Achieving High Throughput and Elasticity in a Larger-than-Memory Store

Chinmay Kulkarni, Badrish Chandramouli*, Ryan Stutsman
University of Utah, *Microsoft Research

One Slide Summary

Multi-core optimized single-node key-value stores are emerging

- ➔ **Very** high throughput ~**100 Million events/sec/server.** Ex: FASTER (SIGMOD'18)

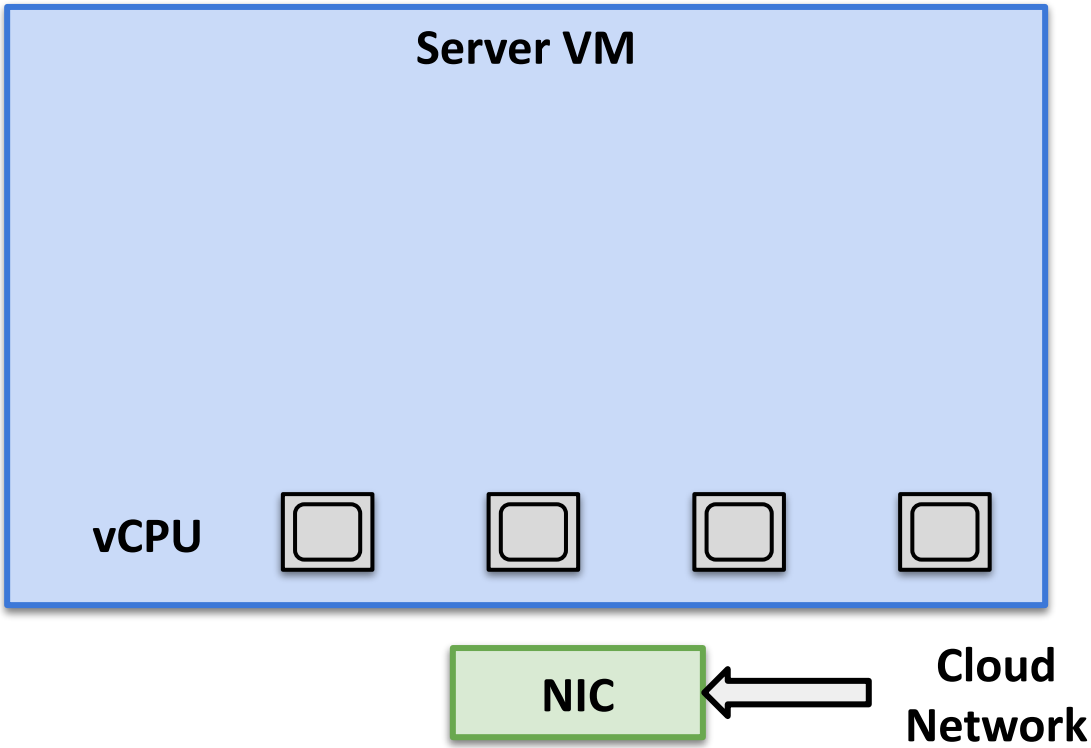
Problem: Retain high throughput in the public cloud

- ➔ Avoid request dispatch and network bottlenecks
- ➔ Support reconfiguration/migration while preserving throughput

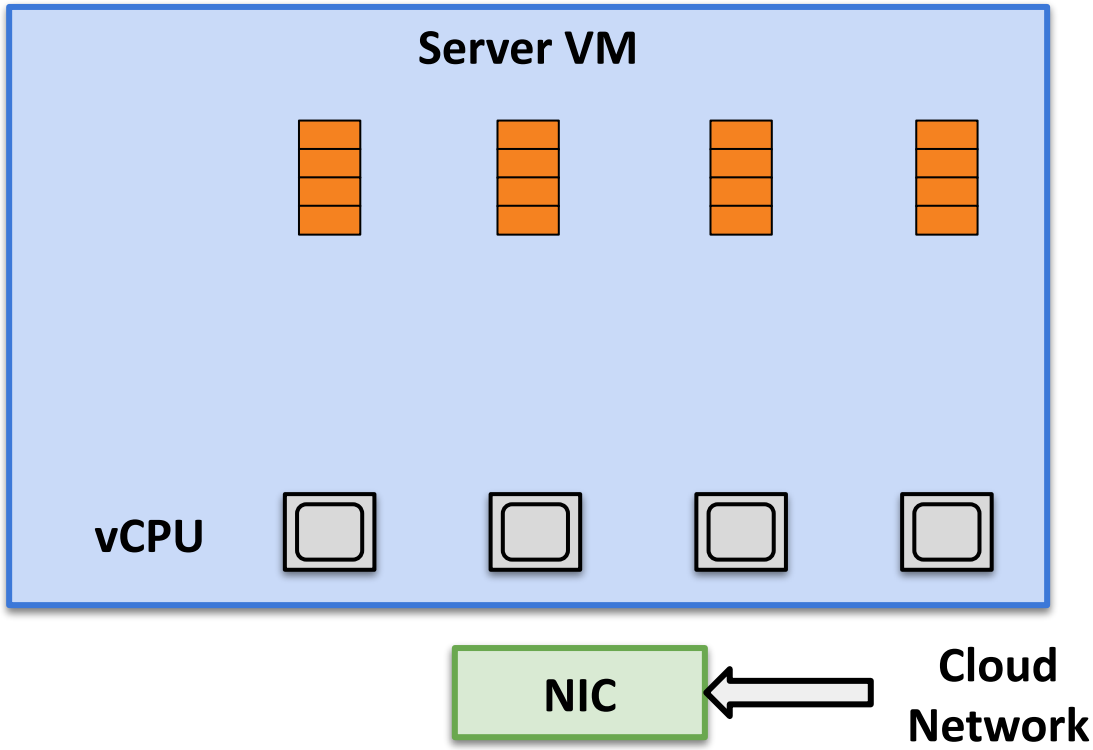
Shadowfax: Distributed key-value store built on FASTER

- ➔ 100 Million events/sec/VM **on Azure**
- ➔ **930 Million events/sec on CloudLab cluster**

Seastar: State-of-the-art Cloud Key-Value Store



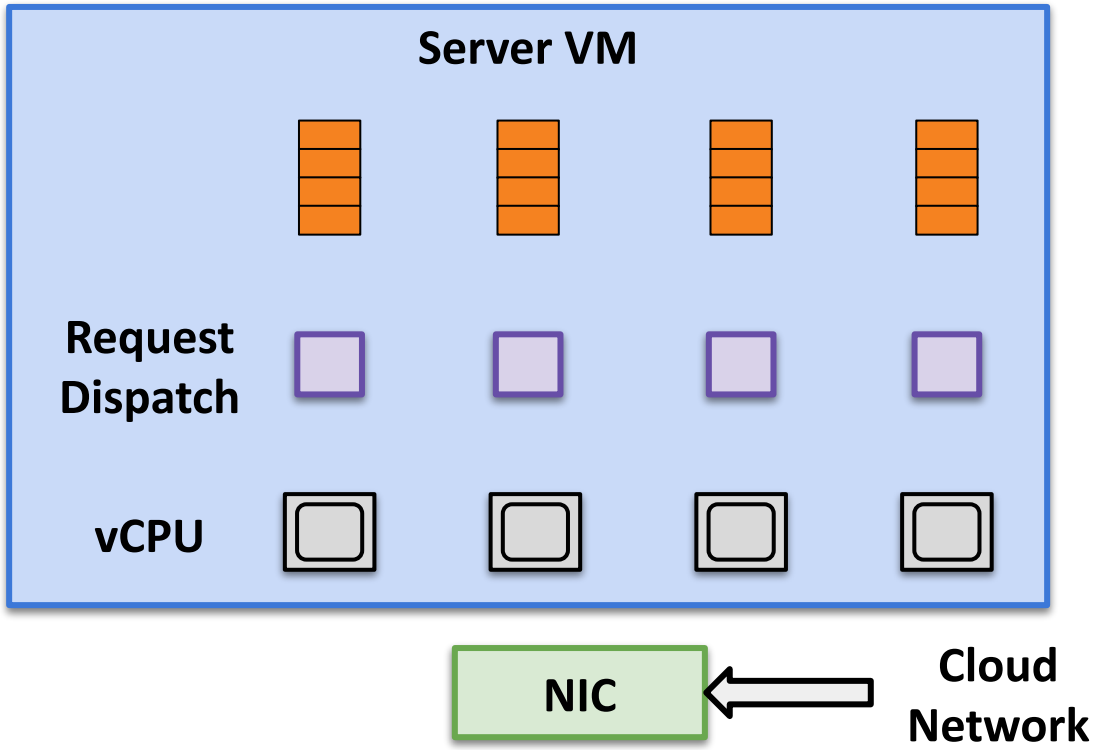
Seastar: Records Partitioned Across Cores



Hash partition records across vCPUs

Avoids synchronization on records

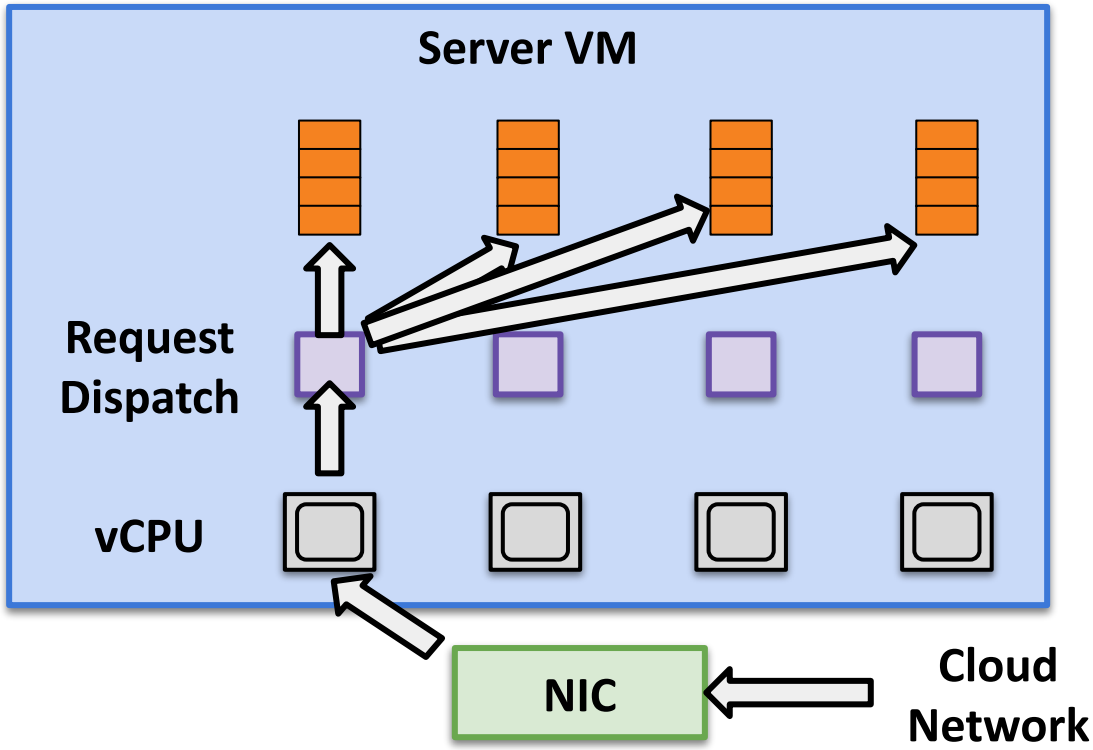
Seastar: Records Partitioned Across Cores



Dispatcher on every vCPU

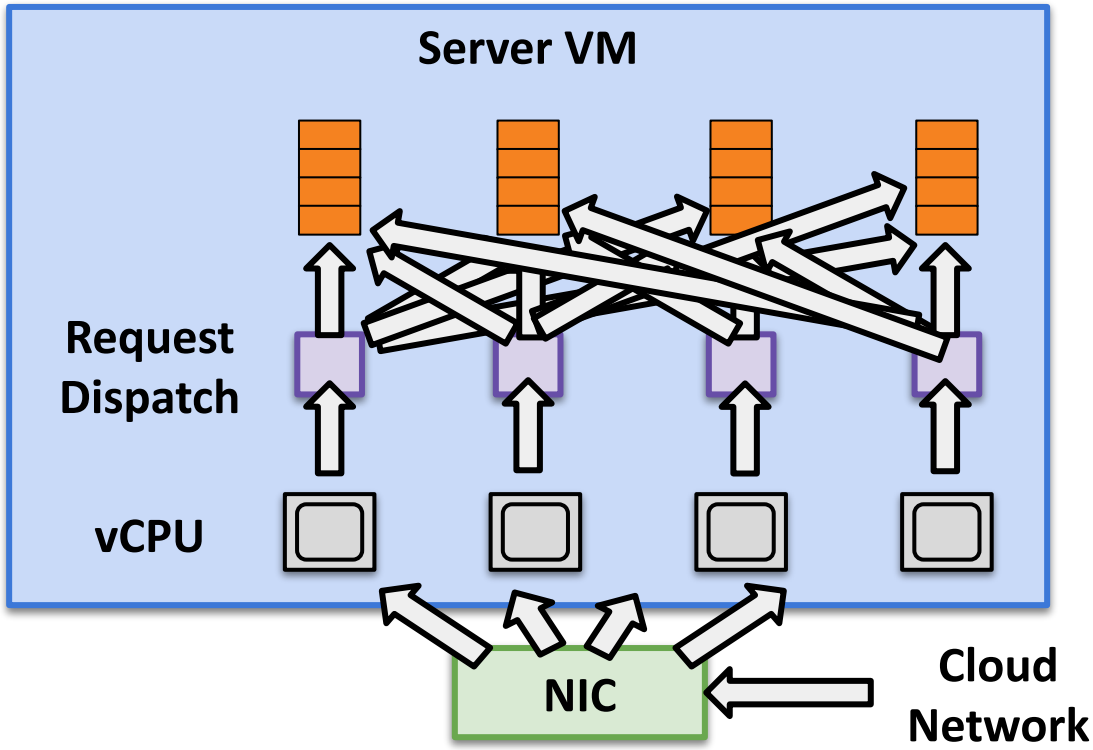
Each listens on different TCP port

Seastar: Requests Routed Within Server



Each dispatcher routes requests to correct partition

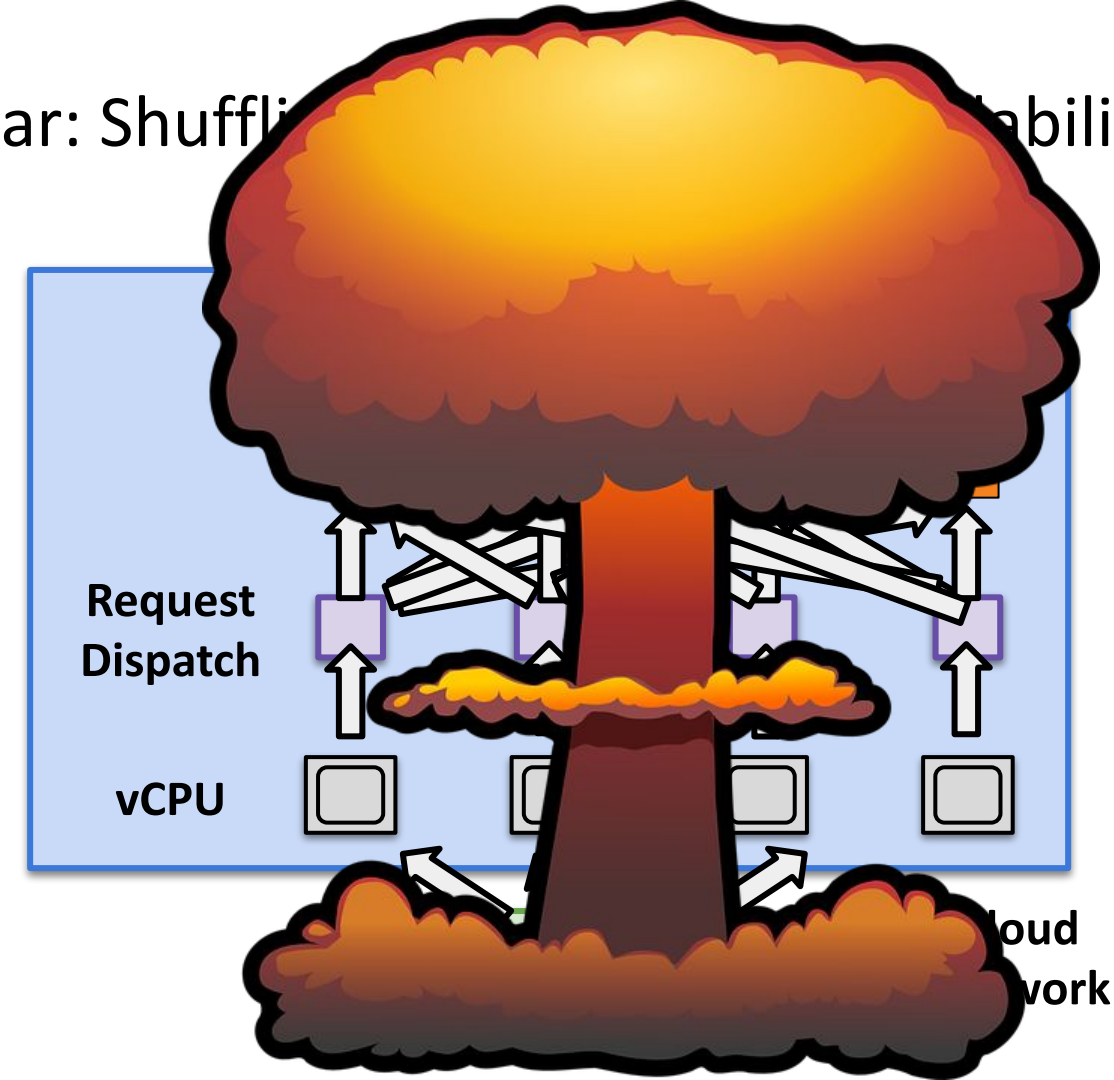
Seastar: Shuffling Requests Limits Scalability



Each dispatcher routes requests to correct partition

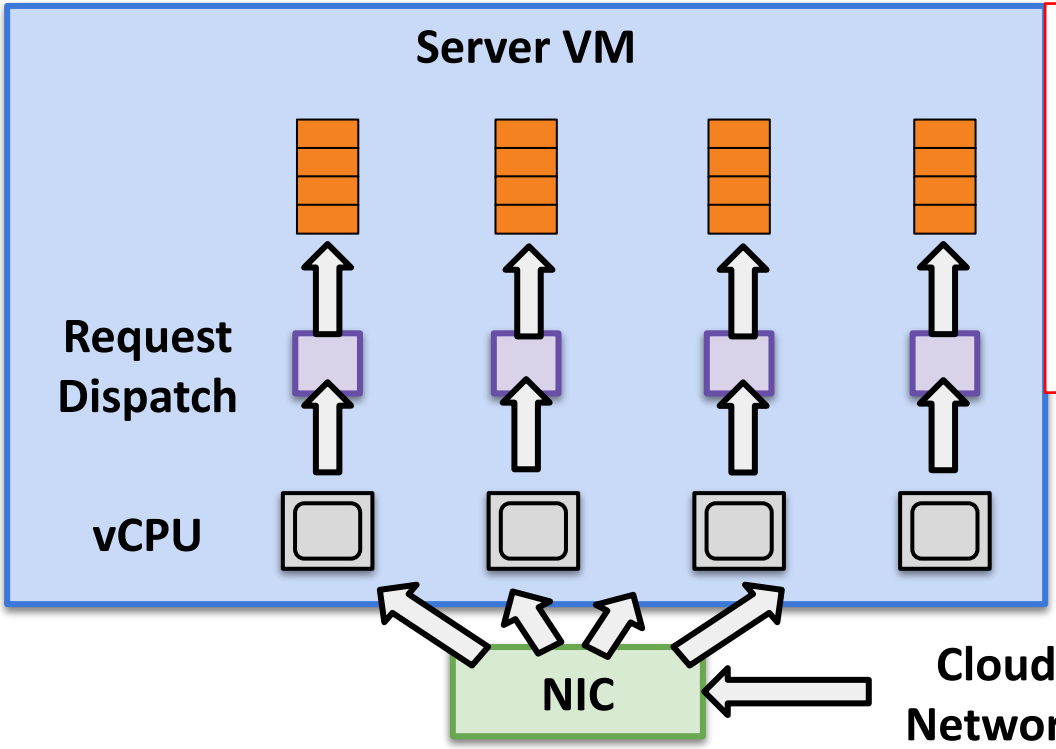
Expensive shuffle!

Seastar: Shuffli…ability



Doesn't scale across vCPUs

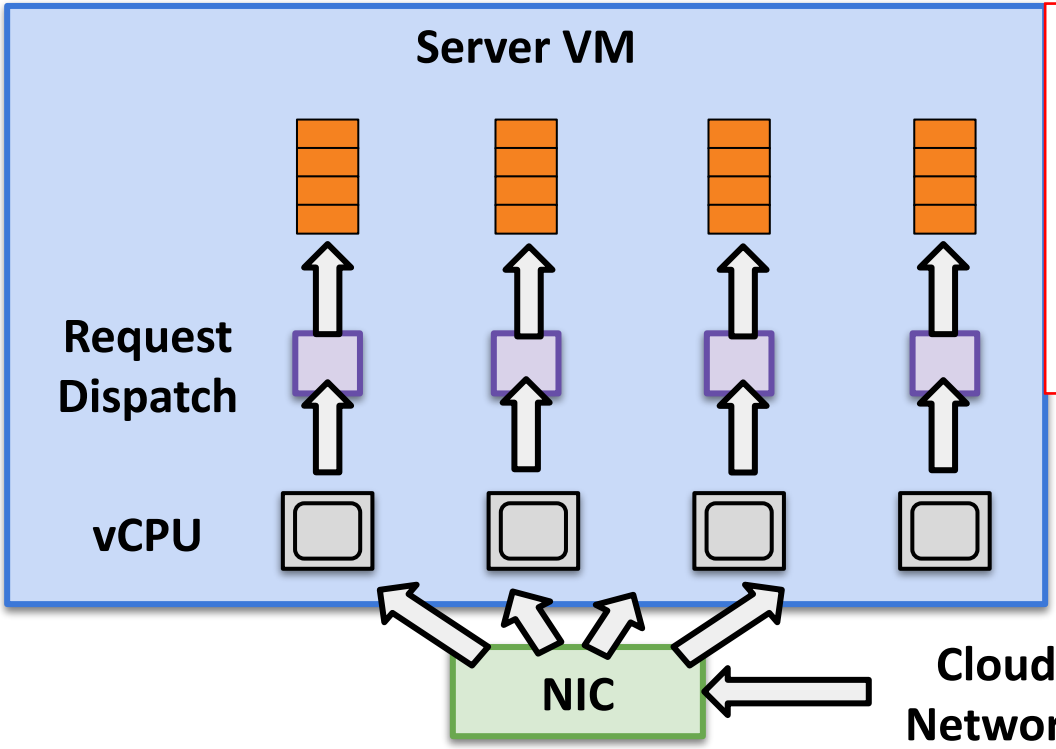
Seastar: Clients Could Route Requests Correctly



Clients could route to correct core/owner

Quadratic connection state!

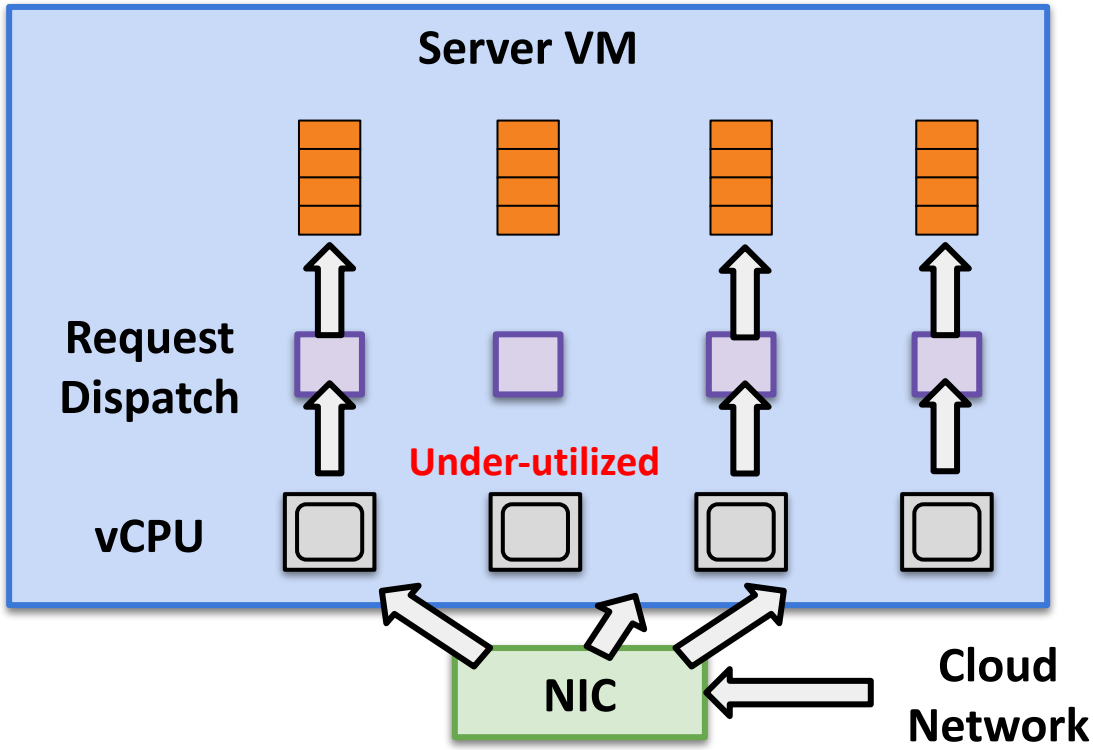
Seastar: Clients Could Route Requests Correctly



Clients could route to correct core/owner

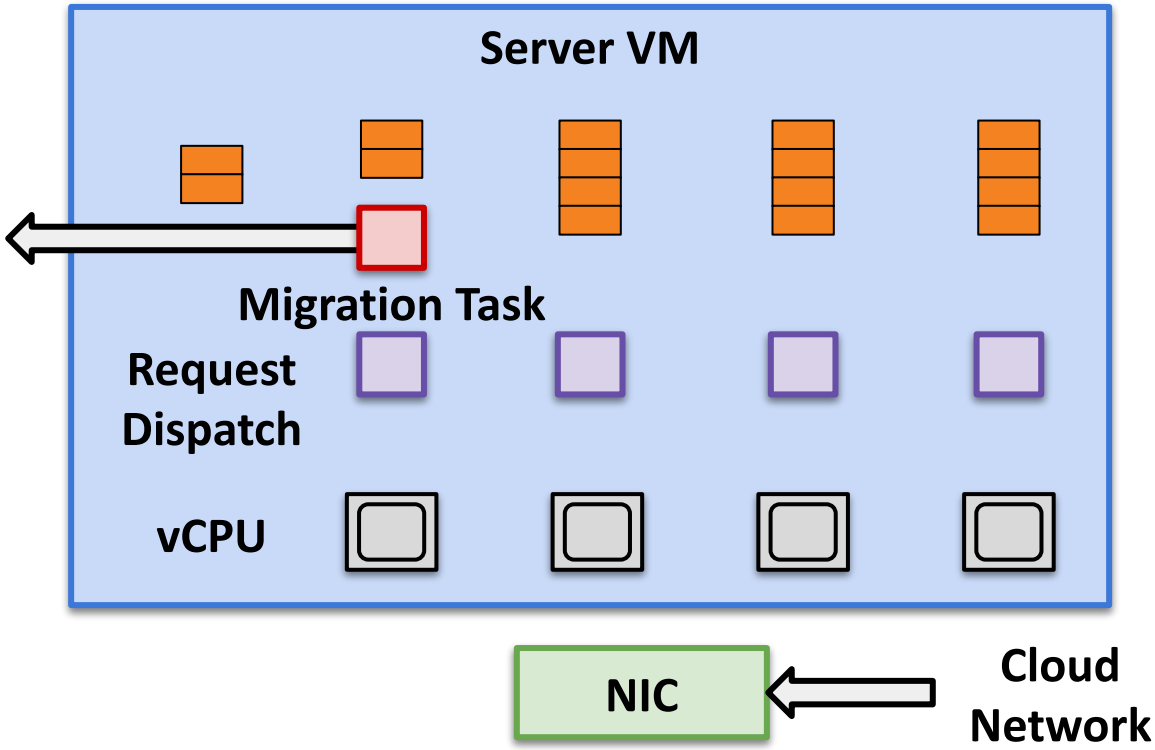
Shuffle requests at client!

Seastar: Bad For Skewed Workloads



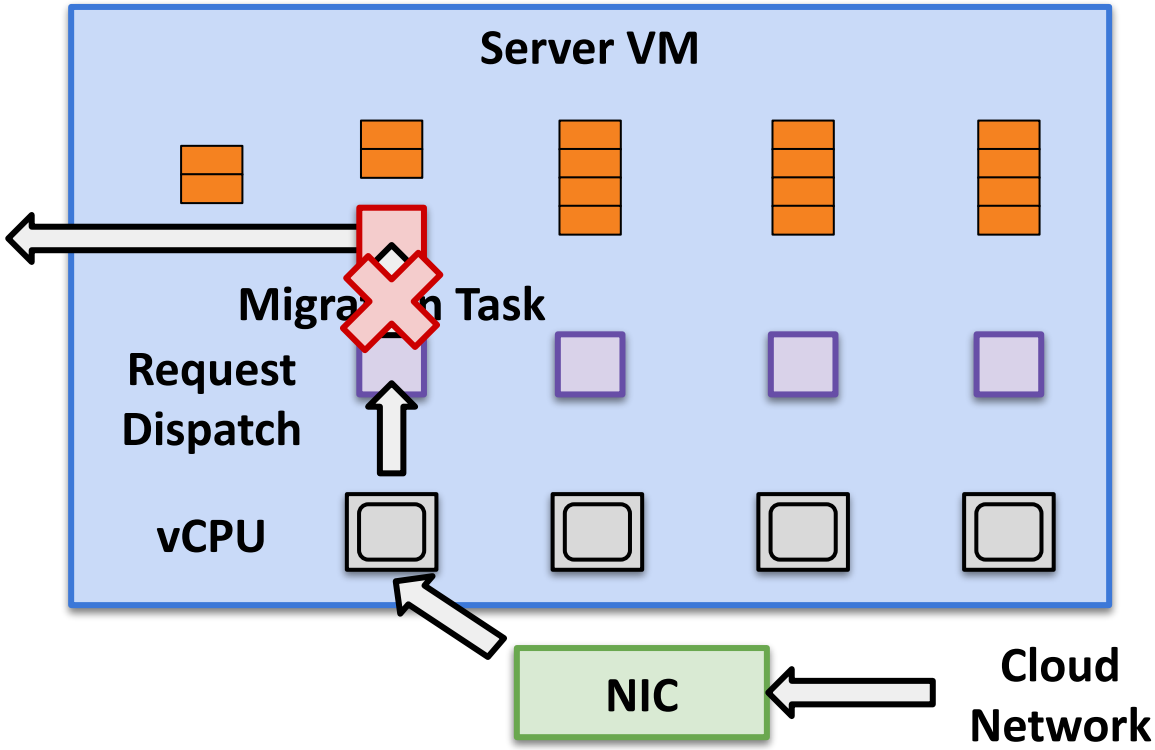
Bad for skewed workloads

Seastar: Migration Has To Be Single Threaded



Migration cannot be parallelized

Seastar: Head-of-line Blocking During Migration



Migration blocks requests on partition

Shadowfax: Design

Partitioned Request Dispatch, Shared Data

- ➔ vCPUs share lock-free index ⟶ record synchronization handled by cache coherence
- ➔ Migration can be parallelized, does not block requests on hash ranges

End-to-End Asynchronous Clients

- ➔ Issue pipelined asynchronous batches of requests to amortize network overhead
- ➔ Good for workloads with inter-request independence. Ex: click counts, heartbeats etc.

Shadowfax: Refer To Paper

Asynchronous Global Cuts

- ➔ Per server view numbers represent hash ranges owned by server
- ➔ Cores avoid coordination during migration ownership changes

Scale-out and Hash Migration Protocol

- ➔ Leverages asynchronous global cuts; is parallel and deadlock-free
- ➔ Fault-tolerant, can be cancelled; can recover if servers crash

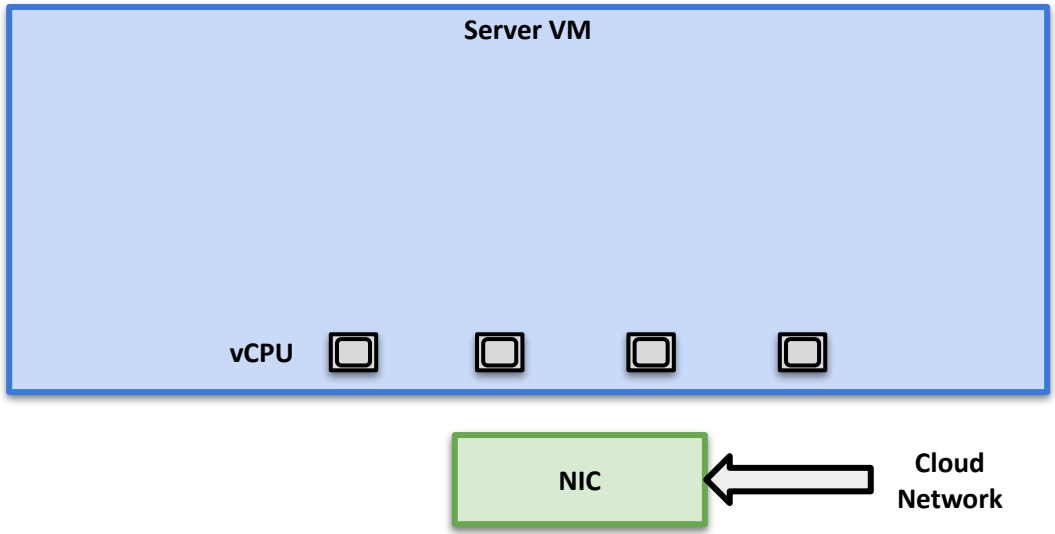
Spans Main-memory, Local SSD, and Shared Remote Storage

- ➔ During migration, only transfer (hot) data in main-memory
- ➔ Transfer pointers to rest (cold), lazily clean these up during compaction

Shadowfax: Partitioned Request Dispatch Shared Data

Problem: **Avoid cross-core coordination at server**

Solution: **Offload all coordination to hardware cache-coherence protocol**



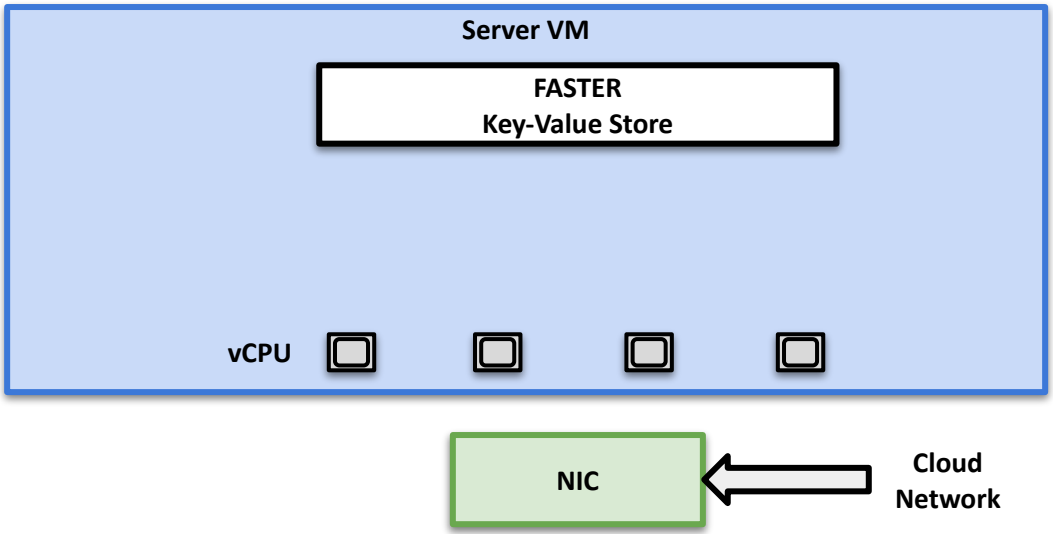
Shared memory is good for skew

Can serve hot records in parallel

Shadowfax: Partitioned Request Dispatch Shared Data

Problem: **Avoid cross-core coordination at server**

Solution: **Offload all coordination to hardware cache-coherence protocol**



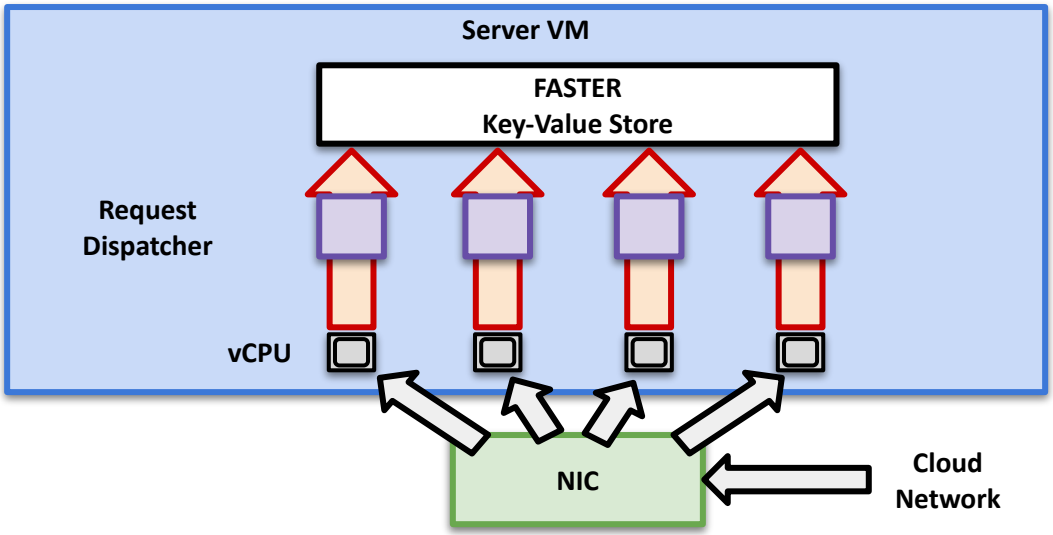
Lock-free index

Record synch happens at cache-coherence protocol (hw)

Shadowfax: Partitioned Request Dispatch Shared Data

Problem: **Multi-core request processing requires cross-core coordination**

Solution: **Offload all coordination to hardware cache-coherence protocol**



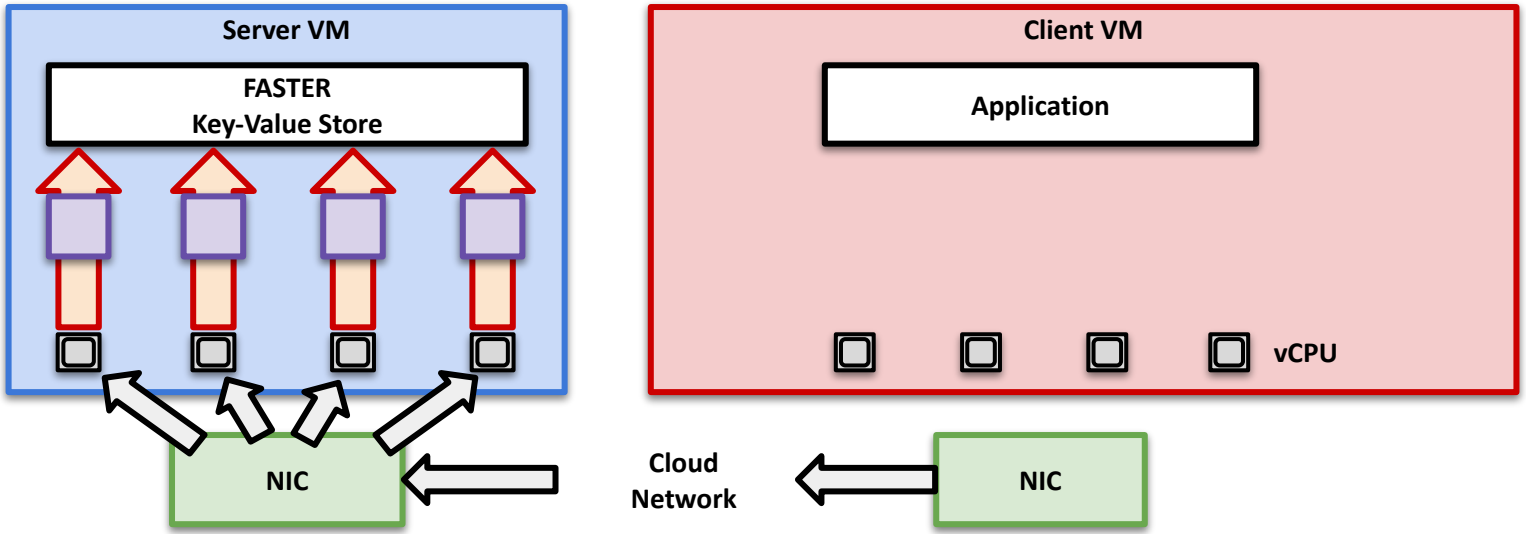
Partition request dispatch

Each dispatcher listens on separate TCP port

Shadowfax: Partitioned Request Dispatch Shared Data

Problem: **Keep servers saturated at index, not network or dispatch**

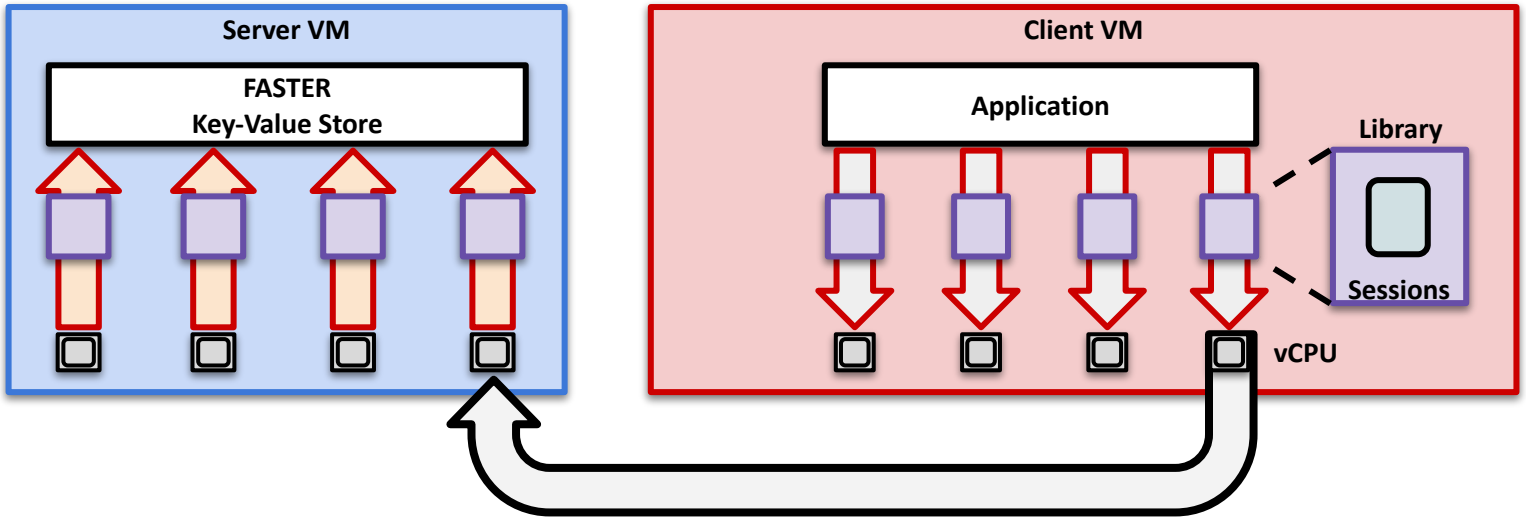
Solution: **Asynchronous client library, transparent network acceleration**



Shadowfax: Partitioned Request Dispatch Shared Data

Problem: Keep servers saturated at index, not network or dispatch

Solution: Asynchronous client library, transparent network acceleration

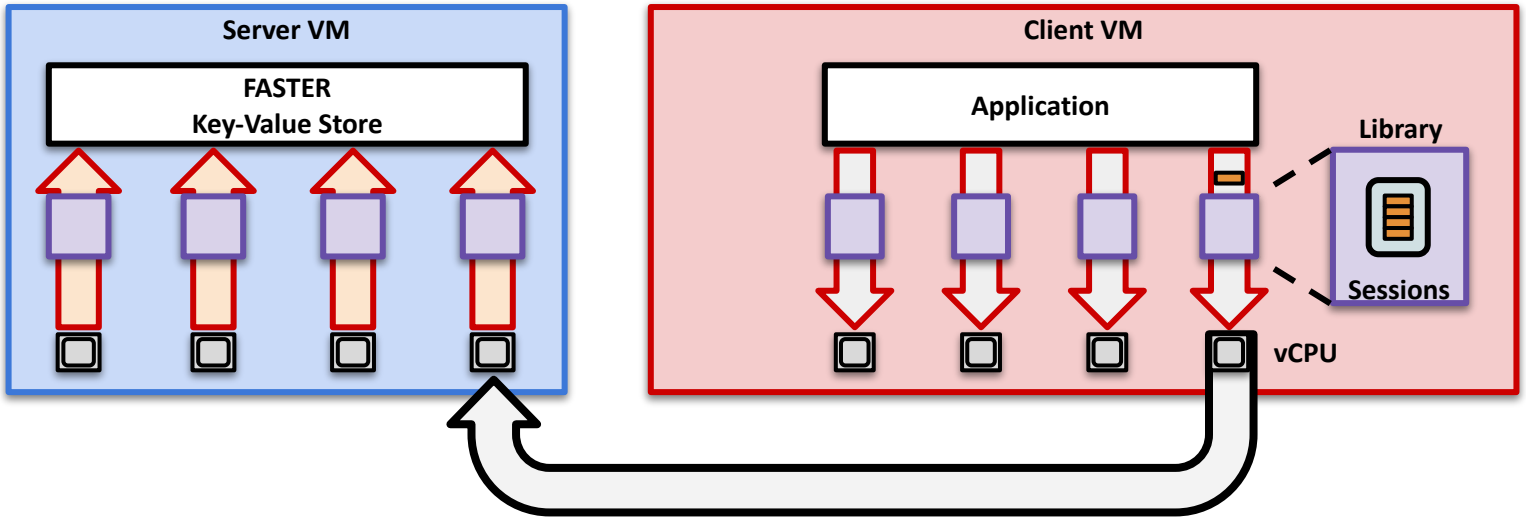


Client cores open *"sessions"* to server cores

Shadowfax: Partitioned Request Dispatch Shared Data

Problem: Keep servers saturated at index, not network or dispatch

Solution: Asynchronous client library, transparent network acceleration



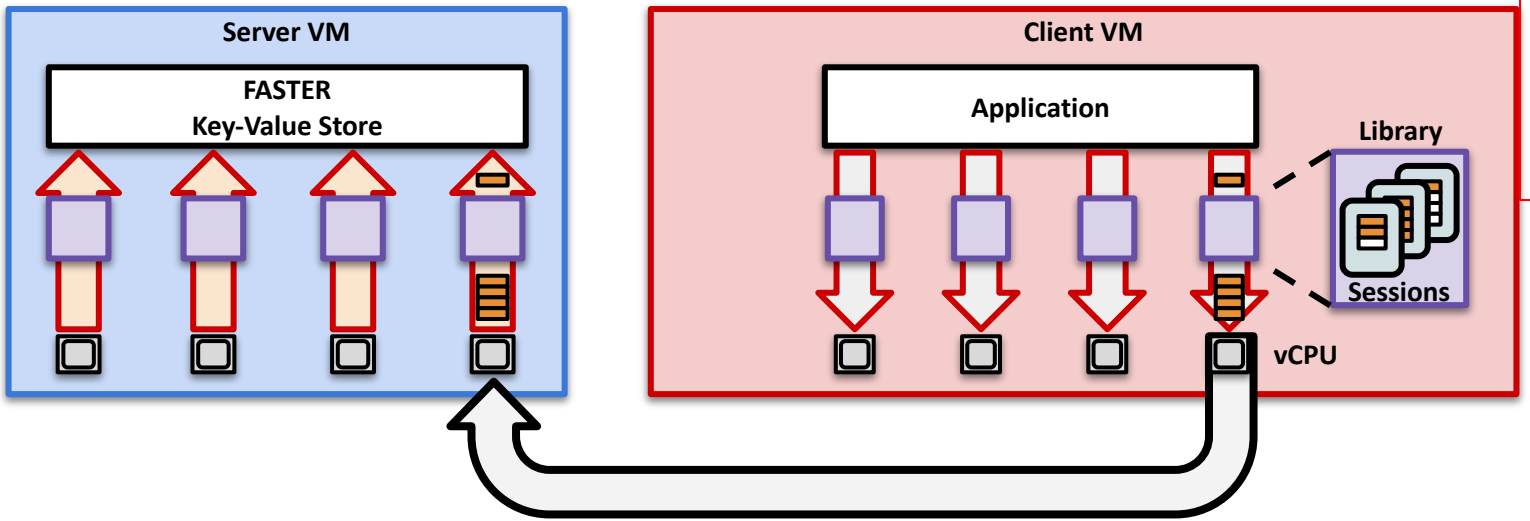
Requests are async

Enqueued in session buffers

Shadowfax: Partitioned Request Dispatch Shared Data

Problem: Keep servers saturated at index, not network or dispatch

Solution: Asynchronous client library, transparent network acceleration

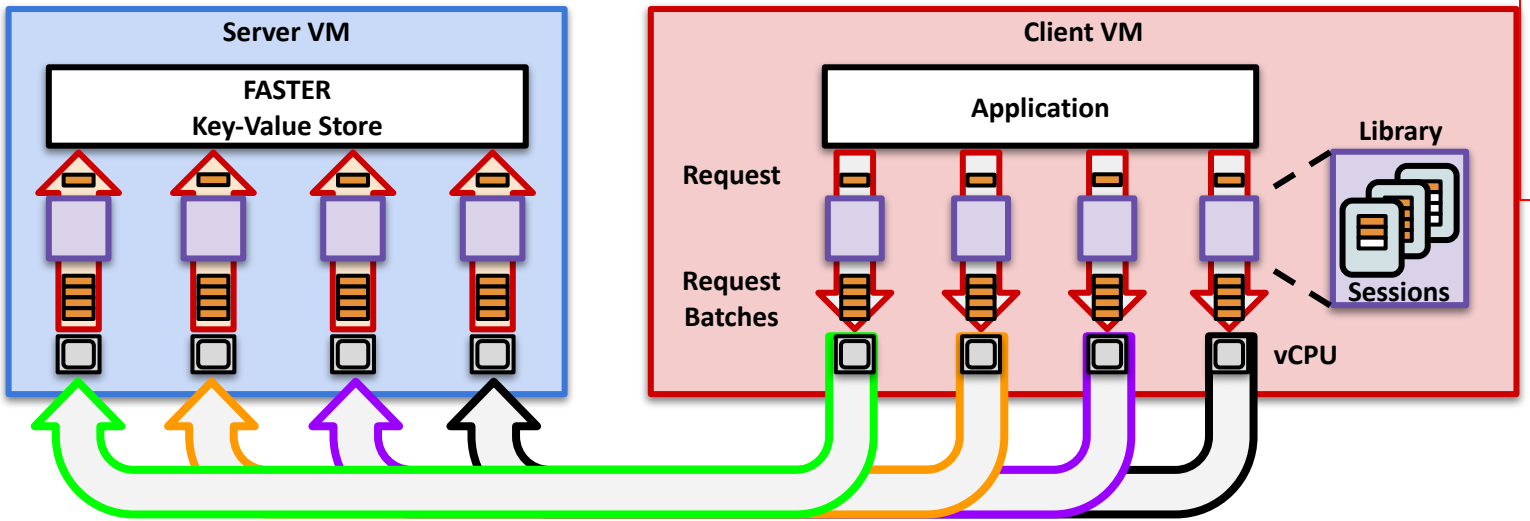


Pipelined batches are transmitted asynchronously

Shadowfax: Partitioned Request Dispatch Shared Data

Problem: Keep servers saturated at index, not network or dispatch

Solution: Asynchronous client library, transparent network acceleration

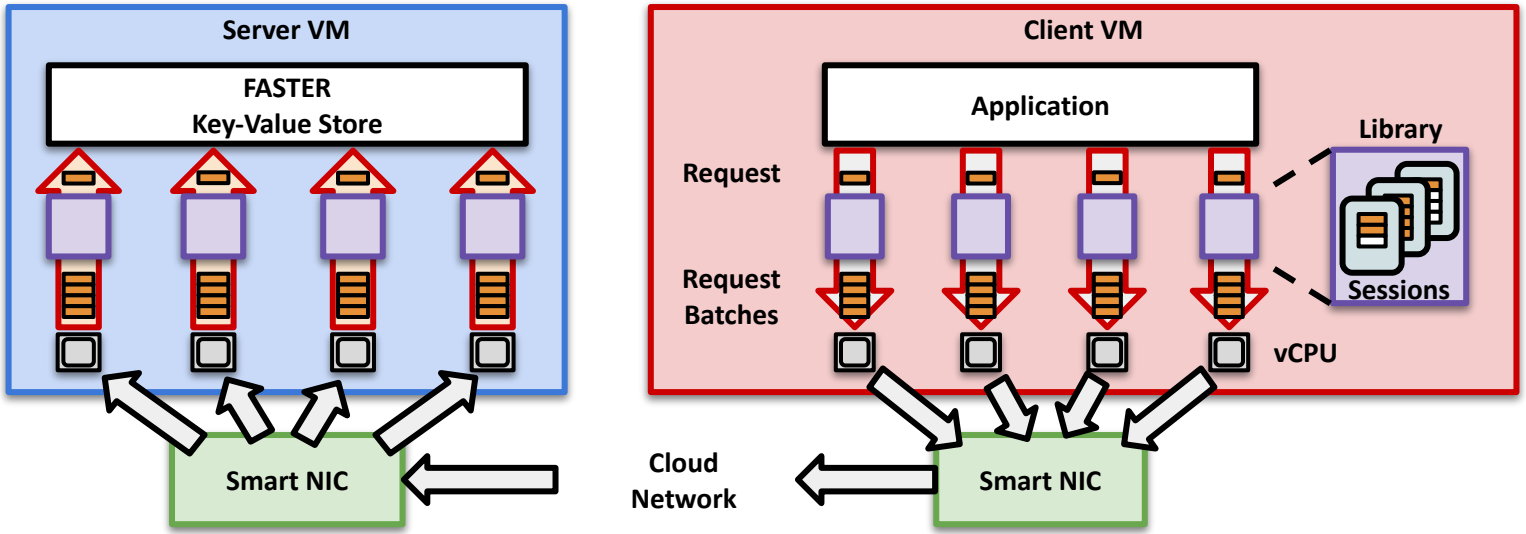


Pipelined batches are transmitted asynchronously

Shadowfax: Transparent Cloud Acceleration

Problem: Keep servers saturated at index, not network or dispatch

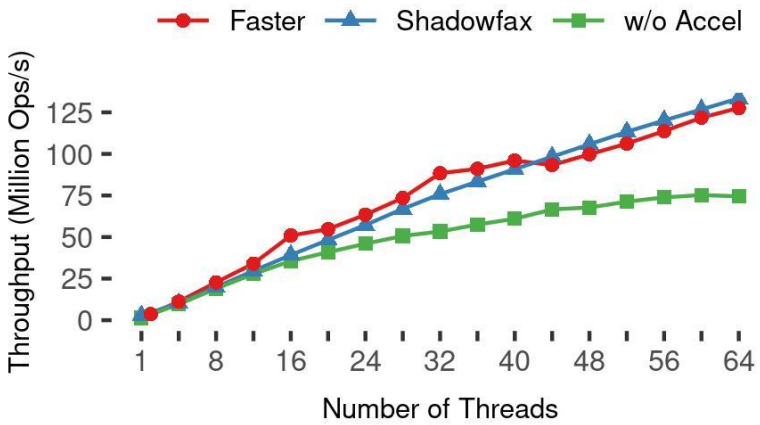
Solution: Asynchronous client library, transparent network acceleration



Hardware accelerated transport (TCP & Infrc)

Performance of Shadowfax

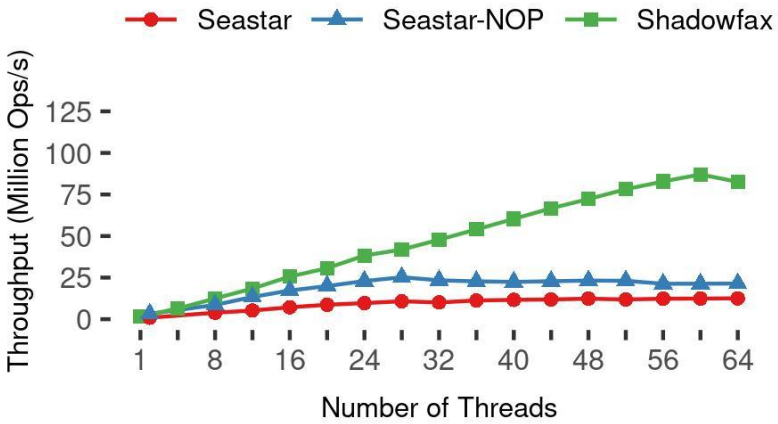
YCSB-F (Read-Modify-Writes, Benchmark Ingest of Events), 250 Million objects



Saturates server at Index

Performance of Shadowfax

YCSB-F (Read-Modify-Writes, Benchmark Ingest of Events), 250 Million objects



Saturates server at Index, 8.5x state-of-the-art

One Slide Summary

Multi-core optimized single-node key-value stores are emerging

- ➔ **Very** high throughput ~**100 Million events/sec/server.** Ex: FASTER (SIGMOD'18)

Problem: Retain high throughput in the public cloud

- ➔ Avoid request dispatch and network bottlenecks
- ➔ Support reconfiguration/migration while preserving throughput

Shadowfax: Distributed key-value store built on FASTER

- ➔ 100 Million events/sec/VM **on Azure**
- ➔ **930 Million events/sec on CloudLab cluster**