Fast RDMA-based Ordered Key-Value Store using Remote Learned Cache

Xingda Wei, Rong Chen, Haibo Chen
KVS: key pillar for distributed systems

Important building block for

- Databases, GraphStore
- Web applications
- Cloud infrastructures
- Serverless platforms
KVS: key pillar for distributed systems
KVS: key pillar for distributed systems

Client

Server KVS

B+Tree

Key-values

Network

CPU

RNIC

CPU

RNIC
Traditional KVS uses RPC (Server-centric)

Client

Get($K_1$)

CPU

Server KVS

$B^+$-Tree

$(K_1,V_1)$

Key-values

CPU

RNIC

Network

RNIC

CPU
Traditional KVS uses RPC (Server-centric)

Client

Server KVS

B+Tree

Key-values

CPU

RNIC

Network

RNIC

CPU

Get(K₁)

(K₁, V₁)
Traditional KVS uses RPC (Server-centric)

Client

\[ \text{Get}(K_1) \]

Server KVS

\[ (K_1, V_1) \]

Network

CPU

RNIC

B\(^+\)Tree

Key-values
Traditional KVS uses RPC (Server-centric)

Client

Get\( (K_1) = V_1 \)

Server KVS

\( (K_1,V_1) \)

CPU

RNIC

Network

RNIC

Key-values

B\(^+\)Tree
Traditional KVS uses RPC (Server-centric)
Server CPU is becoming the bottleneck

Increasing **CPU-NIC gap**

NIC’s speed is growing faster!

![Graph showing CPU-NIC gap over time](image)

**Huge CPU cost with random reads**

[1] Credits: StRoM: Smart Remote Memory @ Eurosys'20
Opportunity: one-sided RDMA (Client-direct)

NIC directly reads/writes memory

- Offload index traversal to NIC
- Totally bypass server CPU

Get($K_1$) = $V_1$
Opportunity: one-sided RDMA (Client-direct)

NIC directly reads/writes memory

- Offload index traversal to NIC
- Totally bypass server CPU

Get(K₁) = V₁
Challenge: limited NIC abstraction

NIC only has simple abstractions

● e.g., memory read/write

Works well for simple index structure

● e.g. HashTable, $O(1)$ network RTT\(^1\)

Inferior for complex index structure

● e.g., $B^+\text{Tree}, O(\log(n))$\(^2\) network RTT

\(^1\) RTT: roundtrip time
\(^2\) n: the scale of the KVS
Challenge: limited NIC abstraction

Client

Get(K₁)

CPU

RNIC

Network

Server KVS

B⁺Tree

Key-values

(K₁,V₁)

(K₁,V₁)
Challenge: limited NIC abstraction

Client

Server KVS

(B+Tree

Key-values

Get(K1)

NIC can only read one level use one RTT!

CPU

RNIC

Network

RNIC

(K0,V0)
Challenge: limited NIC abstraction

Lookup uses multiple RTTs
e.g., ~7 for a 100M KVS

Get(K₁) = V₁
Existing systems adopt caching

Get(K₁)

CPU

RNIC

Network

RNIC

Server KVS

B⁺Tree

Key-values

Client
Existing systems adopt caching

Get(K₁)

Client

Cache Tree at clients

☞ FaRM@SOSP’15, SIGMOD’19
☞ Cell@ATC’16

Cache hash table

☞ DrTM@SOSP’15
Existing systems adopt caching

Cache Tree at clients

Cache hash table

B+Tree index has huge client memory cost!

Get(K₀)

19 roundtrips

Inefficient!
High cache miss cost for caching tree

Tree node size can be much larger than the KV

- e.g., 1K vs. 8B

Recursive invalidation under insertions

- When cache more tree layers

![Graph showing throughput vs. time for different cache configurations]

More layer cached
Trade-off of existing KVS

Server-centric KVS

High CPU utilizations
Trade-off of existing KVS

Server-centric KVS

')->__('High CPU utilizations

Client-direct KVS

ปกครอง: Poor performance
Trade-off of existing KVS

Server-centric KVS

- High CPU utilizations

Client-direct KVS

- Poor performance

Client-direct KVS + cache

- High memory usage
Trade-off of existing KVS

Server-centric KVS
- High CPU utilizations

Client-direct KVS
- Poor performance

Client-direct KVS + cache
- High memory usage

Can we achieve all these properties?
Trade-off of existing KVS

Server-centric KVS

edis: High CPU utilizations

Client-direct KVS

edis: Poor performance

Client-direct KVS + cache

edis: High memory usage

![Diagram showing trade-offs between server-centric, client-direct, and client-direct + cache approaches, with high CPU utilizations and poor performance as drawbacks. The client memory and server CPU axes highlight the performance trade-offs.]
Overview of XSTORE

Hybrid architecture [1]

 kurs: Sever-centric updates
 kurs: Because one-sided has simple semantic

\(O(1)\) Client-direct Get, Scan

[1] Similar to existing RDMA-based KVS, e.g., FaRM@SOSP’15, Cell@ATC’16
Our approach: Learned cache

Using **ML** as the cache structure for tree-based index

Motivated by the *learned index*[^1]

🪛 Replace *index traversal* with *calculation*

🪛 The ML model can be *orders of magnitude smaller* than tree

[^1]: The case for the learned index @ SIGMOD'18
Client-direct Get() using learned cache

Learn a mapping of Address = $B^+\text{Tree}(\text{key})$

Get($K_1$)
Client-direct Get() using learned cache

Learned model with *small memory*

Get(K₁) → CPU

Server KVS

\[ B^+ \text{Tree} \]

Learned models

Key-values

Network
Client-direct Get() using learned cache

Client

Get($K_1$) $K_1$ [0,1]

CPU

RNIC

Server KVS

B$^+$Tree

Key-values

uego to be in [0,1]

Learned model assumes a sorted array
Client-direct Get() using learned cache

![Diagram of client-direct Get() using learned cache](image-url)
Client-direct Get() using learned cache

Client

Get($K_1$)

[0,1]

CPU

Server KVS

B+Tree

Key-values

$(K_1,V_1)$

0 1 2 3

Network

RNIC

RNIC
Client-direct Get() using learned cache

Client

Get\(K_1\)

CPU

\(\{(K_0,0), (K_1,1)\}\)

Network

Server KVS

\(B^+\)Tree

\(\{(K_1,V_1)\}\)

Key-values
Client-direct Get() using learned cache

Client

Get($K_1$)

(K_0,0), (K_1,1)

Addr of $K_1 = 1$

CPU

Server KVS

B+Tree

B + Tree

(K_1,V_1)

0 1 2 3

Key-values

Network

RNIC

RNIC
Client-direct Get() using learned cache

Get($K_1$) = $V_1$

Client

Server KVS

B+Tree

Key-values

CPU

RNIC

Network

RNIC

1

$(K_1, V_1)$

0 1 2 3
Benefits of the learned cache

1. 1 RTT for lookup
2. Small memory footprint

Get($K_1$) = $V_1$

CPU → RNIC → Network → RNIC → Server

Client

B+Tree

Key-values

#1 1 RTT for lookup

#2 Small memory footprint
Challenges of learned cache

Dynamic insertions/deletions?
Learned model assumes a sorted array

Get($K_1$) = $V_1$

Client

Server KVS

0 1 2 3

Key-values

CPU

RNIC

Network

RNIC

($K_1, V_1$)
Outline of the remaining content

Server-side data structure for dynamic workloads

Client-side learned cache & TT

Performance evaluation of XSTORE
XSTORE stores KVs in B+Tree leaf nodes

Dynamic insertions/deletions?
Learned index assumes a sorted array

Client

Server KVS

B+Tree

Key-values

Get(K₁)

CPU

RNIC

Network
Models cannot learn dynamic B$^+$Tree address

Can only learn when the addresses are *sorted*

Not the case for dynamic B$^+$Tree

![Diagram](image)
Solution: another layer of indirection

Observation: leaf nodes are logically sorted

💡 Assign logical addresses to leaf nodes

ML: key → logical

💡 Translation table (TT): logical → physical

Translation Table

<table>
<thead>
<tr>
<th>Logical addresses</th>
<th>Physical addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0</td>
</tr>
<tr>
<td>0x10</td>
<td>1</td>
</tr>
<tr>
<td>0x20</td>
<td>2</td>
</tr>
</tbody>
</table>
Outline of the remaining content

Server-side data structure for dynamic workloads

Client-side learned cache & TT

Performance evaluation of XSTORE
Client-direct Get() using model & TT

Model & TT forms the cache

(K₁,V₁)

0x00 0x10 0x20

CPU

Client

RNIC

Network

RNIC

B+Tree
Client-direct Get() using model & TT

Client

Get($K_1$) $K_1$

TT

[0,1]

Server KVS

B+Tree

(RNIC)

CPU

0x00 0x10 0x20

(K$_1$,V$_1$)

Network
Client-direct Get() using model & TT

Client

Client

CPU

Get($K_1$)

$K_1$

Server KVS

Logical addresses

$[0,1]$

Network

CPU

RNIC

RNIC

Network

$0x00$

$0x10$

$0x20$

$B^+$ Tree

Get($K_1$)

$(K_1,V_1)$

Server KVS

RNIC

Network
Client-direct Get() using model & TT
Client-direct Get() using model & TT

Client

Get(K₁)

CPU

[0x00,0x10]

[0,1]

Server KVS

B⁺Tree

Still one roundtrip to look up the address of K₁

Network

RNIC

RNIC

0x00

0x10

0x20
Model retraining

Model is retrained at server in background threads

△: Small cost & extra CPU usage at the server

XSTORE uses a two-layer RMI to organize models[1]

△: *Fine-grained* model retraining

[1] Recursive Model Inference, following “The case for the learned index @ SIGMOD’18”
Stale model handling

Background update causes *stale learned models*

But stale learned models & TT could correctly *find most keys*

いただける場合、キーを *not moved* にしない限り、スレッドモデルとTTも正しいキーを検出できる

キー ➔ ロジカル ➔ フィジカル
Many other design details & optimizations

Server-side operations

Find *non-trained* keys

**Optimizations** of speculative execution

Dynamic model expansion

Fault tolerance of XSTORE

Scale-out XSTORE
Evaluation of XSTORE

We answer the following questions:

❓ Comparing to server-centric designs?

❓ Comparing to client-direct designs?

❓ Does XStore provide better trade-off?

![Diagram showing comparison of server-centric, client-direct, and client-direct + cache designs in terms of performance and client memory usage.](diagram.png)
Performance of XSTORE on YCSB

100M KVs, uniform workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(S)/(U/I)</td>
<td>50:50</td>
<td>90:10</td>
<td>100:0</td>
<td>95:5</td>
<td>95:5</td>
</tr>
</tbody>
</table>

[*] Read, Scan, Update, Insert
Performance of XSTORE on YCSB

100M KVs, uniform workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(S)/(U/I)</td>
<td>50:50</td>
<td>90:10</td>
<td>100:0</td>
<td>95:5</td>
<td>95:5</td>
</tr>
</tbody>
</table>

[*] Read, Scan, Update, Insert

DrTM+Tree@Eurosys'16
EMassTree@NSDI'19
Cell@ATC'16

XSTORE

40M reqs/s/NIC
Performance of XSTORE on YCSB

100M KVs, uniform workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(S)/(U/I)</td>
<td>50:50</td>
<td>90:10</td>
<td>100:0</td>
<td>95:5</td>
<td>95:5</td>
</tr>
</tbody>
</table>

Bottlenecked by **server CPU**.
Performance of XSTORE on YCSB

100M KVs, uniform workloads

**Traversing B+Tree** with one-sided RDMA is *costly!*

<table>
<thead>
<tr>
<th>Workload</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R(S)/(U/I)</td>
<td>50:50</td>
<td>90:10</td>
<td>100:0</td>
<td>95:5</td>
<td>95:5</td>
</tr>
</tbody>
</table>

[*] Read, Scan, Update, Insert
The XCache in details

For a 100M KVs YCSB dataset

⁻ 500K Linear regression as models, each 14B
⁻ ~ 8μs to retrain each model
⁻ ~ 8s to train the entire cache

#models = 500K
The XCache in details

For a 100M KVs YCSB dataset

- **500K** Linear regression as models, each 14B
- ~ 8μs to retrain each model
- ~ 8s to train the entire cache

Small model to fit the dataset
The XCache in details

For a 100M KVs YCSB dataset

1️⃣: 500K Linear regression as models, each 14B

2️⃣: ~ 8μs to retrain each model

3️⃣: ~ 8s to train the entire cache

Quick retrain under dynamic workload
Sensitive to the dataset

Different dataset has different accuracy

May affect the performance

Throughput drop due to increased error for complex dataset

<table>
<thead>
<tr>
<th>Name</th>
<th>Workloads</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>e.g., YCSB, TPC-C</td>
</tr>
<tr>
<td>Noised Linear</td>
<td>e.g., YCSB</td>
</tr>
<tr>
<td>Open street map</td>
<td>e.g., OpenStreetMap</td>
</tr>
</tbody>
</table>

Peak throughput (100M dataset)

Average latency (μs)
Learned cache vs. Tree-based cache

XStore provides better *memory-performance trade-off*

- YCSB-C uniform workload

![Graph showing peak throughput for XStore and Tree-index with client memory usage.]

- **40M/s** with **16B cache**
- **150MB vs. 600MB**
Current limitations and future work

XSTORE currently only supports fixed-length keys

- Our paper describes our plan to support variable-length keys

Focus on simple models (e.g., LR)

- Efficient upon retraining under dynamic workloads
- May results in huge error for complex data distribution
- Trade-off: retraining speed vs. accuracy vs. memory

Orthogonal to the design of XSTORE
Conclusion

XSTORE provides a new design for RDMA-enabled KVS

- First adopts the learned models for one-sided RDMA READ

XSTORE provides better trade-offs:

- Server-side CPU vs. Client-side memory vs. Performance

Please check XSTORE@

- https://ipads.se.sjtu.edu.cn/projects/xstore

Thanks & QA