2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService) | 978-1-6654-3483-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/BigDataService52369.2021.00011

TransKV: A Networking Support for Transaction Processing in Distributed Key-value Stores

Hebatalla Eldakiky and David Hung-Chang Du
Department of Computer Science and Engineering
University of Minnesota–Twin Cities, USA
E-mails: {eldak002, du}@umn.edu

***Abstract*—Through the massive use of mobile devices, data clouds, and the rise of Internet of Things, enormous amount of data has been generated and analyzed for the benefit of society. NoSQL Databases and specially key-value stores become the backbone in managing these large amounts of data. Most of key-value stores ignore transactions due to their effect on degrading key-value store's performance. Meanwhile, programmable switches with the software-defined networks and the Programming Protocol-Independent Packet Processor (P4) lead to a programmable network where in-network computation can help accelerating the performance of applications. In this paper, we proposed a networking support for transaction processing in distributed key-value stores. Our system leverages the programmable switch to act as a transaction coordinator. Using a variation of the time stamp ordering concurrency control approach, the programmable switch can decide to proceed in transaction processing or abort the transaction directly from the network. Our experimental results on an initial prototype show that our proposed approach, while supporting transactions, improves the throughput by up to 4X and reduces the latency by 35% when compared to the existing architectures.**

I. Introduction

Big data has attracted lots of people's attention. Nowadays, enormous amount of data has been generated and analyzed for the benefit of society at a large scale. With this huge amount of generated data, data is distributed among several storage instances, accessed frequently, retrieved and processed by many applications to extract useful information. So, it is important to improve the data access performance when data is accessed from storage nodes through network.

Nowadays data is being generated by many different sources with un-unified structures, hence this data is often maintained in key-value storage, Which is widely used due to its efficiency in handling data in key-value format, and flexibility to scale out without significant database redesign. According to DB-Engines [5], key-value store is one of the most popular NoSQL databases which are broadly used as the storage engine for high-traffic websites and other high-performance content. Examples of popular key-value stores include Dynamo [6], RocksDB [30], Redis [28], Memcached [24]. These key-value store engines have been extensively used by different clients including Amazon, Facebook, Nokia and Samsung [16].

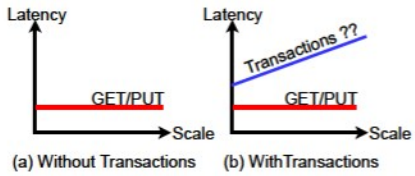
Some of the applications built on these key-value stores employ non-trivial concurrent transactions from multiple clients. Consequently, managing all of these concurrent transactions without adequate concurrency control creates significant problems for the application. For example, in Amazon's e-commerce platform, the shopping cart service processes tens of millions requests that come from over 3 million checkouts in a single day. Each of these requests represents a transaction, that should guarantee the different ACID properties in order to reach a correct database state.

Unfortunately, the distributed architecture of key-value stores makes it difficult to implement the required ACID properties for supporting transactions. Implementing the transaction concepts has a negative effect on the main two targets of any key-value store: scalability and predictable perfromance, as shown in Figure 1. This effect is due to the complexity, locking, starvation introduced by transactions and the interference with other non-transaction operations. That is why, some key-value stores [17], [32] omit the transaction concepts. However, other key-value stores [7], [28] support transaction concepts by introducing a transaction coordinator. The transaction coordinator is responsible for the coordination among the key-value storage nodes. Each storage node implements a concurrency control mechanism to decide whether to accept or reject a transaction, then the transaction coordinator aggregates these decisions from the participating nodes and decides whether to abort or accept the transaction before processing it. Unfortunately, this model introduces lots of communications and forwarding steps in key-value queries processing, which is usually carried out through network switches. These additional steps increase the response time of the key-value queries.

On the networking side, Software-defined Network (SDN) simplifies network devices by introducing a logically centralized controller (control plane) to manage simple programmable switches (data plane). Recently, the Programming Protocol-Independent Packet Processor (P4) [3] unleashes capabilities that give the freedom to create intelligent network nodes performing various functions. Thus, applications can accelerate their performance by offloading part of their computational tasks to these programmable switches to be executed in the network. Nowadays, programmable networks get a bigger foot in the data center doors. Google cloud started to use the programmable switches with P4Runtime [26] to build and control their smart networks [11]. Some Internet Service Providers (ISPs), such as AT&T, have already integrated programmable switches in their networks [1].

Now, as we have control over both network and storage, it is time to think about how to improve data access per-

978-1-6654-3483-6/21/$31.00 ©2021 IEEE
DOI 10.1109/BigDataService52369.2021.00011



(a) Without Transactions (b) WithTransactions

Fig. 1: Performance of KV Stores w/o Transactions

formance when applications access storage through network. In this paper, we propose TransKV: a network support for transaction processing using the programmable switches to further improve the latency of transactional key-value queries. We believe that network latency has a significant impact on the performance of transactions which have to be processed by the storage system in order to ensure serializability. TransKV utilizes the programmable switches as a concurrency control manager to execute the transaction processing logic in network. We are developing a variation of the Timestamp Ordering (TSO) [29] algorithm on the programmable switches. If a transaction can start processing according to the TSO logic, it is accepted and forwarded to the storage nodes. Otherwise, the transaction is aborted early by the programmable switches, and packets are routed back to the client.

TransKV adapts a hierarchical caching scheme to distribute the hottest key-value pairs on the data plane of data center's switches, where higher level of cache contains the hottest key-value pairs, and the hotness of data decreases while going down in the hierarchy. TransKV provides a transactional support by injecting some information about the requested data in packet headers. The programmable switches use this information along with the timestamps saved for all cached key-value pairs to decide whether to accept the transaction or abort it and send the packet back to the client.

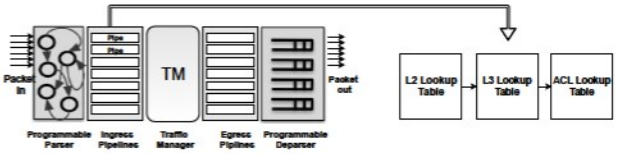
TransKV utilizes the architecture of software-defined network [10], [22]. In our architecture, a logically centralized controller has a global view of the whole system [12]. This logically centralized controller manages the log of all transactions' history for failure recovery. It acts also as the transaction coordinator for the non-cached key-value pairs. It also updates the cache of each switch by the hottest key-value pairs periodically. Our Experimental evaluation based on our initial prototype shows that TransKV improves the throughput by up to 4X and reduces the latency by 35% on average

The remaining sections of this paper are organized as follows. Background is discussed in Section II. Section III provides the architecture of TransKV, while the detailed design of TransKV is presented in Section IV. TransKV implementation is discussed in Section V, while Section VI gives an experimental evidence and analysis of TransKV. Section VII provides a short survey about the related work, and finally, the paper is concluded in Section VIII.

II. Background

A. Preliminaries on Programmable Switches

Software-Defined Network (SDN) simplifies network devices by introducing a logically centralized controller (control plane) to manage simple programmable switches (data plane). SDN controllers set up forwarding rules at the programmable switches and collect their statistics using OpenFlow APIs [22]. As a result, SDN enables efficient and fine-grained network management and monitoring. Recently, P4 [3] has been introduced to enrich the capabilities of network devices by allowing developers to define their own packet formats and build the processing graphs of these customized packets. P4 is a programming language designed to program parsing and processing of user-defined packets using a set of match/action tables. It is a target-independent language, thus a P4 compiler is required to translate P4 programs into target-dependent switch configurations.



(a) Switch Data Plane Architecture (b) Multiple Stages of Pipeline

Fig. 2: Primelinries on Programmable Switches

As shown in Figure 2(a), The data plane for most of modern switch ASICs consists of five main components. These components include programmable parser, ingress pipeline, traffic manager, egress pipeline and programmable deparser. When a packet is received by an ingress port, it goes through the programmable parser. The parser is modeled as a simple deterministic state machine, that consumes packet data and identifies headers that will be recognized by the data plane program. It makes transitions between states typically by looking at specific fields in the previously identified headers.

After the packet is processed by the parser and decomposed into headers, it goes through one of the ingress pipes to enter a set of match-action processing stages as shown in Figure 2(b). In each stage, a key is formed from the extracted data and matched against a table of some entries. These entries can be added and removed through the control plane. If there is a match with one of the entries, the corresponding action will be executed, otherwise a default action will be executed. After the action execution, the packet and the updated headers go through next stages in the pipeline.

After the packet finishes all the stages in the ingress pipeline, it is queued and switched by the traffic manager for an egress pipe for further processing. At the end, there is a programmable deparser that performs the reverse operation of the parser. It reassembles the packet back with the updated headers so that the packet can go back onto the wire to the next hop in the path to its destination.

B. Timestamp Ordering Concurrency Control

Transaction processing systems require high availability and fast response time for hundreds of concurrent users. With the concurrent access of hundreds of users, concurrency control is needed to ensure the correct execution of users' transactions, when multiple transactions submitted by various users interfere with one another in a way that produces incorrect results.

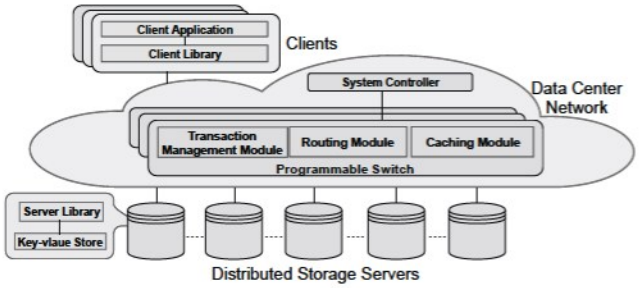


Fig. 3: TransKV Architecture Overview

Timestamp Ordering (TSO) [8], [29] is a concurrency control protocol that guarantees serializability using transaction's timestamps to order transaction execution for an equivalent serial schedule. The idea for this scheme is to order the transactions based on their timestamps, transaction start time, then the transaction processing system will only allow the transactions to be processed according to that order.

The TSO algorithm must ensure that, for each item accessed by conflicting operations in the schedule, the order in which the item is accessed does not violate the timestamp order. To do this, the algorithm associates with each database item X two timestamp (TS) values: read timestamp and write timestamp. The read timestamp of item X is the largest timestamp among all the timestamps of transactions that have successfully read item X. The write timestamp of item X is the largest of all the timestamps of transactions that have successfully written item X. The system checks timestamps for every operation. If a transaction tries to read/write an object from the future, i.e., with greater read/write timestamp than the transaction's timestamp, it aborts and restarts. TSO is different from any locking protocol; it determines serializability order of transactions based on their timestamps before execution without using any locks, and hence it is a deadlock-free protocol which make it more suitable for the programmable switches architecture.

III. TransKV Architecture Overview

TransKV is a networking support for transactions in distributed key-value stores that leverages the capability of programmable switches. TransKV utilizes the programmable switches as a transaction manager to coordinate between the submitted transactions. Figure 3 shows the architecture of TransKV within a data center, which consists of the programmable switches, controller, storage nodes, and clients.

Programmable Switches. Programmable switches are the essential component in our proposed system. We augment the programmable switches with a cache [15] to store the popular key-value pairs, and leverage match-action tables and registers in the programmable switches to design the in-switch transaction coordinator where the values and the time stamps information of cached key-value pairs will be stored. The programmable switch uses a variation of the TSO along with the stored information to decide whether the submitted transaction can start processing and be routed to the target storage node, or can be aborted directly from the network before reaching the storage node. Following this approach, the programmable switches act as transaction coordinator nodes that coordinate between the submitted transactions.

In addition to transaction coordination, each programmable switch has a query statistics module to provide the controller with statistics reports to enable it to update the cache with the popular key-value pairs. TransKV can distinguish between packets. Packets marked as TransKV packets, are processed by our system. Other packets are processed and routed using the standard L2/L3 protocols which make TransKV compatible with other network functions and protocols.

Controller. The controller is primarily responsible for system reconfigurations including (a) log management for recovering from system failures, (b) transaction coordinator for non-cached key-value pairs, and (c) updating each switch's cache with the recent popular key-value pairs. The controller keeps track with all changes made by the transactions on a log. In case of any system failure, this log is done/undone in order to restore the system to a consistent state. Through the control plane, the controller also updates the match-action tables in the switches with the new popular key-value pairs. TransKV controller is an application controller that is different from the network controller in SDN, and it does not interfere with other network protocols or functions managed by the SDN controller. *Our controller only manages the transaction log and the cached key-value pairs. Both controllers can be co-located on the same server, or on different servers.*

Storage Nodes. They represent the location where the key-value pairs reside in the system. The key-value pairs are partitioned among these storage nodes. Each storage node runs a simple shim that is responsible for reforming TransKV query packets to API calls for the key-value store. This layer makes it easy to integrate our design with existing key-value stores without any modifications to the storage layer.

Clients. TransKV provides a client library which can be integrated with the client applications to send TransKV packets through the network, and access the key-value store without any modifications to the application.

IV. TransKV Design

The data plane provides on-switch cache and transaction coordination model for the key-value stores to handle concurrency control earlier in network. In this model, all timestamps for cached key-value pairs are stored on switches, and are used for transaction coordination. Figure 4 represents the whole pipeline that the packet traverses inside the switch before being forwarded to the storage node for processing, or aborted by the switch and forwarded back to the client. In this section, we discuss how the switch supports these functions.

A. Network Protocol Design

Packet Format. Figure 5 shows the format of TransKV request packet sent from clients. The programmable switches use a reserved port number in the TCP/UDP header to identify TransKV packets, and lookup the cache for non-transactional

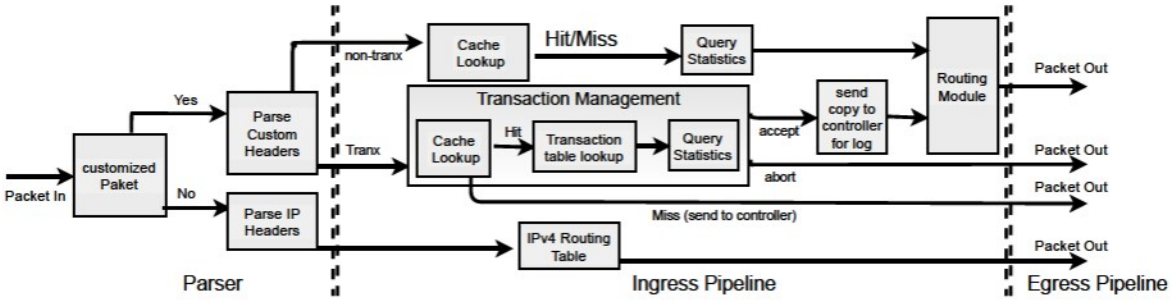


Fig. 4: Logical View of TransKV Data Plane Pipeline

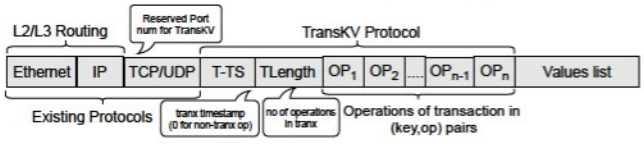


Fig. 5: TransKV Packet Format

packets, or execute the transaction management process for the transactional packets. Other switches in the network do not need to understand the TransKV header, and treat all packets as normal IP packets. The TransKV header consists of three main fields: `T-TS`, `TLength`, and a list of `OP`. The `T-TS` is a 4-byte field which stands for the submitted transaction timestamp. A packet with `T-TS = 0` represents a non-transactional operation. `TLength` stores the number of key-value operations on the submitted transaction. The `OP` list is a list of (key,op) pairs of length equals to `TLength`, each pair represents a 4-bit operation followed by a requested key.

After the packet is processed by the programmable switch, the switch either accepts the packet and sends it to the storage nodes for processing, or aborts the transaction and sends the reply back to the client. The reply packet is a standard IP packet, and the result is added to the packet payload.

Network Routing. TransKV uses existing routing protocols to forward clients' packets through the network. For a TransKV packet, based on the information of data location, the client appropriately sets the Ethernet and IP headers and sends the packet to the storage server where data resides. The programmable switches, placed on the path from the clients to the storage clusters, process TransKV packets. If the packet represents a transaction, the switch either accepts the packet and sends it to the storage nodes for processing or aborts the transaction and sends the reply back to the client. If the packet represents a non-transactional operation, the switch attaches the value to the packet, in case of cache hit, and sends the reply back to the client, or sends the packet to the storage nodes if the requested item is not in cache. Other switches simply forward packets based on the destination MAC/IP address according to the L2/L3 routing protocol. In this way, TransKV can coexist with other network protocols.

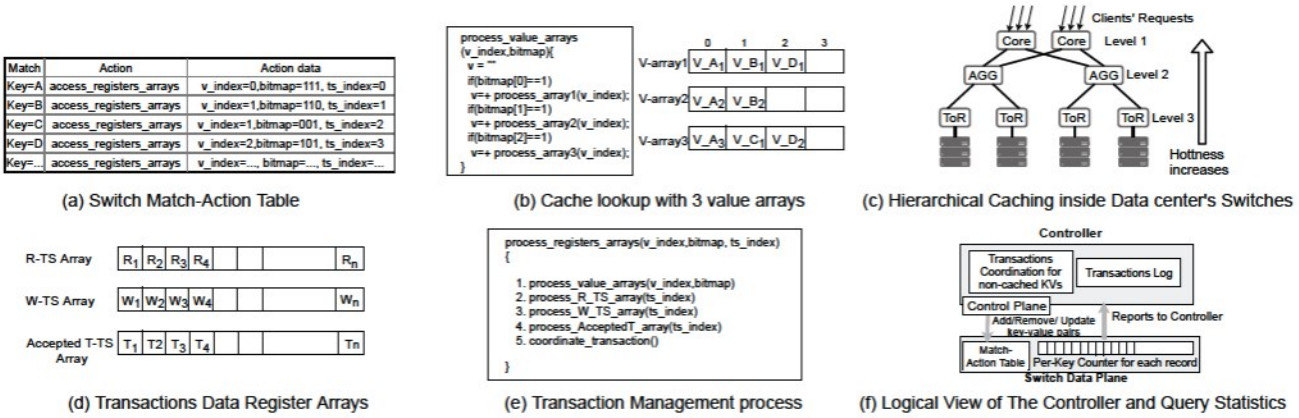
B. On-Switch Cache

TransKV adopts on switch cache, where hot key-value pairs are stored on switch match-action table and switch registers. There are two types of match-action tables inside the switches: the routing match-action table and the key-value cache. In this section we will discuss the design of the key-value cache. The key-value cache design is shown in Figure 6(a). Each record in the table consists of three parts: *match*, *action* and *action data*. The *match* represents the value that we match the requested key against, TransKV uses the exact-match to match a requested key against a record in the match-action table. The *action* represents the transaction management process that will be executed when a requested key matched a record in the match-action table. The *action data* consists of three parts: `v_index`, `bitmap` and `ts_index`. `v_index` and `bitmap` are used as in [15] to support storing of variable length key-value pairs in the switch's registers. Figure 6(b) shows an example of 3 registers cache and how we use the `v_index` and `bitmap` to retrieve the value of a requested key. `ts_index` represents the index of the timestamps associated to a key-value pair in the switch registers. The timestamps of a record are stored at the same index in all registers. There are three registers used to store the timestamps as shown in Figure 6(d): read timestamp register, write timestamp register and submitted transaction timestamp register. The usage of each of these registers will be discussed in Section IV-C.

TransKV uses a hierarchical caching approach as shown in Figure 6(c), the total cache size equals to $n \times m$, where n is the number of switches in the data center network and m is the size of each switch cache. Each key-value pair is cached only once on the whole cache, where the hottest key-value pairs are cached on top level (level 1) where the client requests are submitted, hot key-value pairs are cached on the middle level (level 2), and warm key-value pairs are cached on the bottom level (level 3). The controller decides which key-value pairs reside in each level.

C. On-Switch Transaction Management

TransKV acts as a transaction manager for the cached key-value pairs. Figure 6(e) shows the transaction management process. A variation of Timestamp Ordering (TSO) concurrency control protocol is implemented on the programmable switches. TSO is chosen because it is a deadlock free approach; there is no locking mechanism used on the records, and hence there is no checking in the dependency graph for cycles, which makes it suitable for the match-action pipeline. For each cached key-value pair, the read timestamp $R-TS$,



(a) Switch Match-Action Table (b) Cache lookup with 3 value arrays (c) Hierarchical Caching inside Data center's Switches

(d) Transactions Data Register Arrays (e) Transaction Management process (f) Logical View of The Controller and Query Statistics

Fig. 6: TransKV Design inside Switch Data Plane

the write timestamp $W-TS$, and the timestamp of the current accepted transaction $T-TS$ are stored on the switch registers as shown in Figure 6(d).

Read Operation within a transaction. If there is a submitted transaction T_i with timestamp $TS(T_i)$ submitted on one of the cached key-value pairs X, where T_i contains a read operation, the switch will fetch x's corresponding timestamps from the registers $R-TS(X)$, and $W-TS(X)$. Then, the switch compares $TS(T_i)$ with X's timestamps and decide whether to accept or abort the operation as shown in Figure 7(a). If there is a previously accepted transaction T_j on X $(T-TS>0)$, the switch will fetch $TS(T_j)$ from the register of submitted transaction, and decide whether to accept or abort the operation as shown in Figure 7(b).

Write Operation within a transaction. If there is a submitted transaction T_i with timestamp $TS(T_i)$ submitted on one of the cached key-value pairs X, where T_i contains a write operation, the switch will fetch x's corresponding timestamps. Then, the switch compares these timestamps and decide whether to accept or abort the operation as shown in Figure 8. After the accepted transaction is processed by the storage node, the storage node sends the transaction back to the switch with a committed status. The switch uses the value in the packet of committed transaction to update the value stored on the switch. So the updated value will be available to other transactions submitted on the same key-value pair.

Transaction that contains multiple key-value operations (`Tlength > 1`) will be accepted if all its key-value operations are accepted, and it will be aborted otherwise. The transaction could pass by several switches in the path to the target storage nodes before it is accepted or aborted; as the requested key-value pairs may be cached on different switches.

D. Query Statistics

In TransKV, the data plane has a query statistics module to provide query statistics reports to the controller about the popularity of key-value pairs. Thus, through control plane, the controller updates the cache on each switch with the most popular key-value pairs. As shown in Figure 6(f), the switch's data plane maintains a per-key counter for each key in the match-action table. Upon each hit for a key, its corresponding counter is incremented by one. The switch also will notify the controller about each cache miss, so the controller can estimate the popularity of non-cached items. The controller receives reports periodically from the data plane including these statistics, and resets these counters periodically. Based on the received statistics, the controller updates the cache and all registers with the new popular items.

E. Transaction Log Management

One of the main transaction concepts is consistency, which means that data must be in a consistent state when the transaction starts, and when it ends. If a failure happens during the processing of any transaction, all changes made by that failed transaction must be undone to guarantee data consistency. That is why TransKV controller stores a log of all submitted transactions' history, this log is used to redo/undo the changes made by committed/failed transaction. When a transaction is submitted and reaches the programmable switch, the switch sends a copy of this transaction to the controller, then the controller fetches the old data from key-value stores that is related to the submitted transaction, and finally controller creates a record for that transaction with the after and before images, and append this record to the recovery log. If a failure happens, the controller rolls back all the changes made by any uncommitted transaction, rolls back also any transaction that read a value written by the failed transaction (cascading rollback), and restore the database to a consistent state.

F. Transaction Management for non-cached KV Pairs

The programmable switches have a limited on-chip capacity, so all key-value pairs can not be cached on the switches, and hence switches can only cache the most popular key-value pairs, and manage the transactions targeting these cached data. For other non-cached key-value pairs, the controller will take the transaction coordination role. The controller will act as the transaction coordinator described in [7], [28]. The controller checks with each storage node, participating in the transaction. Each storage node sends its decision of

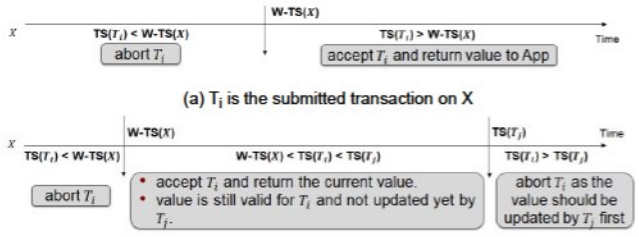


Fig. 7: Acceptance or abortion of Read Operation

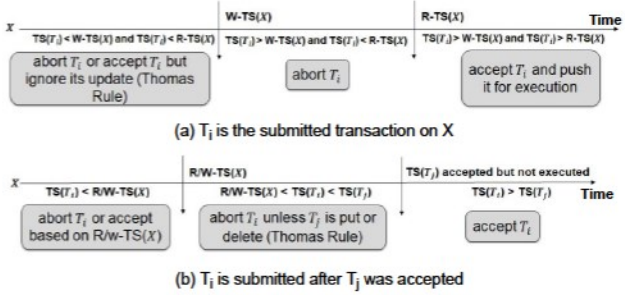


Fig. 8: Acceptance or abortion of write Operation

whether to accept or reject the transaction, then the controller aggregates these decisions and sends the final decision back to the client. Because of the 80/20 rule in data science, we can see that only 20% of data is accessed 80% of the time and vise versa. By applying this rule on TransKV, nearly 80% of submitted transactions will be managed by the switches and 20% will take the normal path of transaction coordination via the controller, so the amortized response time for the transactions will be improved.

V. Implementation

We have implemented a prototype of TransKV, including all switch data plane and control plane features, described in Section IV. We have also implemented the client and server libraries that interface with applications and storage nodes, respectively, as described in Section III. The switch data plane is written in P4, and due to lack of real hardware, it is compiled to the simple software switch BMV2 [2] running on Mininet [23]. The key size of the key-value pair is 16 bytes with total key range spans from 0 to 2^{128}. The cache lookup table has 64K entries. We used 3 registers for storing the value, each register has a 64K 16-byte slots with value granularity of 16 byte and up to 48 bytes. We also used 4 registers, each of them has 64K 4-byte slots: 3 of them for storing the timestamps of all cached key-value pairs, and the other register is used to count the access requests of cached key-value-pairs for query statistics module. The controller is able to update/read the values of these registers through the control plane. It also can add/remove key-value pairs to/from the caching table. The controller is written in Python and can update the switch data plane through the switch driver by using the Thrift API generated by the P4 compiler. The total on-chip used memory is around 5MB leaving enough space for processing other network operations.

The client and server libraries are written in Python using Scapy [31] for packet manipulation. The client can translate a YCSB [4] workload with different distributions and mixed key-value operations into TransKV packets and send them through the network. We used Plyvel [27] which is a Python interface for levelDB [18] as the storage agent. The server library translates TransKV packets into Plyvel format and connects to levelDB to perform the key-value pair operations.

VI. Performance Evaluation

This section provides the experimental results of TransKV. We show the performance improvement of TransKV on the key-value operations latency and system throughput.

Experimental Setup. Our experiments consist of eight simple software switches BMV2 [2] running on Mininet (2 Core switches, 2 aggregation switches and 4 ToR switches). The switches are connected together as shown in Figure 6(c). Each ToR switch is connected to 4 storage nodes with total of 16 storage nodes. Each core switch is connected to 2 clients with total of 4 clients. Each of the clients runs the client library and generates the key-value transactional and non-transactional operations. Each storage node runs the server library and uses LevelDB as the storage agent. The data is range-partitioned over the storage nodes, where each storage node is responsible for handling part of the total key span.

Comparison. We compared our in-switch transaction management model (TransKV) with the transaction coordinator model (we refer to it as Tranx-Coor) shown in [7]. In TransKV, the hottest key-value pairs are stored on the switch cache. Any submitted transaction on one of the cached key-value pairs is managed by the switch. Other non-cached key-value pairs are managed by TransKV controller which acts as the transaction coordinator. In Tranx-Coor, The transaction coordinator is responsible for the coordination among the key-value storage nodes. Each storage node implements the TSO concurrency control mechanism to decide whether to accept or reject a transaction, then the transaction coordinator aggregates these decisions from the participating nodes and decides whether to abort or accept the transaction before processing it.

Workloads. We use both uniform and skewed workloads to measure the performance of TransKV under different workloads. The skewed workloads follow Zipf distribution with different skewness parameters (0.95, 0.99, 1.2). These workloads are generated using YCSB [4] basic database with 16 byte key size and 48 byte value size. The generated data is stored into files, then parsed by the client library to convert them into TransKV packets. We generate different types of workloads: transactional and non transactional read-only workload, transactional write-only workload and transactional mixed workload with multiple write ratios.

A. Effect on System Throughput

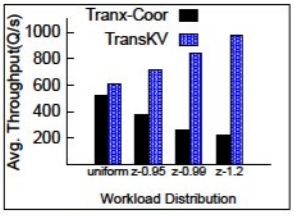
Impact of Read-only Workloads. Figure 9(a) shows the system throughput under different skewness parameters with transactional read-only workload. We compare TransKV vs

	Uniform			Zipf-0.99			Zipf-1.2		
	Mean	50 Percentile	99 Percentile	Mean	50 Percentile	99 Percentile	Mean	50 Percentile	99 Percentile
TransKV	0.07061354	0.072377563	0.104059825	0.063888747	0.067531943	0.099380732	0.053317506	0.048128009	0.095414464
Tranx-Coor	0.076945137	0.075366497	0.104418943	0.075576334	0.074002504	0.104888678	0.078707565	0.076314449	0.111776471

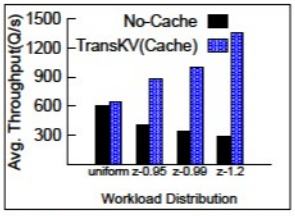
TABLE I: Read Transactions Latency Analysis Under Different Workloads

	Uniform			Zipf-1.2		
	Mean	50 Percentile	99 Percentile	Mean	50 Percentile	99 Percentile
TransKV	0.081189324	0.080517054	0.120302806	0.0614705	0.051731467	0.11642298
Tranx-Coor	0.080760114	0.076931477	0.116496089	0.080588034	0.078615069	0.114953876

TABLE II: Write Transactions Latency Analysis Under Different Workloads

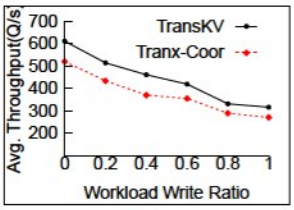


(a) Transactional Operations

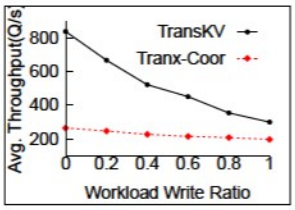


(b) Non Transactional Operations

Fig. 9: Throughput vs Skewness - Read Only



(a) Uniform



(b) Zipf-0.99

Fig. 10: Throughput with Different Write Ratios

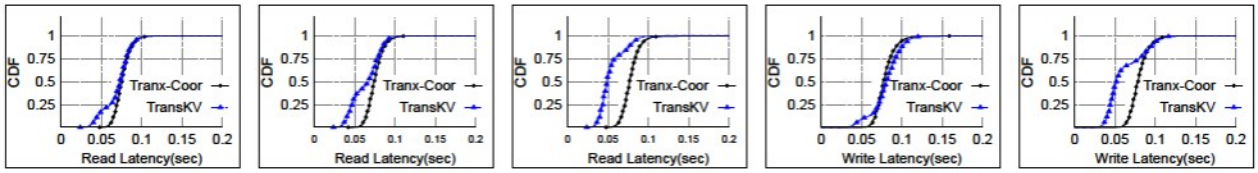
Tranx-Coor. As shown in Figure 9(a), TransKV outperforms Tranx-Coor by minimum of 1.2X in case of uniform workload and by maximum of 4.4X in case of skewed workload. This result is because TransKV manages the transactions for the hot key-value pairs directly in the switch data plane. It decides using the TSO whether to accept or reject the transaction, and eliminates any excessive communication steps between the transaction coordinator and the storage nodes for decisions aggregation in case of Tranx-Coor. Moreover, the key-value pairs are stored on the switch data plane, they are retrieved directly from the switch without the need to go to the storage node to fetch the value. Also, when the skewness parameter increases, the throughput of the system increases; more hits occur on the switch cache ,and small amount of key-value pair misses are coordinated through TransKV controller.

Figure 9(b) shows the system throughput under different skewness parameters for the non-transactional read-only workload. We compare the performance of TransKV when it caches the hottest key-value pairs in the switches without performing the TSO logic vs the performance of accessing the key-value pairs normally from the storage nodes without any caching on the switch side (No Cache). As shown in Figure 9(b), TransKV also outperforms the No Cache approach by 7% in case of uniform workload and maximum of 5X in the skewed workload. This result is because frequent requests to hot data over cold data lead to load imbalance among storage nodes; some nodes are heavily congested while others become under-utilized. This results in a performance degradation of the whole system and a high tail latency. But using caching on the switch side absorbs the hot key-value queries and increase the throughput of the whole system. So we can conclude from Figure 9(a) and Figure 9(b) that TransKV improves the throughput of the key-value storage for both the transactional and non-transactional key-value operations.

Impact of Write Ratio. Figure 10(a) and Figure 10(b) show the system throughput under uniform and skewed workload with varying the workload write ratio for transactional Operations. As shown in Figure 10(a) and Figure 10(b), TransKV outperforms the transaction coordinator (Tranx-Coor) for both the unifrom and skewed workload by minimum of15% and maximum of 25% in case of uniform workload and by minimum of 1.5X and maximum of 3X in case of skewed workload. This is because the management of the concurrency control logic in switch data plane and the elimination of excessive communication steps between the transaction coordinator and the storage nodes. We can see also from the same figures that as the workload write ratio increases, the throughput decreases. This result is because write requests can't be completed without writing the value in the storage node, so that other requests to the same Key-value pair can see the latest update. So when the write ratio increases, higher percentage of the requests will travel longer path to reach the storage node and persist the value which reduces the throughput of the system.

B. Effect on Key-value Transactions Latency

Figures 11(a), 11(b) and 11(c) show the CDF of read transactions latency on key-value pairs under uniform, Zipf-0.99 and Zipf-1.2 workloads, respectively, for TransKV and Tranx-Coor. The analysis of these three figures is shown in Table I. Figure 11(a) shows that TransKVperforms the same as the Tranx-Coor with very small latency improvement equals to 8% on average for the uniform workload; as the effect of caching significantly degrades when the data is uniformly accessed, small amount of key-value transactions will be managed by the switch and other transactions will be managed by TransKVcontroller, which is the normal transaction coordinator path as well in Tranx-Coor. When the skewness



(a) Uniform (b) Zipf-0.99 (c) Zipf-1.2 (d) Uniform (e) Zipf-1.2

Fig. 11: Request Latency (the three left subfigures (a, b and c) represent read latency under different workloads, and the two right subfigures (d and e) represent write latency under different workloads)

of data increases, the caching effect goes in action with more hits in the switch cache. This effect makes TransKV outperforms Tranx-Coor with 15% and 32% on average in the case of zipf-0.99 workload and zipf-1.2 workload as shown in Figures 11(b) and 11(c), respectively; as TransKV manages larger number of transactions using TSO logic in the switch data plane and decides directly whether to accept or abort them. This approach makes TransKV avoids the excessive communications introduced by Tranx-Coor. Moreover, for read transactions, the transaction can retrieve the value directly from the switch cache without the need to go to the storage node to retrieve it.

Figures 11(d) and 11(e) show the CDF of write transactions latency on key-value pairs under uniform and Zipf-1.2 workloads for TransKV and Tranx-Coor. The analysis of these two figures is shown in Table II. Similar to our observation from the read latency results, TransKV has no improvement in case of uniform workload, but TransKV outperforms Tranx-Coor when the data skewness increases as shown in Figure 11(e); this is because hit ratio in switch data plane increases, and TransKV manages the transactions of the cached key-value pairs directly in the switch. TransKV reduces the latency by 24% on average and by 34% for the 50^{th} percentile. The latency improvement in write transactions is less than the read transactions, because the write transactions need to reach the storage node to update the value and make it accessible for other transactions.

VII. Related Work

Distributed Key-value Stores. Key-value storage is widely used to support lots of large-scale applications. Some key-value stores manage data in DRAM for faster data access [24], [25], [28]. Other key-value stores are presistent key-value stores which save data on presistent storage devices [6], [17], [18], [30], while other key-value stores use hybrid storage(DRAM and SSD) [32]. Most key-value stores omit the transactions because of the negative effect on key-value store performance, but others [7], [28] use the concept of transaction coordinator to solve this problem. Also, there is Wrap [9], a transactional system over key-value store, that uses a protocol called acyclic transactions for providing serializable transactions on top of a sharded data store. It allows multiple transactions to prepare and commit simultaneously. TransKV supports transactions in distributed key-value stores without any extra communications or forwarding steps introduced by the transaction coordinator.

Hardware Acceleration. Emerging hardware is used speed up the performance of the distributed systems. Some distributed systems use the programmable switches to improve their performance like: [15], [19], [21] that use the programmable switches to balance the load among storage nodes, NetChain [14] that uses the switches to implement in-network key-value store, iSwitch [20] which uses the switches to improve the performance of the distributed reinforcement learning, JoiNS [34] which uses the OpenFlow switches to prioritize I/O packets to meet their latency SLO, Concordia [33], a distributed shared memory that use the programmable switches for in-network cache coherence and finally Gotthard [13] that uses the optimistic concurrency control along with the programmable switches to cache some transaction results, and based on that cached history, it aborts some transactions that are likely to be aborted by the storage server. Gotthard operates only on a single storage server and single switch. TransKV uses the TSO for transaction processing on the switches. It caches the hot key-value pairs on switch's data plane to execute the TSO logic and accept or reject transactions directly from network. Moreover, TransKV is scalable to the data center network with multiple switches and distributed Key-value nodes.

VIII. Conclusion

In this paper, we presented TransKV: a networking support for transaction processing in distributed key-value stores, that leverages the power and flexibility of the new programmable switches to act as a transaction manager. TransKV switches coordinate between the submitted transactions on the key-value pairs that are cached on the switch data plane, while TransKV controller manages the log for restoring the system to a consistent state after transactions failure. TransKV also takes the benefit of the data center's network structure to design a hierarchical caching mechanism on the switches. We believe that TransKV can be deployed on the programmable switches currently integrated in the data center's network to improve the performance of distributed key-value stores.

IX. Acknowledgement

This work was partially supported by NSF I/UCRC Center Research in Intelligent Storage and the following NSF awards 1439662, and 1812537.

REFERENCES

[1] Prog. switches in ATT: https://www.sdxcentral.com/articles/news/att-picks-barefoot-networks-programmable-switches/2017/04/.

[2] P4 Software Switch Website: http://www.bmv2.org/index.html.

[3] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming Protocol-Independent Packet Processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.

[4] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, page 143–154, 2010.

[5] Database Ranking categories listed by DB-engine: https://db-engines.com/en/ranking_categories.

[6] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon's Highly Available Key-Value Store. In *Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, page 205–220, 2007.

[7] Transactions at Amazon DynamoDB: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/transactions.html.

[8] Ramez Elmasri et al. *Fundamentals of Database Systems, 3rd Edition*. Addison-Wesley-Longman, 2000.

[9] Robert Escriva, Bernard Wong, and Emin Gün Sirer. Warp: Lightweight Multi-Key Transactions for Key-Value Stores. *CoRR*, abs/1509.07815, 2015.

[10] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The Road to SDN: An Intellectual History of Programmable Networks. *SIGCOMM Comput. Commun. Rev.*, 44(2):87–98, April 2014.

[11] Google Cloud using P4Runtime to build smart networks: https://cloud.google.com/blog/products/gcp/google-cloud-using-p4runtime-to-build-smart-networks.

[12] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. NOX: Towards an Operating System for Networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[13] Theo Jepsen, Leandro Pacheco de Sousa, Huynh Tu Dang, Fernando Pedone, and Robert Soulé. Gotthard: Network Support for Transaction Processing. In *Proceedings of the Symposium on SDN Research*, SOSR '17, page 185–186, 2017.

[14] Xin Jin, Xiaozhou Li, Haoyu Zhang, Nate Foster, Jeongkeun Lee, Robert Soulé, Changhoon Kim, and Ion Stoica. NetChain: Scale-Free Sub-RTT Coordination. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 35–49, Renton, WA, April 2018.

[15] Xin Jin, Xiaozhou Li, Haoyu Zhang, Robert Soulé, Jeongkeun Lee, Nate Foster, Changhoon Kim, and Ion Stoica. NetCache: Balancing Key-Value Stores with Fast In-Network Caching. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 121–136, 2017.

[16] Customers of Popular Key-value stores listed by DB-engine: https://db-engines.com/en/ranking/key-value+store.

[17] Avinash Lakshman and Prashant Malik. Cassandra: A Decentralized Structured Storage System. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, April 2010.

[18] LevelDB: A light-weight single-purpose library for persistent key-value store, https://github.com/google/leveldb.

[19] Jialin Li, Jacob Nelson, Ellis Michael, Xin Jin, and Dan R. K. Ports. Pegasus: Tolerating Skewed Workloads in Distributed Storage with In-Network Coherence Directories. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 387–406. USENIX Association, November 2020.

[20] Youjie Li, Iou-Jen Liu, Yifan Yuan, Deming Chen, Alexander Schwing, and Jian Huang. Accelerating distributed reinforcement learning with in-switch computing. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 279–291, 2019.

[21] Zaoxing Liu, Zhihao Bai, Zhenming Liu, Xiaozhou Li, Changhoon Kim, Vladimir Braverman, Xin Jin, and Ion Stoica. DistCache: Provable Load Balancing for Large-Scale Storage Systems with Distributed Caching. In *17th USENIX Conference on File and Storage Technologies (FAST 19)*, pages 143–157, Boston, MA, February 2019. USENIX Association.

[22] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[23] Mininet Website: http://mininet.org/.

[24] Rajesh Nishtala, Hans Fugal, Steven Grimm, Marc Kwiatkowski, Herman Lee, Harry C. Li, Ryan McElroy, Mike Paleczny, Daniel Peek, Paul Saab, David Stafford, Tony Tung, and Venkateshwaran Venkataramani. Scaling Memcache at Facebook. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 385–398, Lombard, IL, April 2013.

[25] John Ousterhout, Parag Agrawal, David Erickson, Christos Kozyrakis, Jacob Leverich, David Mazières, Subhasish Mitra, Aravind Narayanan, Guru Parulkar, Mendel Rosenblum, Stephen M. Rumble, Eric Stratmann, and Ryan Stutsman. The Case for RAMClouds: Scalable High-Performance Storage Entirely in DRAM. *SIGOPS Oper. Syst. Rev.*, 43(4):92–105, January 2010.

[26] P4Runtime: https://p4.org/p4-runtime/.

[27] Python Interface for LevelDB: https://plyvel.readthedocs.io/en/latest/.

[28] S. Sanfilippo and P. Noordhuis, Redis: in-memory Key-value store, http://redis.io, 2009.

[29] David P. Reed. Implementing Atomic Actions on Decentralized Data. *ACM Trans. Comput. Syst.*, 1(1):3–23, February 1983.

[30] RocksDB. A facebook fork of leveldb which is optimized for flash and big memory machines, 2013. https://rocksdb.org.

[31] Scapy: https://scapy.readthedocs.io/en/latest/.

[32] V. Srinivasan, Brian Bulkowski, Wei-Ling Chu, Sunil Sayyaparaju, Andrew Gooding, Rajkumar Iyer, Ashish Shinde, and Thomas Lopatic. Aerospike: Architecture of a Real-Time Operational DBMS. *Proc. VLDB Endow.*, 9(13):1389–1400, September 2016.

[33] Qing Wang, Youyou Lu, Erci Xu, Junru Li, Youmin Chen, and Jiwu Shu. Concordia: Distributed Shared Memory with In-Network Cache Coherence. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*, pages 277–292. USENIX Association, February 2021.

[34] Hao Wen, Zhichao Cao, Yang Zhang, Xiang Cao, Ziqi Fan, Doug Voigt, and David Du. JoiNS: Meeting Latency SLO with Integrated Control for Networked Storage. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 194–200, 2018.