NetCache: Balancing Key-Value Stores with Fast In-Network Caching

COS 316: Principles of Computer System Design

Lecture 10

Amit Levy & Jennifer Rexford

1

What is a Key-Value Store?

Memcached

redis

amazon
DynamoDB

2

Key-Value Store: A Very Simple Database

- Hash table
  - key: an arbitrary string (e.g., “Joe”, ”jrex.jpg”)
  - value: unstructured data (e.g., number, string, complex object)
- Primitive operations
  - put(key, value)
  - value = get(key)
  - delete(key)
- A “dictionary”
  - Array, map…

key	value
“system”	“a set of things working together as parts of a mechanism or an interconnecting network.”
“internet”	“the single worldwide computer network that interconnects other computer networks”
“database”	“a comprehensive collection of related data organized for convenient access, generally in a computer.”
“cache”	“ a temporary storage space or memory that allows fast access to data”

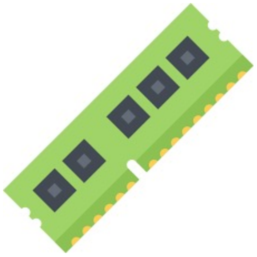
Applications of Key-Value Stores

- When to use a key-value store
  - Real-time, random data access
  - Caching for frequently accessed data
  - Simple applications
- Many example applications
  - Online shopping: user id (key) and shopping cart (value)
  - User profile: user id (key) and preferences (value)
  - Movies/music: file name (key) and file contents (value)
  - Blockchain: hash value (key) and block entity (value)

…

High-Speed Key-Value Store

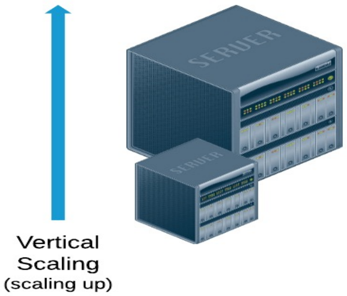
- A need for speed
  - “web pages routinely fetch thousands of key-value pairs” [Facebook]
  - Systems with many users
- A key-value store needs resources
  - Storage: to store many key-value pairs
  - Bandwidth: to handle many operations
- In-memory key-value store
  - Avoid reads and writes to slow disk-based systems
  - But, what if you need more storage and/or bandwidth?



5

Distributed, High-Speed Key-Value Store

- Vertical scaling
  - Add more resources to a server
  - E.g., more processing, memory, or network resources
- Horizontal scaling
  - Add more servers
  - Reduce the responsibilities of each server



6

Horizontal Scaling: Partitioning the Keys

- **Sharding**: breaking data up into partitions
  - Each server handles operations for a portion of the keys
  - Reduces query load and storage space on individual servers

key	value
"system"	"a set of things working together as parts of a mechanism or an interconnecting network."
"internet"	"the single worldwide computer network that interconnects other computer networks"
"database"	"a comprehensive collection of related data organized for convenient access, generally in a computer."
"cache"	" a temporary storage space or memory that allows fast access to data"

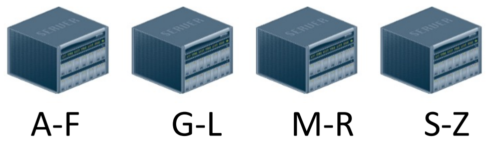
Server 0

Server 1

7

Goals in Partitioning the Keys

- Easy to direct requests to the right server
- Even distribution of data storage
- Even distribution of request load

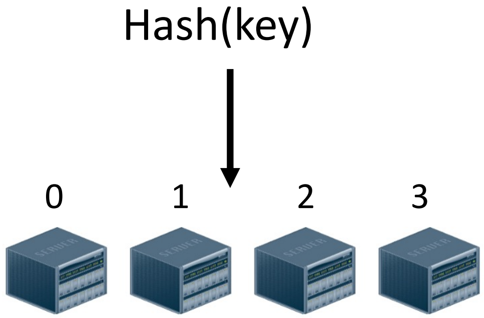


- E.g., range of keys
  - Alphabet: letters A-F, G-L, M-R, S-Z
  - Months: Jan-Mar, Apr-Jun, Jul-Sep, Oct-Dec

8

Better Design: Hash-Based Sharding

- Hashing on the key
  - Randomize and distribute requests
  - Easy to compute without coordination



- Achieves the goals?
  - Easy to direct requests to the right server?
  - Even distribution of data storage?
  - Even distribution of request load?

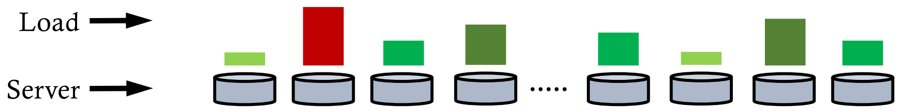
9

Distributed, In-Memory Key-Value Stores Need a Cache

10

Challenge: Skewed Workload

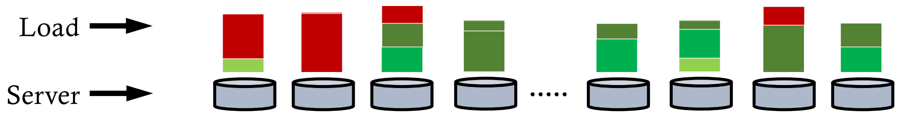
low throughput & **high tail latency**



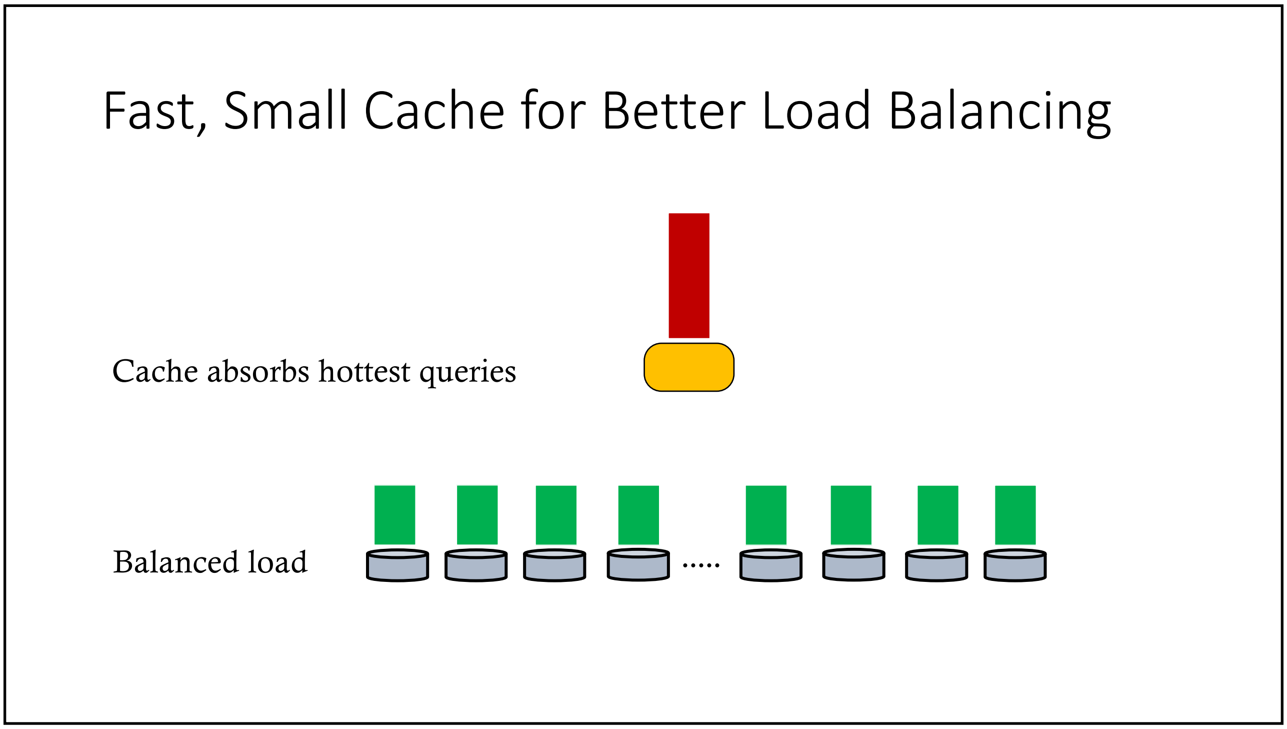
- Popular items receive far more queries than others
- E.g., “10% of items account for 60-90% of queries in the Memcached deployment at Facebook”

Challenge: Dynamic Workloads

low throughput & **high tail latency**



- The popular items change rapidly
- E.g., popular posts, limited-time offers, trending events

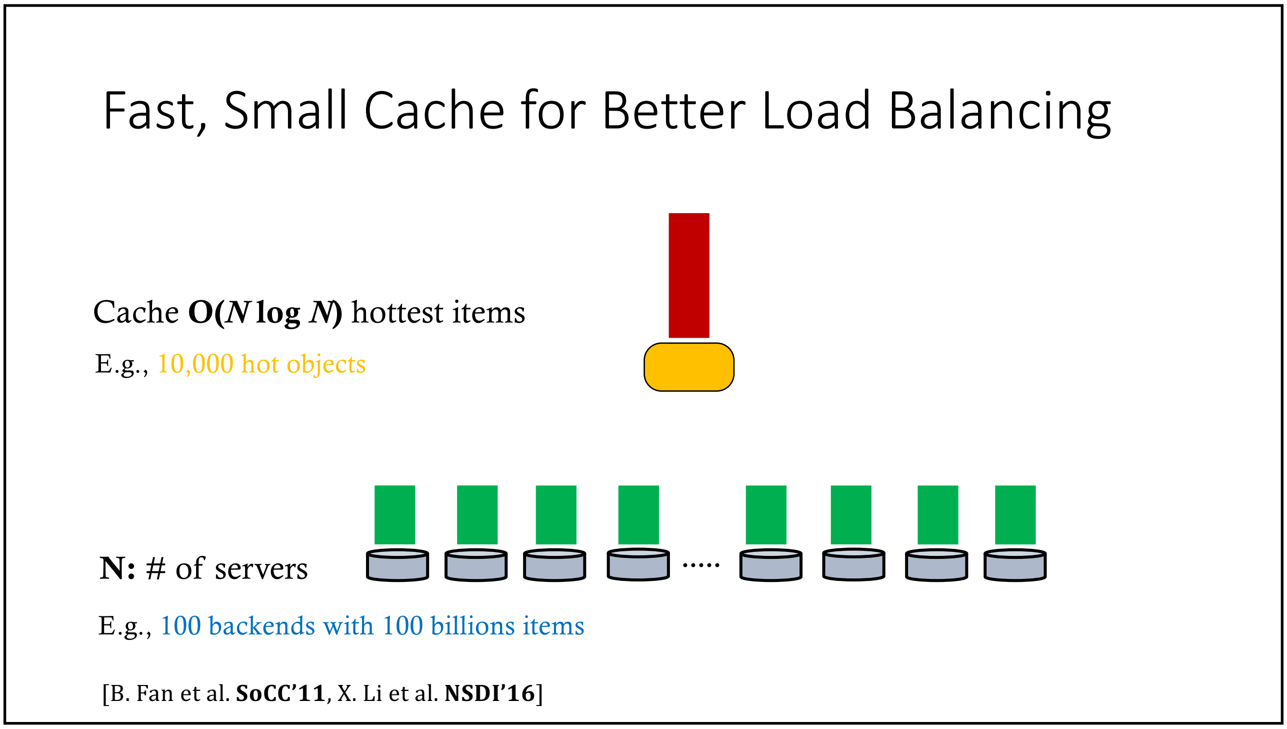


Fast, Small Cache for Better Load Balancing

Cache absorbs hottest queries

Balanced load

13



Fast, Small Cache for Better Load Balancing

Cache **O(*N* log *N*)** hottest items

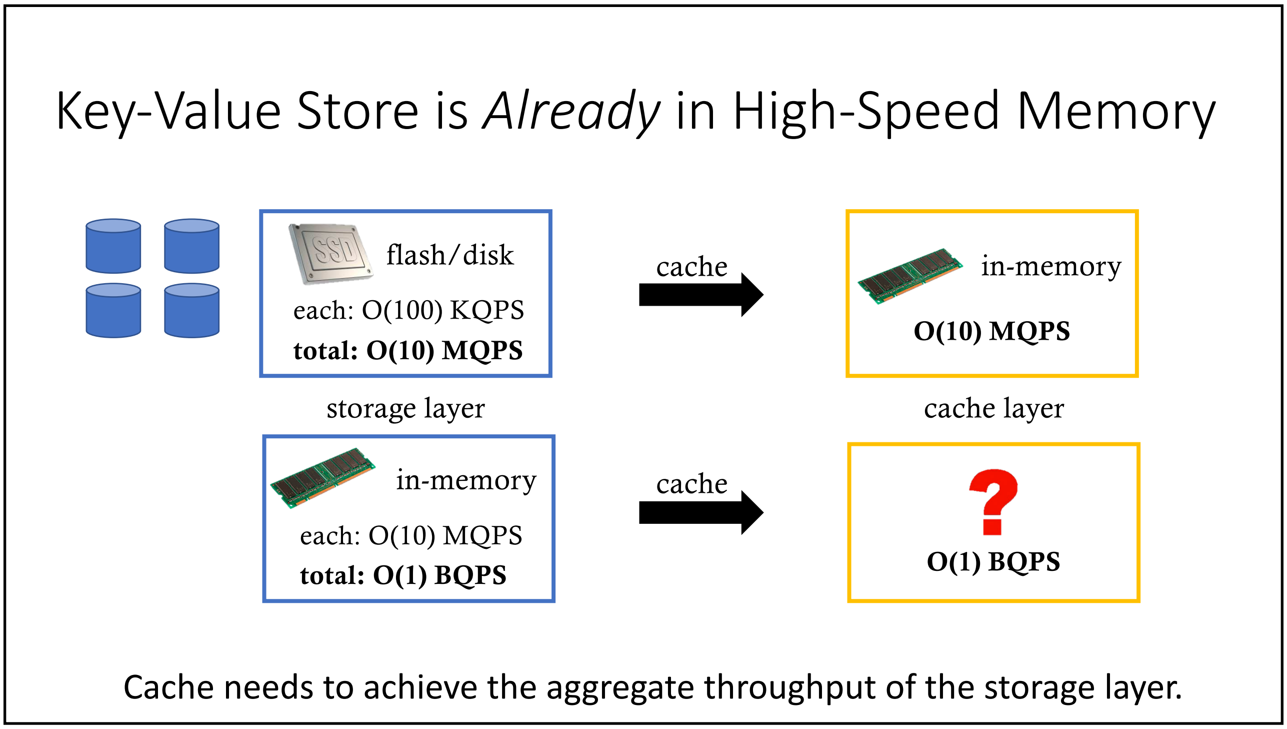
E.g., 10,000 hot objects

N: # of servers

E.g., 100 backends with 100 billions items

[B. Fan et al. **SoCC'11**, X. Li et al. **NSDI'16**]

14



Key-Value Store is *Already* in High-Speed Memory

flash/disk

each: O(100) KQPS

total: O(10) MQPS

storage layer

cache

in-memory

O(10) MQPS

cache layer

in-memory

each: O(10) MQPS

total: O(1) BQPS

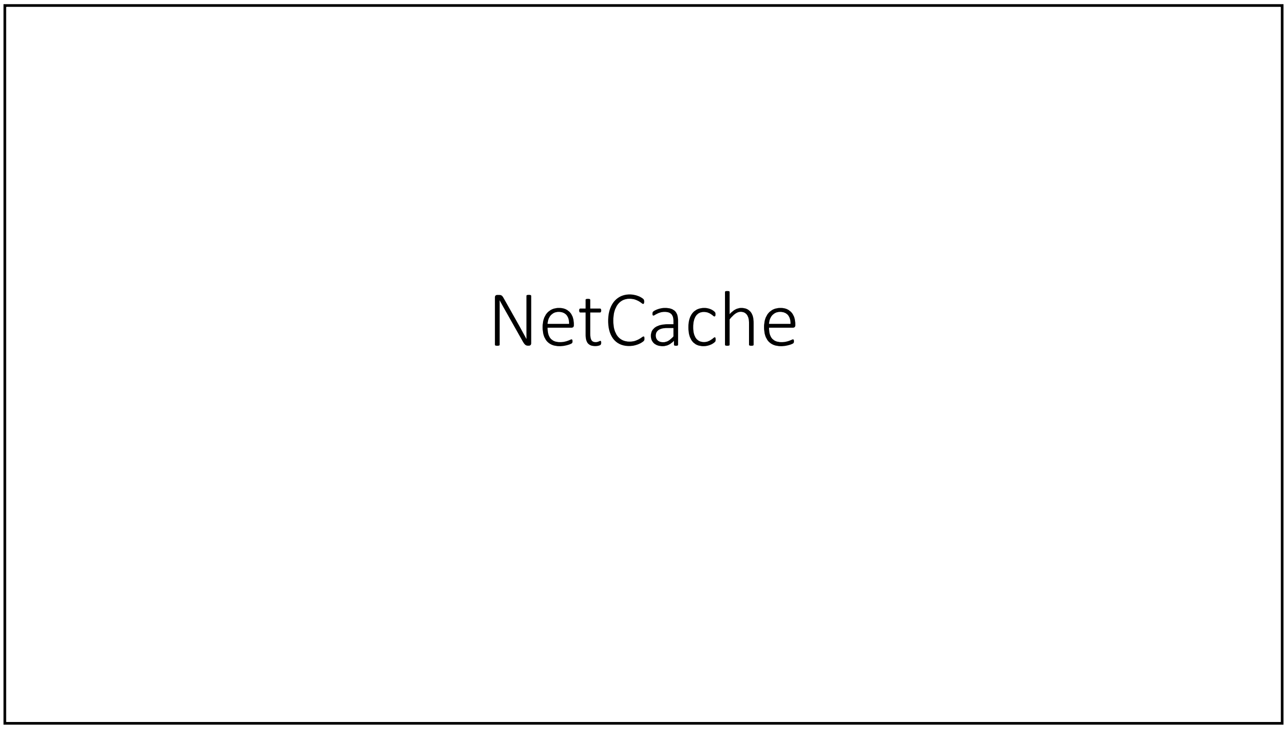
cache

?

O(1) BQPS

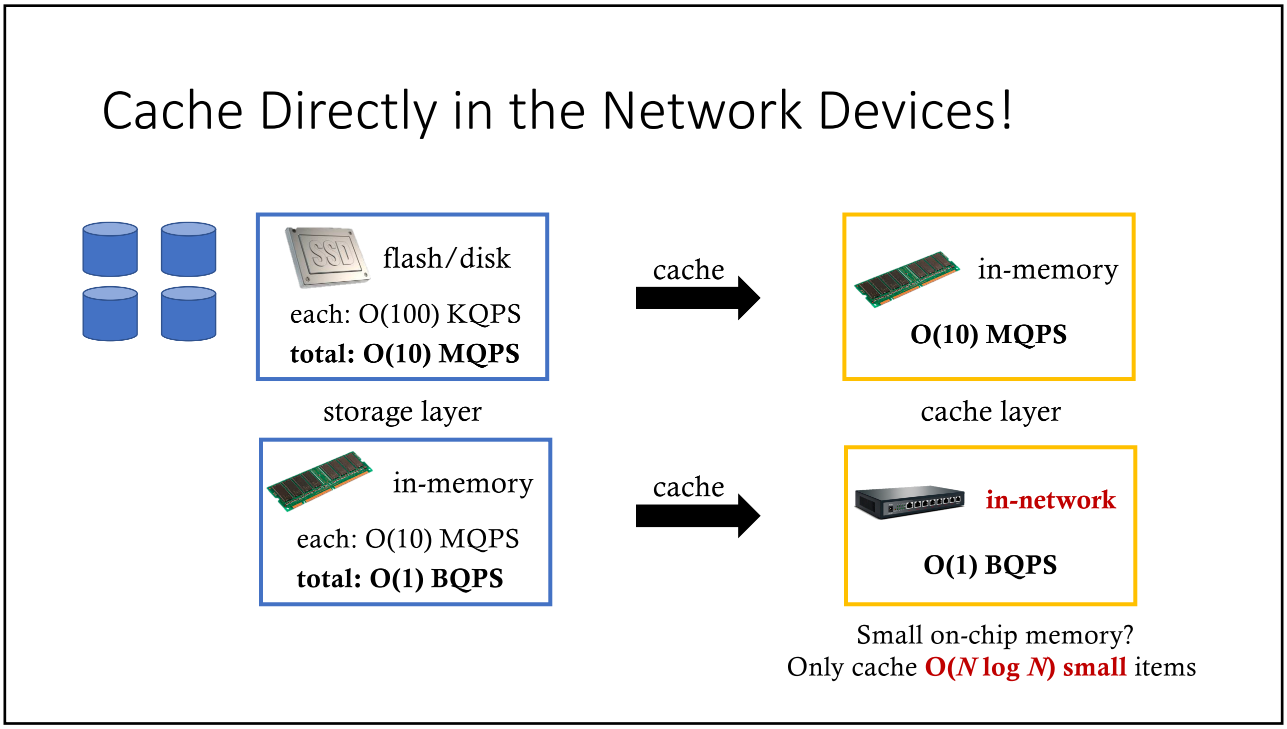
Cache needs to achieve the aggregate throughput of the storage layer.

15



NetCache

16



Cache Directly in the Network Devices!

flash/disk

each: O(100) KQPS

total: O(10) MQPS

storage layer

cache

in-memory

O(10) MQPS

cache layer

in-memory

each: O(10) MQPS

total: O(1) BQPS

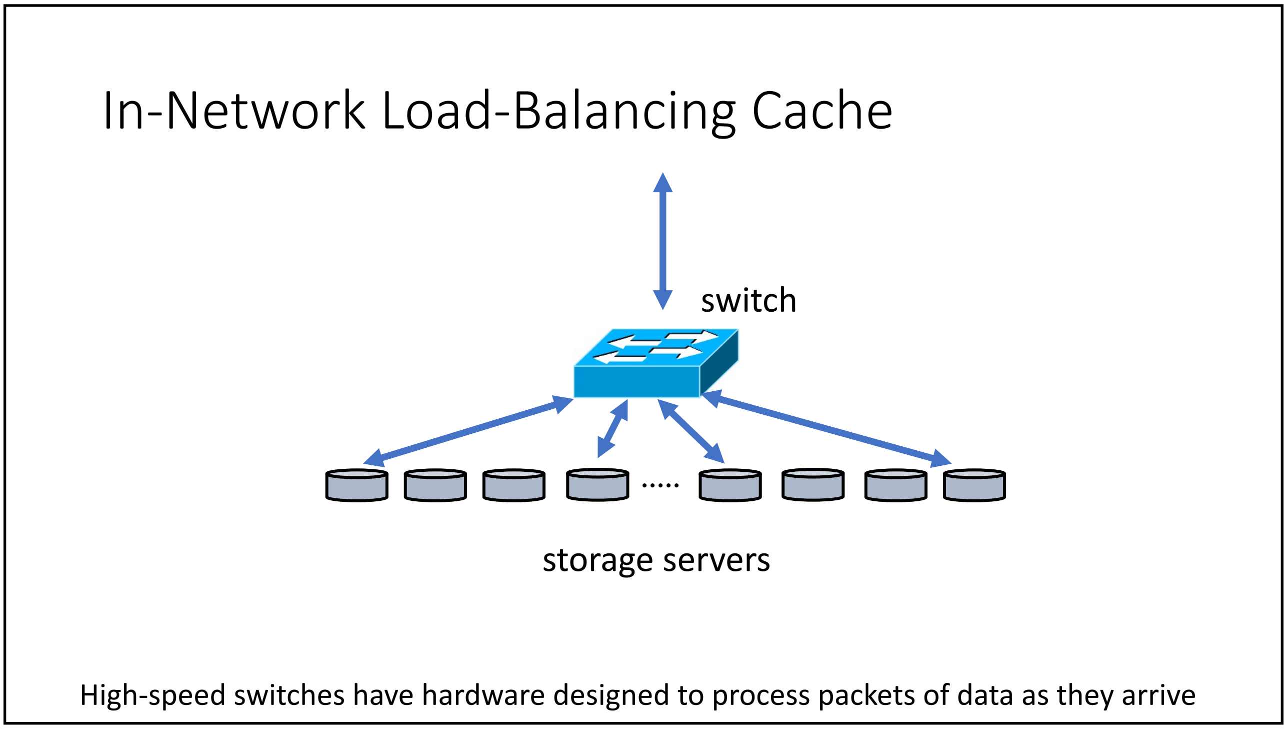
cache

in-network

O(1) BQPS

Small on-chip memory?
Only cache **O(*N* log *N*) small** items

17



In-Network Load-Balancing Cache

switch

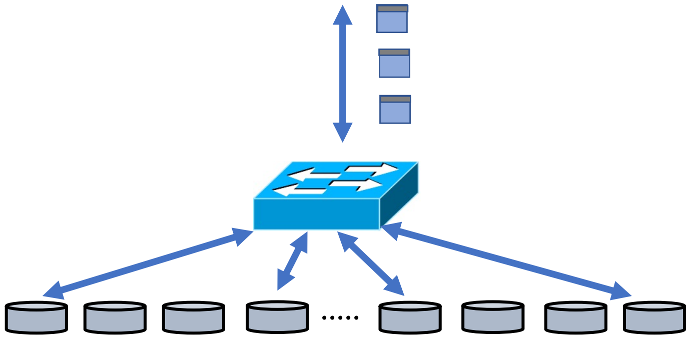
.....

storage servers

High-speed switches have hardware designed to process packets of data as they arrive

18

Individual Packets Carry the Messages

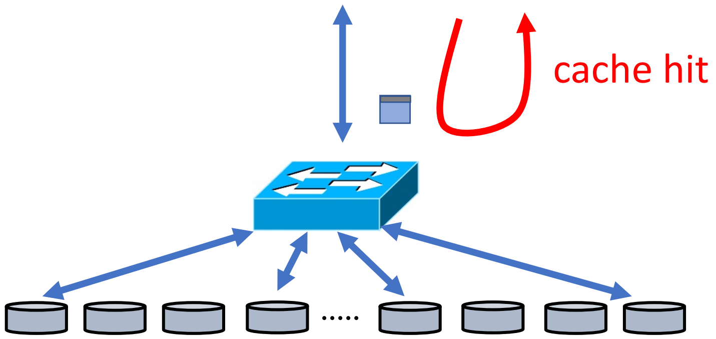


ETH	IP	TCP/UDP	OP	SEQ	KEY	VALUE

get, put, or delete

19

Cache Hit and Cache Miss



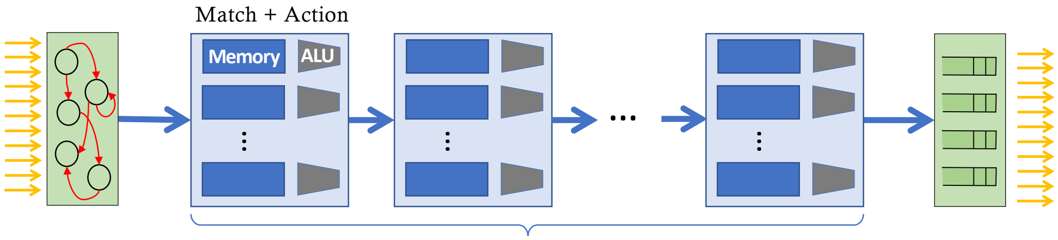
- Cache hit: switch responds directly
- Cache miss: switch forwards to the server, which responds

20

Programmable Packet-Processing Hardware

- Modern switches are increasingly programmable
  - Parsing of packet headers
  - Match-action processing on header fields
  - Registers for storing data across packets

www.p4.org



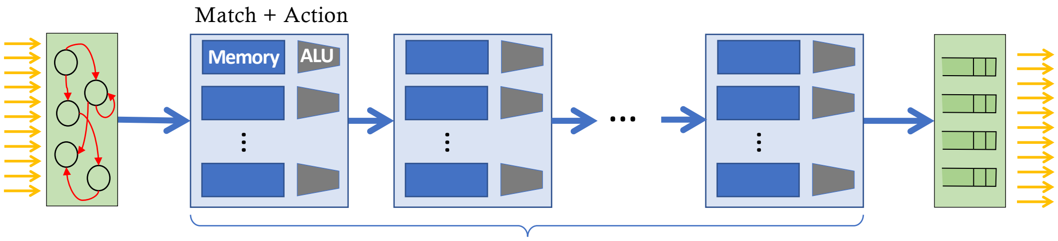
Programmable parser

Programmable match-action pipeline

21

NetCache Uses Programmable Switches

- Programmable parser
  - Parse custom key-value fields in the packet
- Programmable match-action pipeline
  - Read and update key-value data
  - Maintain query statistics for cache updates

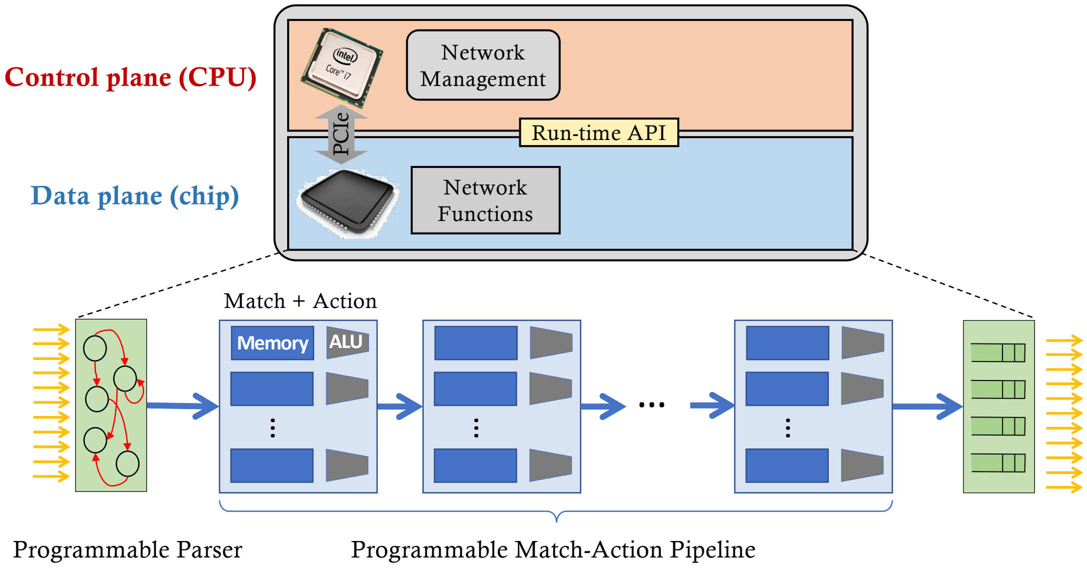


Programmable Parser

Programmable Match-Action Pipeline

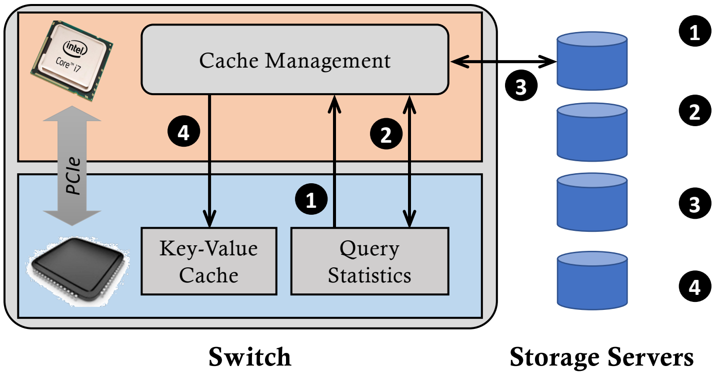
22

Control Plane of Programmable Switches



23

Cache Eviction and Replacement



❶ Data plane reports hot keys

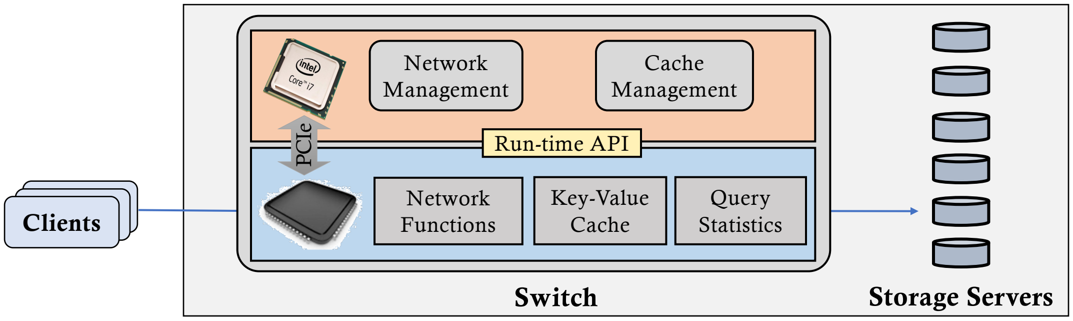
❷ Control plane compares loads of new hot and sampled cached keys

❸ Control plane fetches values for keys to be inserted to the cache

❹ Control plane inserts and evicts keys

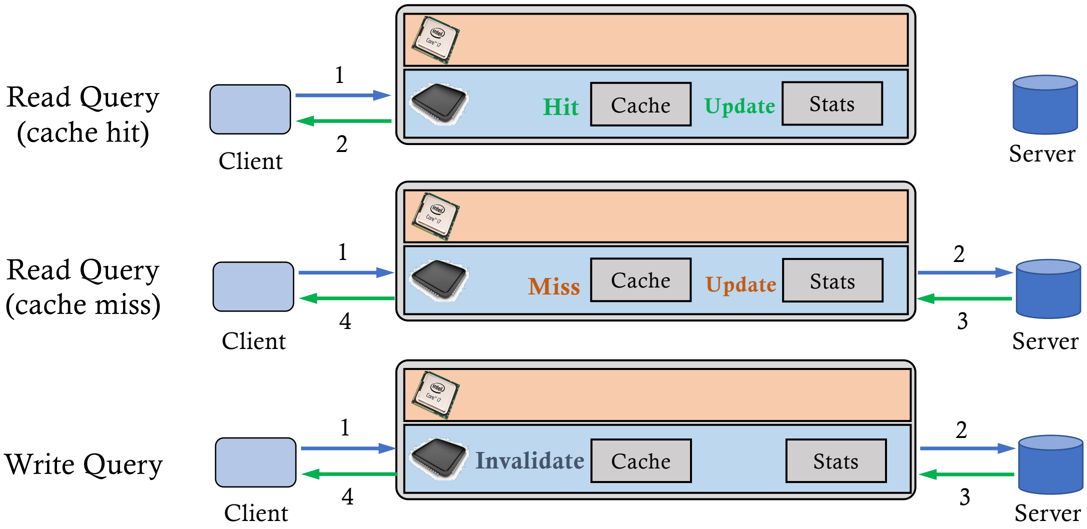
24

NetCache Architecture



- Switch data plane
  - Key-value store to serve queries for cached keys
  - Query statistics to enable efficient cache updates
- Switch control plane
  - Insert hot items into the cache and evict less popular items
  - Manage memory allocation for on-chip key-value store

Data-Plane Query Handling



NetCache Query Statistics

27

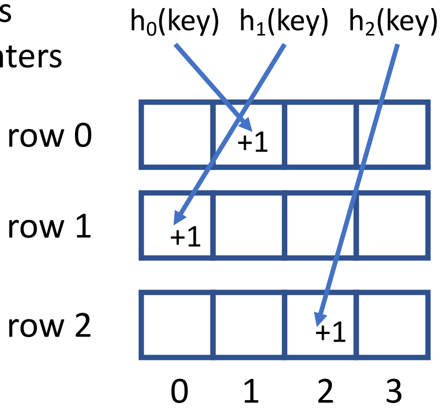
Query Statistics in the Data Plane

- For cached keys
  - Count the number of accesses to each cached key
  - … so the control plane can decide which key-value pairs to *evict*
- For uncached keys
  - Identify new “hot” keys
  - … so the control plane can decide which key-value pairs to *insert*
  - Problem: huge number of uncached keys

28

Compact Data Structure: CountMin Sketch

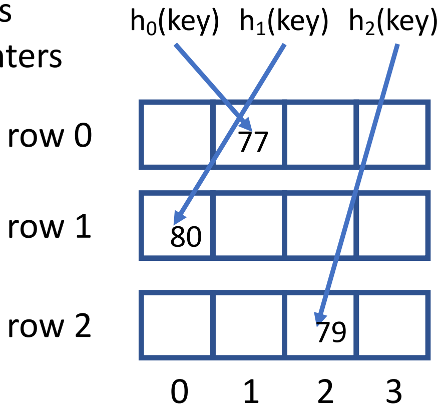
- Approximate counts without per-key memory
  - *r* rows, each with a hash function $h_i()$
  - *c* columns
  - *r* x *c* counters



29

Compact Data Structure: CountMin Sketch

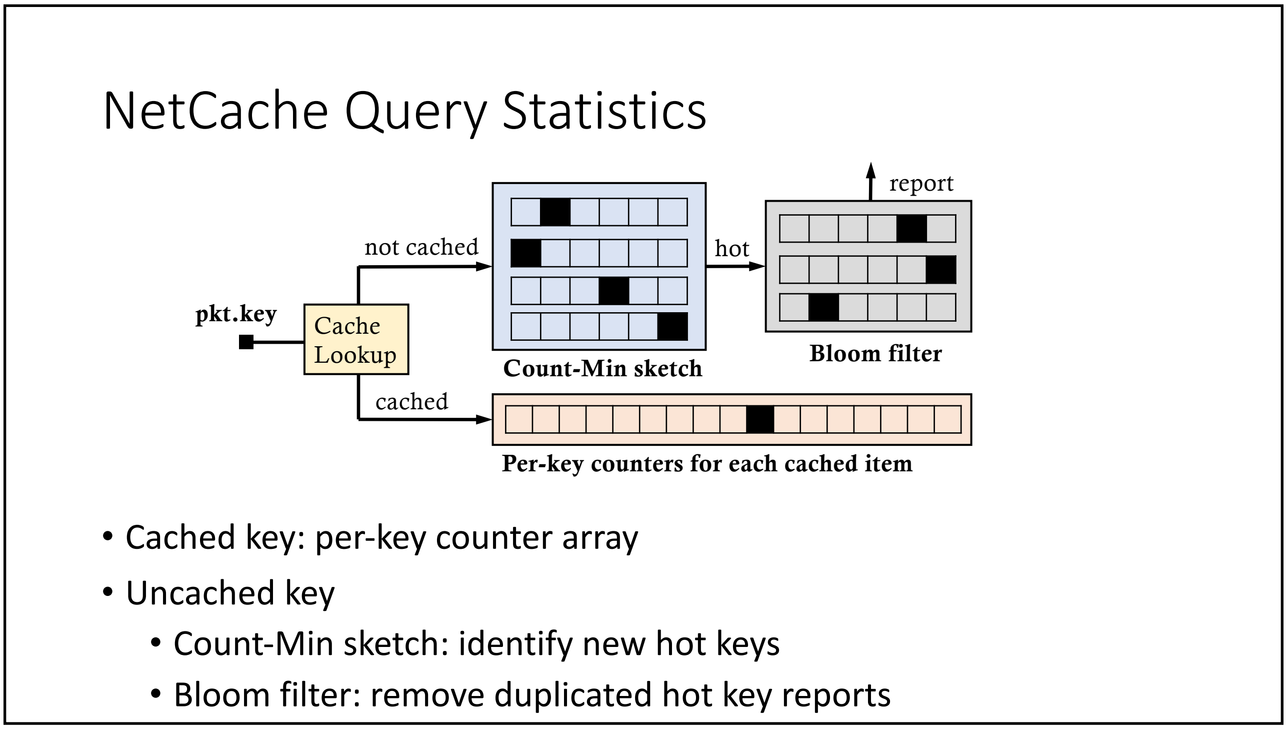
- Approximate answers with less memory
  - *r* rows, each with a hash function $h_i()$
  - *c* columns
  - *r* x *c* counters



Estimated count:
min{77, 80, 79} = 77

Report this hot key!

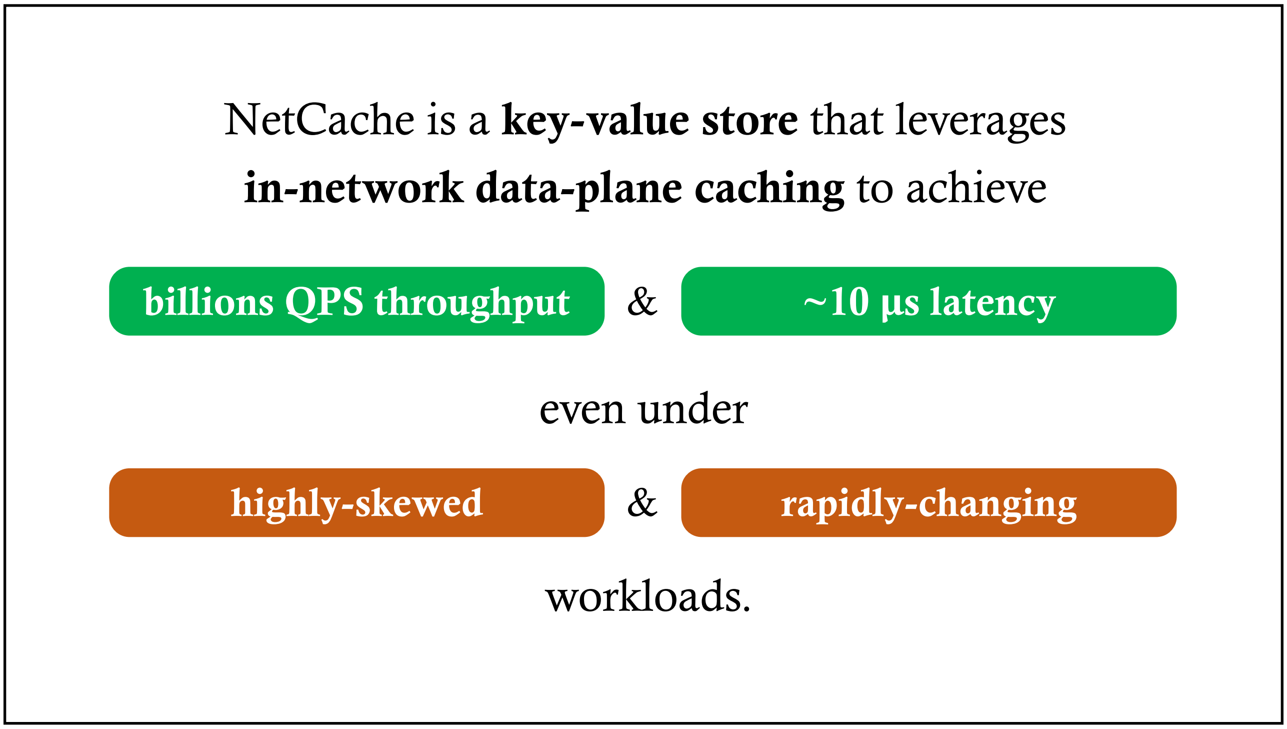
30



NetCache Query Statistics

- Cached key: per-key counter array
- Uncached key
  - Count-Min sketch: identify new hot keys
  - Bloom filter: remove duplicated hot key reports

31



NetCache is a **key-value store** that leverages **in-network data-plane caching** to achieve

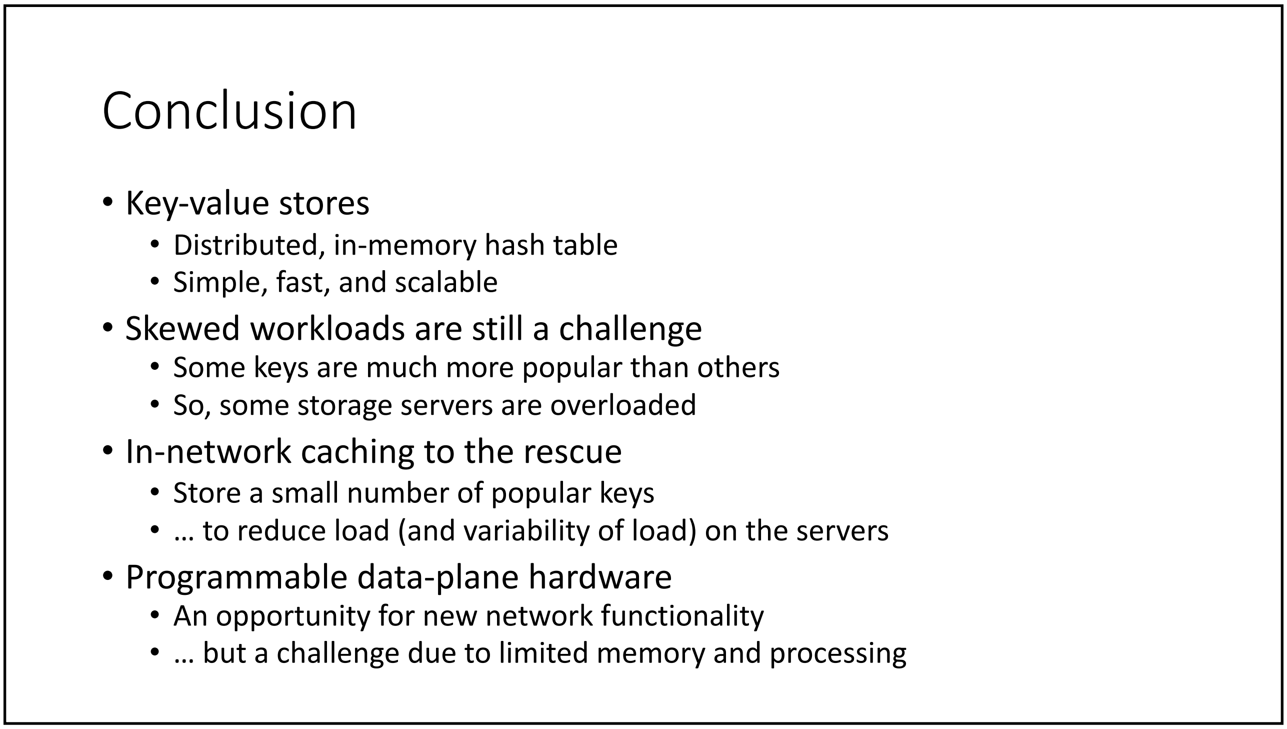
billions QPS throughput & **~10 µs latency**

even under

highly-skewed & **rapidly-changing**

workloads.

32



Conclusion

- Key-value stores
  - Distributed, in-memory hash table
  - Simple, fast, and scalable
- Skewed workloads are still a challenge
  - Some keys are much more popular than others
  - So, some storage servers are overloaded
- In-network caching to the rescue
  - Store a small number of popular keys
  - … to reduce load (and variability of load) on the servers
- Programmable data-plane hardware
  - An opportunity for new network functionality
  - … but a challenge due to limited memory and processing

33