation of equipping a master with three slaves in the case of using RDMA to accelerate, and the results are shown in Figure 7. Compared with the setting of no slave node, both average latency and 99% tail latency of executing `SET` commands on the master increase. In particular, the tail latency increases by more than 25%. In addition, the throughput also drops significantly in our experiments. This is because in the implementation of Redis, `SET` commands received by the master are sent to all slaves one by one to perform the replication operation, and this consumes a lot of CPU cycles. Only after that, the master sends the results of executing the commands to the client. To alleviate this problem, the master host can choose to use multi-threading to send data from the master node to the slave nodes and clients. However, in this situation, multi-threading consumes a large number of host CPU cores. At the same time, the overhead caused by thread switching reduces the per-core throughput. In order to save the host CPU resources and improve the system performance, we focus on offloading the replication operation to the SmartNIC.

The replication process is divided into two phases, namely the initial synchronization phase and the steady-state replication phase. The overhead of the steady-state replication phase significantly affects the performance of the whole distributed

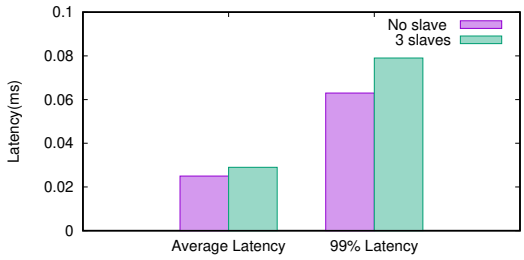


Fig. 7. The performance degradation with multiple slaves.

key-value store, so in SKV the steady-state replication operation is completely offloaded to SmartNIC.

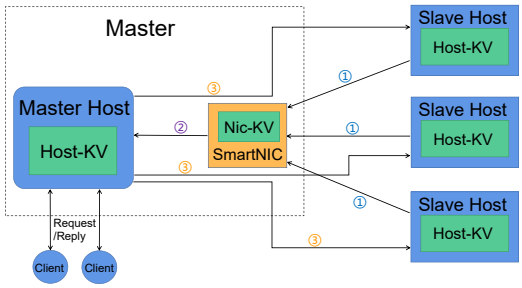


Fig. 8. Initial synchronization phase.

Initial synchronization phase. Figure 8 presents the initial synchronization phase. When a `SLAVEOF` command is executed on the slave node, the slave records the address and port of the master node. Host-KV on the slave checks whether there is master node information at every certain interval. After the master information is discovered, the slave sends an initial synchronization request to the SmartNIC on the master node ①. This request contains its own replication ID, replication offset and the address and port number of the master. For each new slave, Nic-KV creates an object of type client for it. In addition, a node list storing the corresponding relationship between the master node and the slave node is maintained on the SmartNIC. After receiving the request from the slave node, the replication status information of the slave node (including replication ID and replication offset) will be stored at the end of this list. After that, Nic-KV notifies the master node that there is a slave node that wants to synchronize with it ②. After the master node learns, it persists all its own key-value data. Then, it establishes an RDMA connection with the slave node, and stores the information of the slave node in its own slave linked list. Then the master node checks the replication offset of the slave node, compares it with its own offset and gets a deviation. If the deviation is zero, it means that the data of the master node and the slave node are exactly the same, so it can directly enter the steady-state replication state. If it is not zero, the master node can figure out the range of data that needs to be replicated to the slave node based on the deviation. If the range is contained in the backlog buffer, the data within the range in the backlog buffer will be sent to the slave node. If not, the master node will send its own data file containing all key-value pairs to the slave node ③. After receiving the data file, the slave loads all the data into its memory. Then, the entire system can enter the steady-state replication phase.

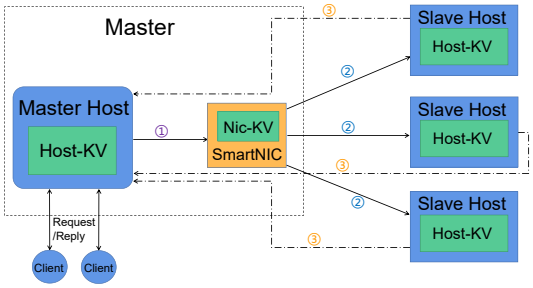


Fig. 9. Steady-state replication phase.

Steady-state replication phase. Figure 9 presents the steady-state replication phase. When the master node receives a command from the client, the Host-KV on it first checks whether the command can change the value of the data in the storage. If it can, for example it is a `SET` command or a `DEL` command, it needs to be replicated to all the slave nodes. Now the Host-KV on master node only needs to send a replication request to the SmartNIC ①. This replication request contains the command to replicate. Nic-KV queries the node list, which stores the previously recorded information of slave nodes. After getting information of all the slave nodes from the node list, this command is written to the send buffer of each slave on the SmartNIC. When the write callback function is called, all commands in send buffers are sent to all slave nodes through `WRITE_WITH_IMM` ②. Every time the slave node receives a new command, it executes the command immediately to ensure that its data is consistent with the master node. For the master node, the entire steady-state replication process is almost carried out in the background. The master does not remain blocked until the slave confirms that replication is finished. In other words, the master node can normally execute commands and return results to clients quickly while the replication is taking place. In addition, at every certain interval, a slave node reports the progress of the data replication to the master node ③. If the progress is too slow, that is, the master node finds that the replication offset of a slave node is too far from its own, it will return an error message to the client. During this phase, Host-KV only uses a single thread, which saves host CPU cores.

When the strong consistency between the master and slave nodes matters, the replication process needs to be accelerated. If the cores on the SmartNIC are idle, SKV can choose to use multiple-threading for the replication operation on the SmartNIC. This makes full use of the processing power of the SmartNIC. We provide a `thread-num` parameter to specify the number of threads on the SmartNIC for replication. Nic-KV checks this parameter and makes sure that the actual number of threads used for replication cannot be greater than

the minimum value of the number of SmartNIC cores and slave nodes. In multi-threaded mode, slave nodes in the node list are evenly spread across multiple threads. However, as replication is done in the background, the speedup of replication cannot improve the latency and throughput of the execution of commands on the master node. Since multi-threading requires a large number of cores on the SmartNIC and brings the overhead of thread switching, the default configuration of SKV is to disable multi-threading.

Nic-KV does not handle requests from clients. Instead, it only interacts with other server nodes. When the SmartNIC receives a request, Nic-KV reads the message in the receive buffer and parses it. After parsing, if it is found that it is sent from a slave node, it means that the message is an initial synchronization request sent from the slave after executing the `SLAVEOF` command. This indicates that the slave node has entered the initial synchronization phase. If it is a request from the master node, it must be a replication request in steady-state replication phase and Nic-KV will perform the corresponding operation in this phase.

D. Failure Detection

One of the primary goals of the master-slave mode of a distributed key-value store is to improve the availability of the whole system. Therefore, node failure should be taken into account. For better specification, here we provide the user with two parameters: `min-slaves` and `waiting-time`. In order to detect the status of the master and slave nodes, Nic-KV sends probe messages to the master and slave nodes every 1 second. When the master node and the slave nodes receive this message, they reply to Nic-KV immediately. If the probe message is sent `waiting-time` ago and Nic-KV does not receive a reply from a node, the node is considered to have crashed. Nic-KV will update the node list on SmartNIC and set an invalid flag for the failed node. If it is found that the master node has crashed, then one of the available slave nodes is selected as the master node. When the original master node is found recovered, Nic-KV let it continue to be the master node and downgrades the previously selected master node to a slave node again. If a slave node is found to have crashed and the number of available slave nodes is less than `min-slaves`, the master node will return an error message to the client when it receives a write command.

IV. Implementation

The SKV we implemented consists of two parts: Host-KV and Nic-KV. Host-KV is deployed on the host and it can be linked to the applications (SKV clients) running on the host. Host-KV is mainly responsible for command execution, data persistence, data organization and interaction with clients. Nic-KV is deployed on the SmartNIC, which is mainly responsible for the offloaded part of the distributed key-value store. Nic-KV communicates with Host-KV via RDMA.

We use Redis as a building block of SKV. Except for the design described in Section III, SKV closely follows the design of Redis. Many components of SKV, such as the implementation of data structures such as dynamic strings, skip tables, sets, the way of data persistence and the hash algorithm, are inherited from Redis. Some other implementation decisions in SKV are shown as follows.

A. Data Storage

In previous works such as Xenic [17] and Direct-KV [23], the SmartNIC stores a portion of data, which is the most frequently accessed. If the data requested by the client is stored on the SmartNIC, the host memory does not need to be accessed. This mechanism alleviates the burden of processing read requests on the host and reduces the PCIe latency from the SmartNIC to host memory during the communication process. However, these works leverage on-path multi-core SoC SmartNIC and FPGA-based SmartNIC, whose characteristics are far from the off-path SmartNIC used in this paper. If SKV follows this idea, the latency of accessing data will increase significantly due to the weaker processors and relatively larger RDMA latency of the off-path SmartNIC. Therefore, SKV stores all key-value pairs on the host, and Nic-KV is only used for cooperating between nodes and offloading some operations of the distributed key-value store.

B. Nic-KV Implementation

When Nic-KV starts, it enters the main loop. In order to handle multiple concurrent requests, Nic-SKV uses epoll IO multiplexing as a network IO model. So Nic-KV needs to create an epoll instance and link it to the main loop. Each RDMA connection corresponds to a file descriptor, and read/write event handler function is bound to each file descriptor. In the process of master-slave replication, all the host-KVs of the master node and the slave nodes need to establish a connection with Nic-KV. Nic-KV calls the registered connection handler to accept connection requests from the master node or slave node, and creates a client object for each connected node. Here, both the master node and the slave node are regarded as clients. In the main loop, Nic-KV calls `epoll()` constantly to get ready read events. Then, Nic-KV uses a loop to iterate through each ready event. When a ready read event is encountered, a new request is received. At this time, the read handler will be called to read the request from the buffer. The request is placed in the receive buffer field of the client object. Then, Nic-KV parses and processes the received request. Finally, the result of processing the request is written to the send buffer of client object. Since RDMA channel has only `POLLIN` events and no `POLLOUT` events, it is always writable as long as the buffer is not full. When the write callback function is triggered, it iterates over all clients (including the master and slaves) whose send buffer is not empty. During each iteration, it calls `ibv_post_send()` to ask the NIC to send the content in send buffer to the corresponding node.

V. Evaluation

Our main evaluation objects are throughput and latency. The throughput refers to the total number of commands

executed by the server in a unit of time, while the latency refers to the time between the client sending a command to server and receiving the execution result of the command. We first demonstrate the performance difference between original Redis and Redis based on RDMA (RDMA-Redis), and then evaluate the performance of SKV.

A. Experimental Setup

Each server in our experiment has two 16-core Intel(R) Xeon(R) Gold 5218 CPU processors with the frequency of 2.30 GHz, and the size of DRAM is 64 GB. Each CPU core is equipped with a 64 KB L1 cache and a 1 MB L2 cache. 16 cores on a CPU share a 22 MB L3 cache. Our testbed runs RoCE (RDMA over Converged Ethernet) [24]. Each server is equipped with an ConnectX-5 MCX516A-CDAT 100Gbps RoCE NIC connected to a Mallanox SN2100 100Gbps RoCE switch. Ubuntu 18.04 with MLNX_OFED_LINUX-5.6-1.0.3.3 stack runs on each server.

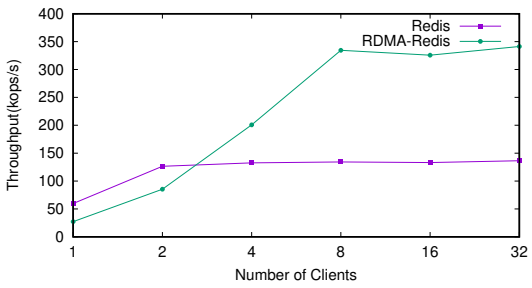
A BlueField MBF2H516A-CENOT SmartNIC is installed on the master node. It is equipped with 8 *ARMv8 A72 cores (64-bit), 16 GB on-board DDR and eMMC flash memory. Every two cores share a 1 MB L2 cache, and the L3 cache size is 6 MB. The SmartNIC has 100Gb/s Ethernet ports and MLNX_OFED_LINUX-5.4-2.4.1.3 stack on it.

B. RDMA Performance

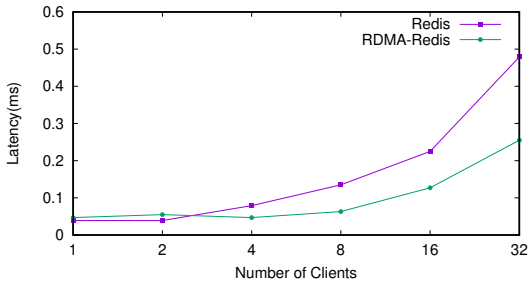
To understand how much RDMA can accelerate the data transfer process, we evaluate the performance difference between RDMA-Redis and original Redis. In the absence of slave nodes, we test the throughput and latency of executing `SET` commands. We use redis-benchmark to measure the performance and each client issues queries as quickly as possible.

Figure 10(a) shows the throughput. In both cases, the throughput increases gradually with the increase of the number of concurrent client connections and finally approaches a fixed value. The reason is that when the number of concurrent clients is small, the throughput is limited by the speed at which the clients send requests. When the degree of concurrency becomes higher, the throughput is limited by the network IO of the server. In the case that the degree of concurrency is small, RDMA-Redis does not show obvious advantages in throughput. With the increase of the number of concurrent connections, the throughput of RDMA-Redis can reach more than 330 kops/s, which greatly exceeds original Redis. Original Redis almost reaches the maximum throughput when there are two concurrent clients, which is only about 130 kops/s.

Figure 10(b) shows 99% tail latency. The latency is almost directly proportional to the number of clients, because the server uses a single thread to read requests and returns the results serially. When the degree of concurrency is small, the latency of the two settings are similar. As the number of concurrent clients increases, the latency of original Redis is nearly double that of RDMA-Redis. The reason is that RDMA one-sided primitive can bypass the system kernel and reduce the time of memory copies. This shows that transferring data



(a) Throughput of original Redis and RDMA-based Redis.



(b) Latency of original Redis and RDMA-based Redis.

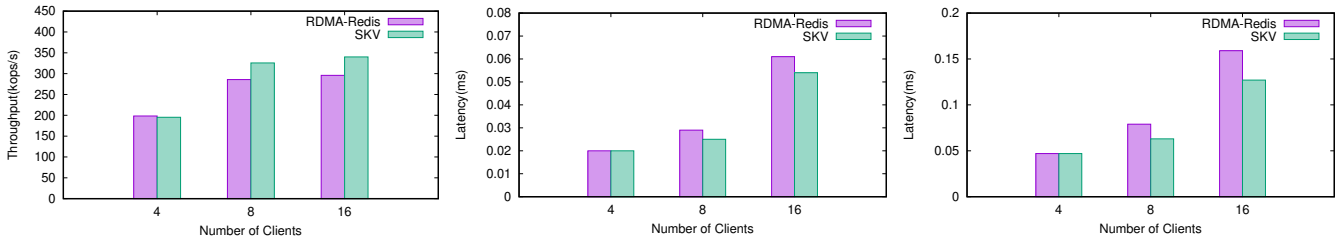
Fig. 10. Performance of original Redis and RDMA-based Redis.

using RDMA alleviates the performance bottleneck brought by the traditional kernel network stack to the distributed key-value store. Therefore, it is unfair to compare the performance of SKV with original Redis. In order to reflect the advantages of SmartNIC offloading, SKV is compared with RDMA-Redis in the following experiments.

C. Replication Offloading Performance

In order to test the performance improvement of SKV in the master-slave mode, we test the throughput and latency of the master node executing commands in the setting of one master with three slaves. We test the situation of 4, 8 and 16 clients respectively. What we are measuring here is the performance of the steady-state replication phase. At the beginning of the test, the master-slave relationship between nodes has been established, and commands are sent to the master node for execution.

It can be seen from Figure 11 that SKV does not obtain a significant performance improvement in the case of 4 clients, but achieves significantly higher throughput and lower latency when there are 8 and 16 concurrent connections. In the case of 8 concurrent client connections, the `SET` throughput of SKV is 14% higher than that of RDMA-Redis. At the same time, the average latency is reduced by 14%, and the tail latency is reduced by 21%. This is because in the steady-state replication phase, the master node in RDMA-Redis needs to send each received `SET` command to all slave nodes one by one. Every time the command is sent, the host CPU needs to post a Work Request (WR) to let the NIC perform RDMA operation. In contrast, SKV only needs the master to send a replication request to Nic-KV for each received `SET` command, which



(a) Throughput. (b) Average latency. (c) 99% latency.

Fig. 11. The performance of SKV when executing SET commands compared with RDMA-Redis.

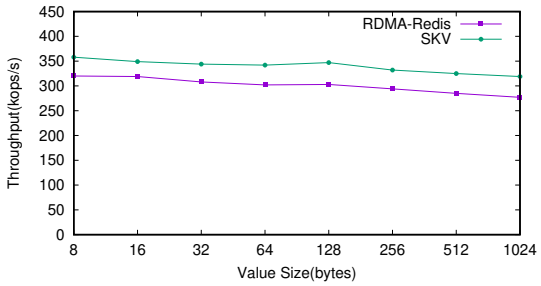


Fig. 12. Throughput under different value sizes.

means that the host CPU only needs to post one WR for the replication of each SET command. This reduces the time the WR is posted, resulting in reducing the CPU consumption in this process. Now the SmartNIC is doing the job of steady-state replication instead of the host. By this way, the CPU cycles saved can be used to send responses to clients, thereby responding to more client requests in the same period of time. When the number of concurrent connections is small, using the SmartNIC to optimize the overhead of replication will not bring about an apparent performance improvement, since the network IO of the master node is not fully occupied. Moreover, we evaluate the performance when the value size varies as shown in Figure 12. Throughput of SKV is higher than RDMA-Redis under different value sizes.

We also test the performance of executing GET commands. The evaluation results are shown in Figure 13. SKV does not achieve better latency and throughput than RDMA-redis under different numbers of concurrent connections. Under 8 and 16 concurrent connections, the throughput of both cases are about 340 kops/s. This is because when the master node receives a GET command, host-KV will discover that this command cannot change the stored value. So this GET command will not be sent to any slave node. The master executes it immediately and returns the query result to the client as soon as possible. No matter how many slave nodes there are, the master node will not spend a lot of CPU cycles to post redundant WRs. The whole execution process is just like there is no slave node, so there is no overhead of replication between nodes. Therefore, SKV cannot take advantage of the offloading design of SmartNIC when performing read tasks.

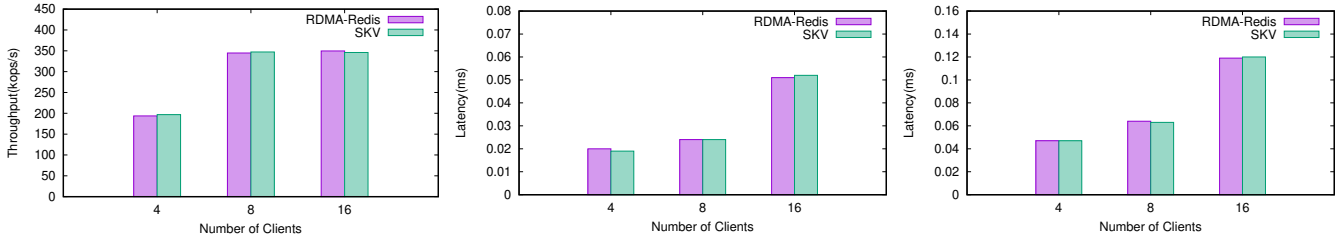
D. Availability

We evaluate the availability of SKV in the case of slave failure. We send commands to the master, and let Host-KV on one of the slaves crash. Figure 14 shows the change in throughput over time. At 4 seconds, Nic-KV detects the failure because a slave does not reply to its probe messages within the specified time. Nic-KV sets an invalid flag for the failed slave in the node list, so the upcoming SET commands will not be replicated to the failed node. Commands are only replicated to available slave nodes. The throughput of the master node does not drop significantly, and it remains above 300 kops/s. At 9 seconds, the slave recovers and it is discovered by Nic-KV. The invalid flag for this slave on the node list is removed, and Nic-KV replicates normally as before. During this period, the client is not aware of the failure of slave.

VI. Related Works

Key-value store is always a hot topic among researchers. To avoid the overhead of the kernel network stack, there are a lot of works using RDMA to speed up key-value stores [6]–[10], [25], [26]. Due to the higher throughput of RDMA one-sided operations, Pilaf [7] and FaRM [6] use one-sided RDMA primitives for GET operations. Pilaf utilizes RDMA READ to process GET requests with low CPU overhead, and proposes self-validating data structures to avoid read and write contention between client and server without client-server coordination. FaRM also uses RDMA READ to read remote data, and it uses RDMA WRITE to pass messages between nodes. Compared with traditional network stacks, FaRM greatly improves latency and throughput. Although the one-sided RDMA operation can bypass the kernel of the peer host, GET operation in this way often brings multiple network round trips. HERD [8] proposes a mixed-use of two-sided SEND/RECV and one-sided WRITE primitives, that is, the WRITE primitive is used to write requests to the remote server, and both GET and SET replies are sent to clients using SEND/RECV. In this way, HERD reduces the round-trip latency and improves the throughput by 2 times compared with FaRM and Pilaf.

As a hardware device that appears in recent years, SmartNIC has gained more and more attention from researchers. First, a batch of works on FPGA-based SmartNICs appear [23], [27]–[35]. Microsoft proposes the ClickNP [28] programming



(a) Throughput. (b) Average latency. (c) 99% latency.

Fig. 13. The performance of SKV when executing GET commands compared with RDMA-Redis.

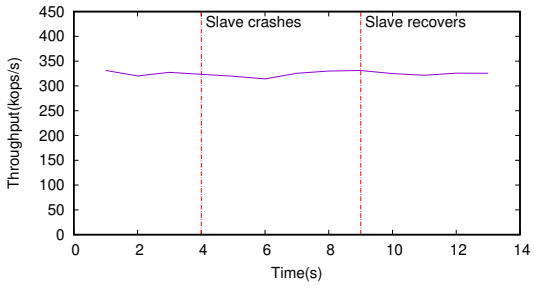


Fig. 14. Throughput during the slave failure.

framework, which is used in commercial servers to support high-performance network functions, such as firewalls and load balancers. NICA [31] presents a hardware-software co-design framework for inline acceleration of application data planes on SmartNICs in multi-tenant systems. KV-Direct [23] proposes a high-performance key-value store, using programmable NIC to extend RDMA primitives and enables remote direct key-value access to host memory. In this work, the SmartNIC is also used to store frequently accessed data, which hides the PCIe latency between the NIC and the host memory.

Now many manufacturers have released SmartNICs with multi-core SoC, including on-path and off-path SmartNICs. Most of the works on multi-core SoC SmartNICs are based on on-path SmartNICs [17]–[19], [36]–[39]. FLOEM [36] tries to simplify the development of network applications that are split across CPUs and NICs by providing a unified framework. ipipe [18] dynamically schedules tasks between the host core and the SmartNICs, maximizing the resource utilization of the SmartNIC and preventing overloading the SmartNIC cores. Xenic [17] uses SmartNICs to accelerate distributed transaction systems, employing a co-designed data store spread across the NIC and the host to improve performance. E3 [19] offloads the microservices platform to SmartNICs and shows that leveraging SmartNICs can improve cluster energy-efficiency.

There are also works utilizing off-path SmartNICs [20], [40]–[43]. Lynx [40] builds a neural network training and inference acceleration platform based on SmartNIC to schedule and manage heterogeneous AI accelerators, which frees the host CPU from heavy tasks. LineFS [20] proposes a persist-and-publish model and pipeline parallelism to offload a distributed file system based on persistent memory to SmartNICs. LineFS outperforms the baseline when the replica nodes are busy. Gimbal [42] demonstrates a software storage switch that enables multi-tenant storage disaggregation on SmartNIC JBOF. HyperNAT [43] offloads network address translation function to multiple SmartNICs on a host, exceeding the processing performance of a single server. However, none of these existing studies focuses on offloading distributed key-value stores.

VII. Conclusion

This paper presents SKV, the first distributed key-value store accelerated with off-path multi-core SoC SmartNIC. We find that some operations of a distributed key-value store consume a lot of CPU resources, resulting in an overall performance degradation. To address this problem, we design a new replication mechanism and offload data replication and failure detection to the SmartNIC. The entire SKV spans the host and the SmartNIC, separating the background data processing from interaction with clients in the front. Our experiments show that SKV improves throughput by 14% and reduces latency by 21% compared with the baseline. In the future, we hope to explore what other applications are suitable for offloading using multi-core SoC SmartNICs.

VIII. Acknowledgement

This research is supported in part by the National Key Research and Development Program of China (2021YFC3300600).

References

[1] “Nosql.” [Online]. Available: https://en.wikipedia.org/wiki/NoSQL
[2] “Redis.” [Online]. Available: https://redis.io/
[3] “Memcached.” [Online]. Available: https://memcached.org/
[4] “Leveldb.” [Online]. Available: https://github.com/google/leveldb
[5] “Key-value store.” [Online]. Available: https://en.wikipedia.org/wiki/Key-value_database
[6] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, “Farm: Fast remote memory,” in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, 2014, pp. 401–414.
[7] C. Mitchell, Y. Geng, and J. Li, “Using one-sided rdma reads to build a fast, cpu-efficient key-value store,” in *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, 2013, pp. 103–114.

[8] A. Kalia, M. Kaminsky, and D. G. Andersen, "Using rdma efficiently for key-value services," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, 2014, pp. 295–306.

[9] ——, "Design guidelines for high performance rdma systems," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 437–450.

[10] B. Cassell, T. Szepesi, B. Wong, T. Brecht, J. Ma, and X. Liu, "Nessie: A decoupled, client-driven key-value store using rdma," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, pp. 3537–3552, 2017.

[11] H. Huang and S. Ghandeharizadeh, "Nova-lsm: A distributed, component-based lsm-tree key-value store," in *Proceedings of the 2021 International Conference on Management of Data*, 2021, pp. 749–763.

[12] "Bluefield data processing units." [Online]. Available: https://www.nvidia.com/en-us/networking/products/data-processing-unit/

[13] "Marvell's data processing units." [Online]. Available: https://www.marvell.com/products/data-processing-units.html

[14] "Broadcom stingray smartnics." [Online]. Available: https://www.broadcom.com/blog/at-a-glance–the-broadcom-stingray-ps1100r-delivers-breakthrough-performance-and-efficiency-for-nvme-of-storage-target-applications

[15] "Netronome agilio smartnic." [Online]. Available: https://www.netronome.com/products/agilio-cx/

[16] A. Ousterhout, J. Fried, J. Behrens, A. Belay, and H. Balakrishnan, "Shenango: Achieving high cpu efficiency for latency-sensitive datacenter workloads," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 361–378.

[17] H. N. Schuh, W. Liang, M. Liu, J. Nelson, and A. Krishnamurthy, "Xenic: Smartnic-accelerated distributed transactions," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 740–755.

[18] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "Offloading distributed applications onto smartnics using ipipe," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 318–333.

[19] M. Liu, S. Peter, A. Krishnamurthy, and P. M. Phothilimthana, "E3:energy-efficient microservices on smartnic-accelerated servers," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 363–378.

[20] J. Kim, I. Jang, W. Reda, J. Im, M. Canini, D. Kostić, Y. Kwon, S. Peter, and E. Witchel, "Linefs: Efficient smartnic offload of a distributed file system with pipeline parallelism," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 756–771.

[21] "Redis replication." [Online]. Available: https://redis.io/docs/manual/replication/#important-facts-about-redis-replication

[22] J. Liu, C. Maltzahn, C. Ulmer, and M. L. Curry, "Performance characteristics of the bluefield-2 smartnic," *arXiv preprint arXiv:2105.06619*, 2021.

[23] B. Li, Z. Ruan, W. Xiao, Y. Lu, Y. Xiong, A. Putnam, E. Chen, and L. Zhang, "Kv-direct: High-performance in-memory key-value store with programmable nic," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 137–152.

[24] "Rdma over converged ethernet (roce)." [Online]. Available: https://www.roceinitiative.org/

[25] N. S. Islam, D. Shankar, X. Lu, M. Wasi-Ur-Rahman, and D. K. Panda, "Accelerating i/o performance of big data analytics on hpc clusters through rdma-based key-value store," in *2015 44th International Conference on Parallel Processing*. IEEE, 2015, pp. 280–289.

[26] Y. Wang, X. Meng, L. Zhang, and J. Tan, "C-hint: An effective and reliable cache management for rdma-accelerated key-value stores," in *Proceedings of the ACM Symposium on Cloud Computing*, 2014, pp. 1–13.

[27] M. T. Arashloo, A. Lavrov, M. Ghobadi, J. Rexford, D. Walker, and D. Wentzlaff, "Enabling programmable transport protocols in high-speed nics," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 93–109.

[28] B. Li, K. Tan, L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen, "Clicknp: Highly flexible and high performance network processing with reconfigurable hardware," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 1–14.

[29] S. Ibanez, A. Mallery, S. Arslan, T. Jepsen, M. Shahbaz, C. Kim, and N. McKeown, "The nanopu: A nanosecond network stack for datacenters," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, 2021, pp. 239–256.

[30] J. Lin, K. Patel, B. E. Stephens, A. Sivaraman, and A. Akella, "Panic: A high-performance programmable nic for multi-tenant networks," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 243–259.

[31] H. Eran, L. Zeno, M. Tork, G. Malka, and M. Silberstein, "Nica: An infrastructure for inline acceleration of network applications," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 345–362.

[32] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung *et al.*, "Azure accelerated networking: Smartnics in the public cloud," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 51–66.

[33] M. Lavasani, H. Angepat, and D. Chiou, "An fpga-based in-line accelerator for memcached," *IEEE Computer Architecture Letters*, vol. 13, no. 2, pp. 57–60, 2013.

[34] K. Lim, D. Meisner, A. G. Saidi, P. Ranganathan, and T. F. Wenisch, "Thin servers with smart pipes: Designing soc accelerators for memcached," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 36–47, 2013.

[35] Y. Tokusashi and H. Matsutani, "A multilevel nosql cache design combining in-nic and in-kernel caches," in *2016 IEEE 24th Annual Symposium on High-Performance Interconnects (HOTI)*. IEEE, 2016, pp. 60–67.

[36] P. M. Phothilimthana, M. Liu, A. Kaufmann, S. Peter, R. Bodik, and T. Anderson, "Floem: A programming system for nic-accelerated network applications," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 663–679.

[37] Y. Qiu, J. Xing, K.-F. Hsu, Q. Kang, M. Liu, S. Narayana, and A. Chen, "Automated smartnic offloading insights for network functions," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 772–787.

[38] R. Shashidhara, T. Stamler, A. Kaufmann, and S. Peter, "Flextoe: Flexible tcp offload with fine-grained parallelism," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 87–102.

[39] P. Gao, Y. Xu, and H. J. Chao, "Ovs-cab: Efficient rule-caching for open vswitch hardware offloading," *Computer Networks*, vol. 188, p. 107844, 2021.

[40] M. Tork, L. Maudlej, and M. Silberstein, "Lynx: A smartnic-driven accelerator-centric architecture for network servers," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 117–131.

[41] Y. Le, H. Chang, S. Mukherjee, L. Wang, A. Akella, M. M. Swift, and T. Lakshman, "Uno: Uniflying host and smart nic offload for flexible packet processing," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 506–519.

[42] J. Min, M. Liu, T. Chugh, C. Zhao, A. Wei, I. H. Doh, and A. Krishnamurthy, "Gimbal: enabling multi-tenant storage disaggregation on smartnic jbofs," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, 2021, pp. 106–122.

[43] S. Fang, Q. Liu, and W. Wu, "Hypernat: Scaling up network addresstranslation with smartnics for clouds," *arXiv preprint arXiv:2111.08193*, 2021.