Carbink: Fault-Tolerant Far Memory

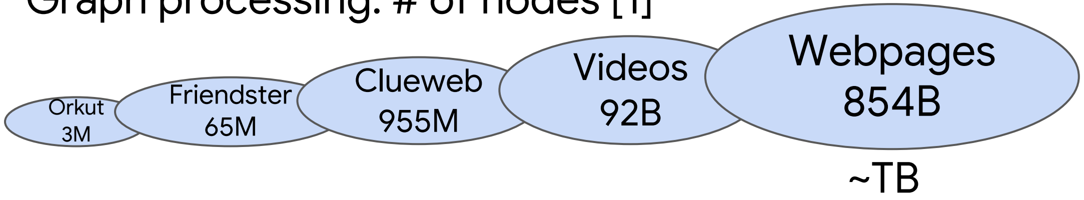
Yang Zhou[1*] Hassan M. G. Wassel[2] Sihang Liu[3*] Jiaqi Gao[1] James Mickens[1] Minlan Yu[1,2]
Chris Kennelly[2] Paul Turner[2] David E. Culler[2] Henry M. Levy[2,4] Amin Vahdat[2]

[1]Harvard University [2]Google
[3]University of Virginia [4]University of Washington

* Contributed to this work during internships at Google.

Memory-Intensive Applications in Data Centers

Graph processing: # of nodes [1]



VOLTDB [2]

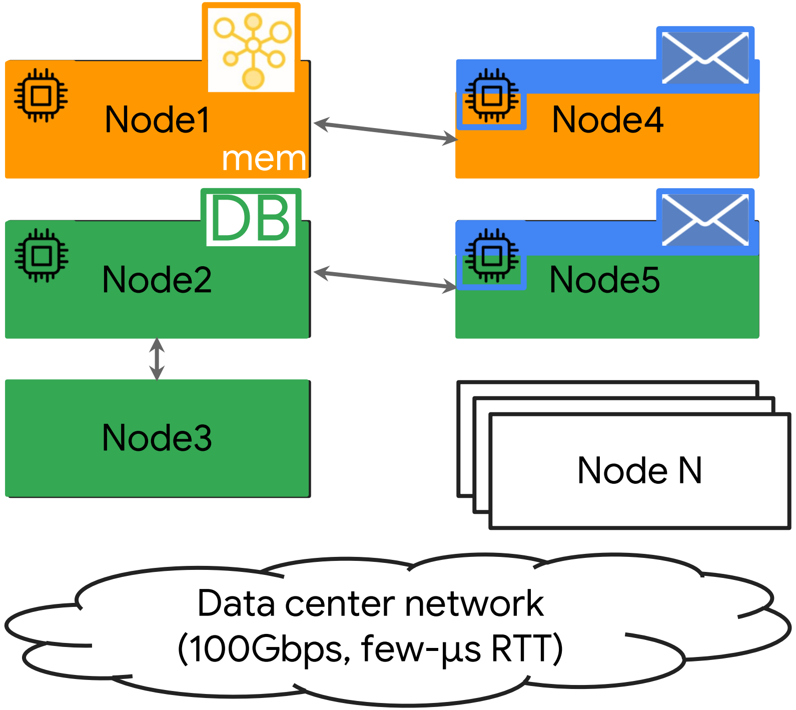
Memory provisioning is hard, as memory is limited by server physical boundary

- Over-provisioning memory for peak usage → 40%-60% memory utilization [3]
- Growing data in one process even exceeds single-server memory limit

Can applications dynamically utilize the unused memory on other servers?

[1] Łącki, Jakub, et al. "Connected components at scale via local contractions." arXiv preprint 2018
[2] Stonebraker, Michael, et al. "The VoltDB Main Memory DBMS." IEEE Data Eng. Bull 2013
[3] Tirmazi, Muhammad, et al. "Borg: the next generation." EuroSys 2020

Background: Far Memory on Commodity Servers [1,2,3,...]

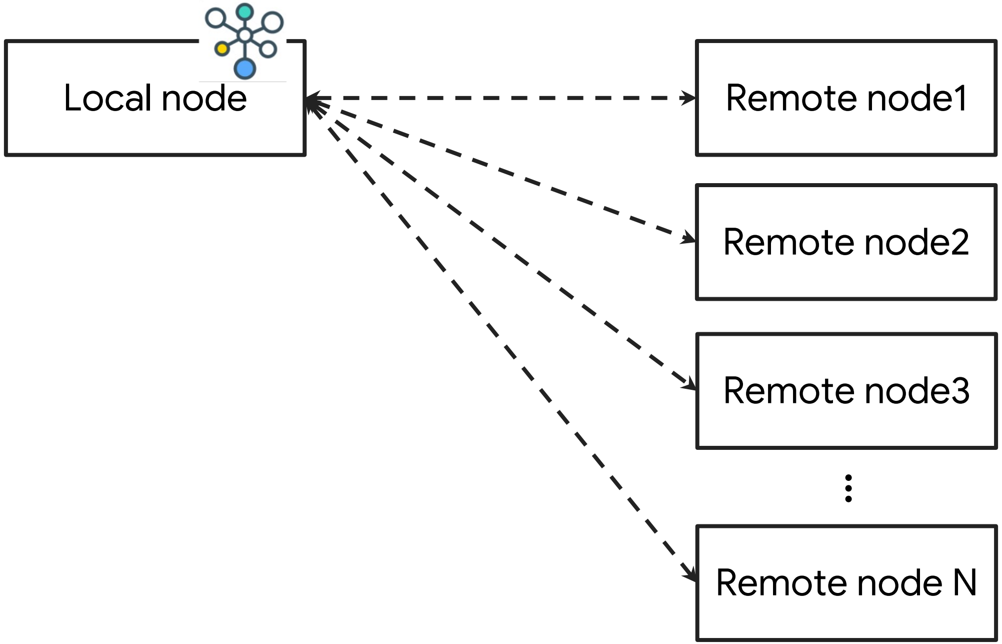


Benefits of far memory:

- Dynamically provisioning unused memory to memory-intensive apps
- Apps can use much more memory than single-machine limit

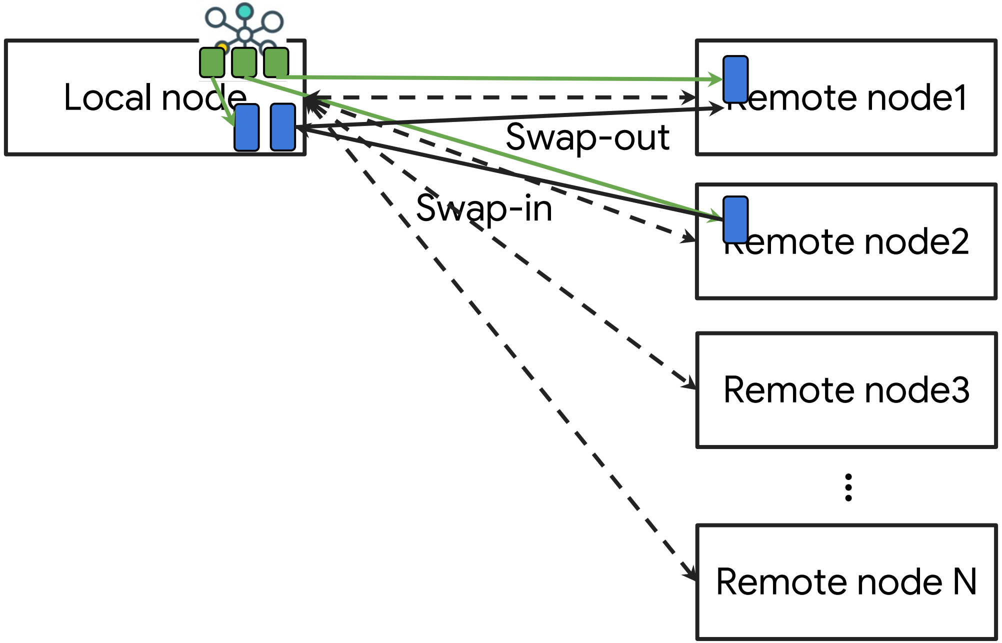
[1] Gu, Juncheng, et al. "Efficient memory disaggregation with infiniswap." NSDI 2017
[2] Aguilera, Marcos K., et al. "Remote regions: a simple abstraction for remote memory." ATC 2018
[3] Amaro, Emmanuel, et al. "Can far memory improve job throughput?." EuroSys 2020
...

Background: Far Memory on Commodity Servers [1,2,3,...]



[1] Gu, Juncheng, et al. "Efficient memory disaggregation with infiniswap." NSDI 2017
[2] Aguilera, Marcos K., et al. "Remote regions: a simple abstraction for remote memory." ATC 2018
[3] Amaro, Emmanuel, et al. "Can far memory improve job throughput?." EuroSys 2020
...

Application Interface: Remotable Pointers



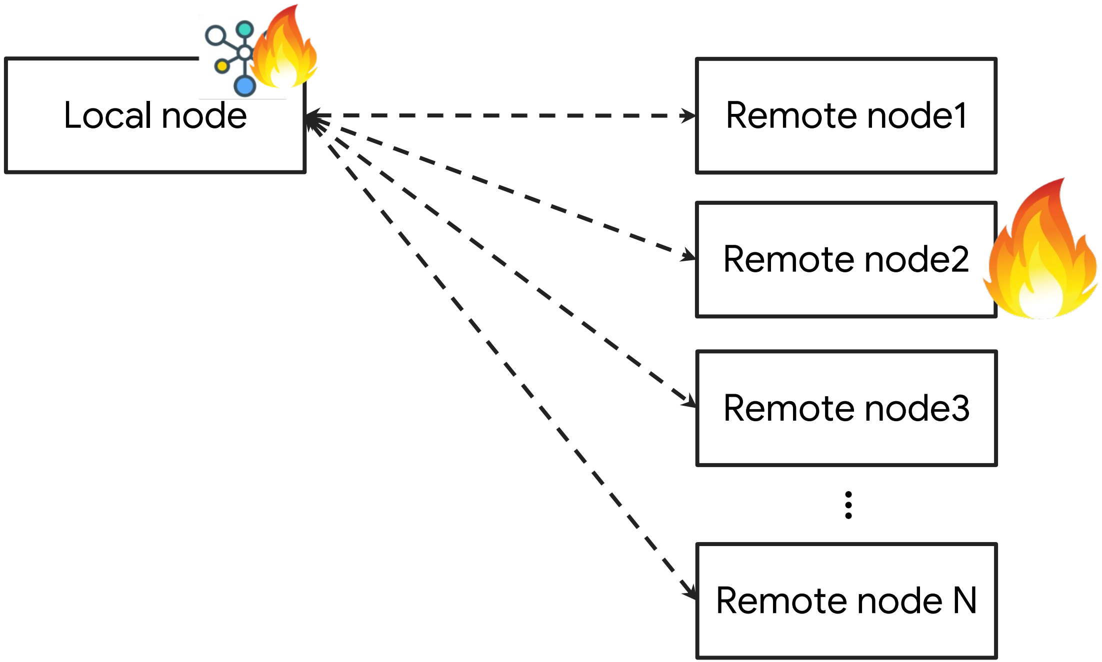
Application-Integrated Far Memory [1]

- Remotable pointer
- Remotable object

What if remote nodes fail?

[1] Ruan, Zhenyuan, et al. "AIFM: High-performance, application-integrated far memory." OSDI'20

The Must-Have Feature: Fault Tolerance



Probability of application crash grows almost linearly with N

How to build a fault-tolerant far memory system?

... assume fail-stop faults and no partial network failures

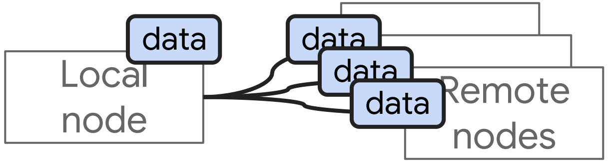
Talk Outline

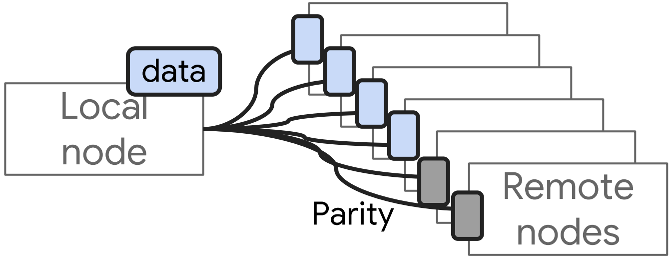
Direction: in-memory erasure coding for fault tolerance

Carbink: making erasure coding work in practice

Evaluation: performance and cost of Carbink

Replication vs. Erasure Coding





Replication

- High memory overheads (3x)

Erasure coding (EC)

- Much smaller memory usage (1.5x)
- Single core achieves 4GB/s encoding tput [1]

SSD vs. Memory

SSD would become bottleneck during bursty workloads or failure recovery [1]

In-memory erasure coding

[1] Lee, Youngmoon, et al. "Hydra: Resilient and Highly Available Remote Memory." FAST'22

Talk Outline

Direction: in-memory erasure coding for fault tolerance

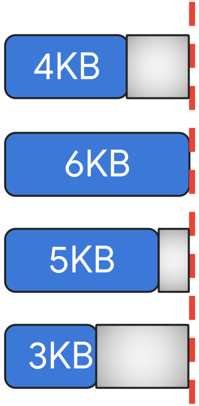
- High performance & low memory usage

Carbink: making erasure coding work in practice

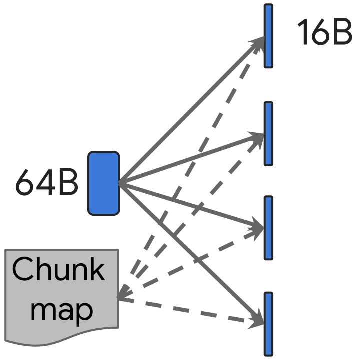
Evaluation: performance and cost of Carbink

Challenge 1: Remotable Objects Have Different Sizes

Erasure coding irregular-sized objects is hard



Padding: objects aligned but wasting memory



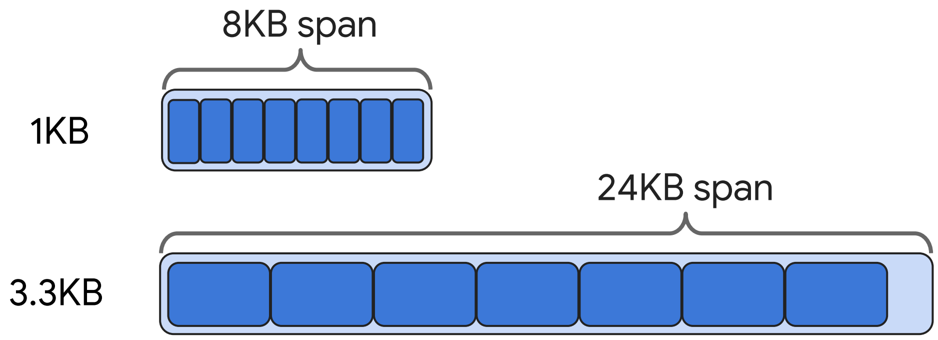
Splitting: small objects incurs large metadata

Carbink approach: grouping similar-sized objects into spans (like TCMalloc [1])

- Spans are page-aligned and regular-sized

[1] Hunter, Andrew Hamilton, et al. "Beyond malloc efficiency to fleet efficiency: a hugepage-aware memory allocator." OSDI'21

Grouping Similar-Sized Objects into Spans

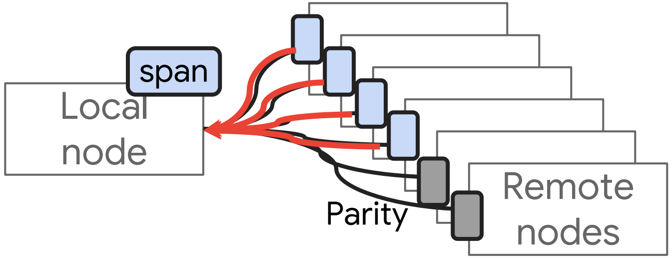


Span-centric memory pooling

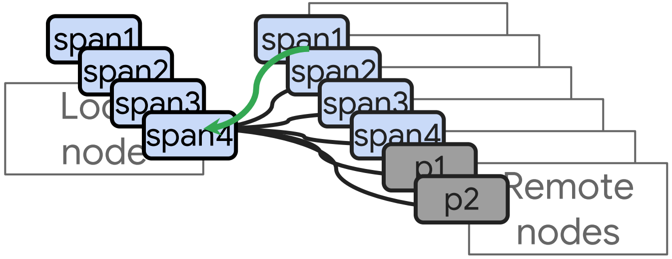
- Applying spans to object management and data swapping
- Spans are page-aligned, and never end with a partial object

Challenge 2: Efficient Swapping under Erasure Coding

EC-Split (Hydra [1]):
erasure codes individual spans



EC-Batch (Carbink):
erasure codes **spansets**



Multiple network IOs to swap-in/out a span

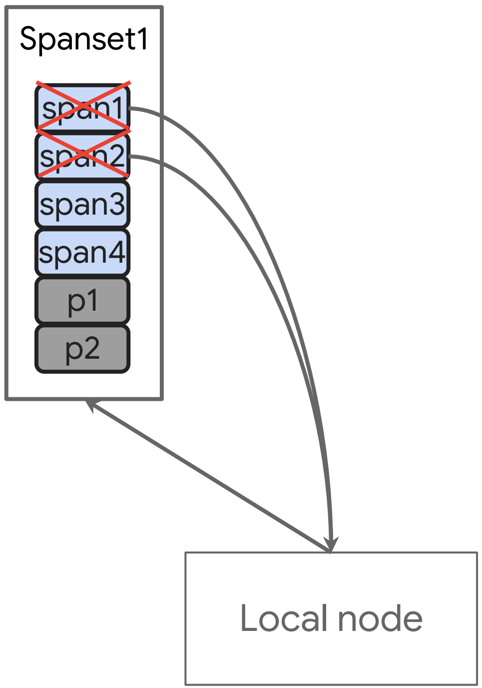
- Stressing network stack ⟶ slow swapping
- Stragglers ⟶ high tail latency

Single network IO to swap-in a span

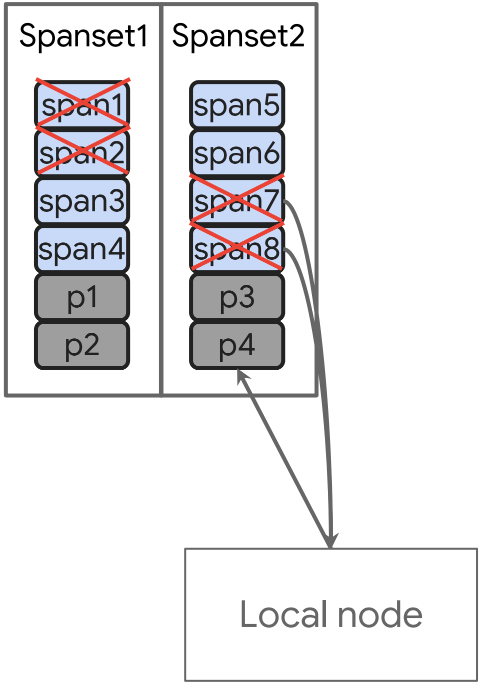
- Fast swapping and low tail latency

[1] Lee, Youngmoon, et al. "Hydra: Resilient and Highly Available Remote Memory." FAST'22

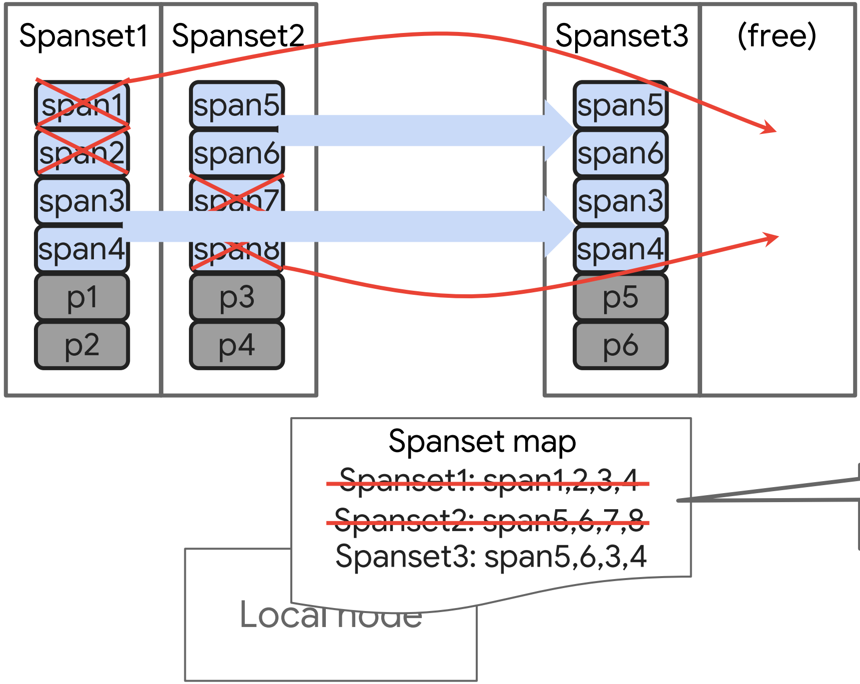
Swap-In&Out Granularity Mismatch → Remote Fragmentation



Swap-In&Out Granularity Mismatch ⟶ Remote Fragmentation



Remote Compaction for Defragmentation



No impacts on span swapping perf: off the critical path of swap-ins/outs

Penalty: may consume more memory; dead spans not compacted immediately

Zero-copy span merging

Talk Outline

Direction: in-memory erasure coding for fault tolerance

- High performance & low memory usage

Carbink: making erasure coding work in practice

- Span-centric memory pooling → managing arbitrary-sized objects
- Erasure coding spansets → achieving swapping efficiency

<u>Evaluation: performance and cost of Carbink</u>

Evaluation Overview

Workloads:

- An internal transactional KV-store doing TPC-A transactions
- Graph connected components (skipped here due to time limit)
- A microbenchmark dereferencing remotable objects

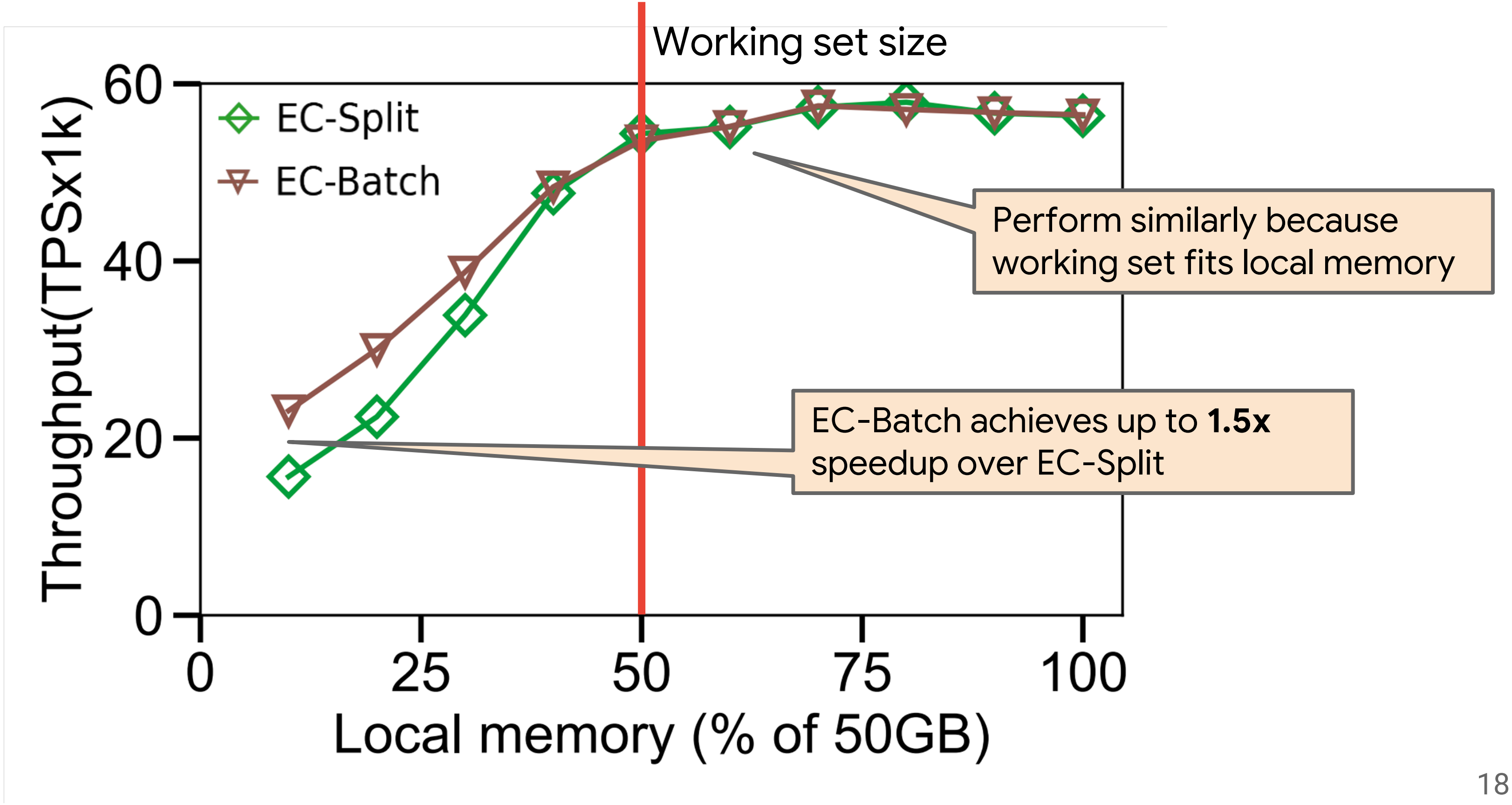
Metrics: throughput, tail latency, memory usage

Testbed:

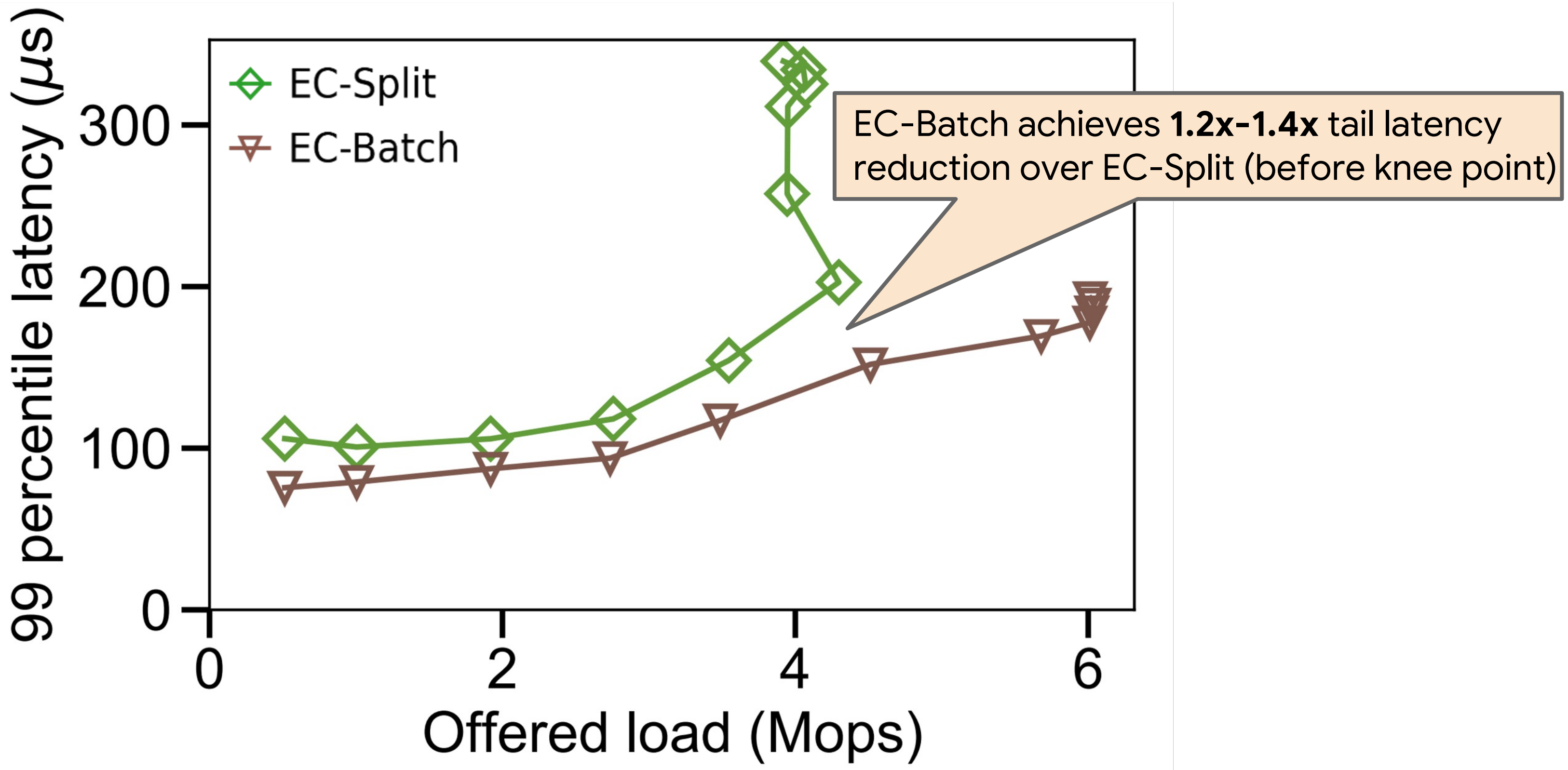
- Servers with 50 Gbps NIC and PonyExpress [1] user-space network stacks
- One-sided RMAs for span swapping; RPCs for remote compaction

[1] Marty, Michael, et al. "Snap: A microkernel approach to host networking." SOSP'19

Throughput (KV-store)



Tail Latency (Microbenchmark)



Other Results

Remote memory usage:

- EC-Batch consumes at most 35% more memory than EC-Split
- … but still only ⅔ of replication memory usage

More in the paper!

- Remote compaction resource usage
- Failure recovery times
- AIFM (swapping individual objects) vs. Carbink

Carbink Summary

Fault tolerance is a must-have feature for applications to use far memory

Carbink: making erasure coding FT work in practice for far memory system

- Grouping objects into spans → handle arbitrary-sized objects
- Erasure coding spansets → single network IO data-fetch

Up to 1.5x application speedup and 1.4x tail latency reduction with up to 35% more memory usage (compared to state-of-the-art EC-Split)

Thank You!

Carbink: making erasure coding FT work in practice for far memory system

Up to 1.5x application speedup and 1.4x tail latency reduction with up to 35% more memory usage (compared to state-of-the-art EC-Split)

Icons from Flaticon.com, vecteezy.com, icon-library.com, nicepng.com

Backup Slides

Carbink Design:

- AIFM Programming Interface
- Thread Synchronization
- Mitigating Swap-in Amplification

Carbink Evaluation:

- AIFM vs. Carbink Performance
- Remote Memory Usage (KV-store)
- Failure Recovery (KV-store)

Back

Application-Level Remoteable Pointers (like AIFM[1])

```
RemUniquePtr<Node> rem_ptr = AIFM::MakeUnique<Node>();
{
  DerefScope scope;
  Node* normal_ptr = rem_ptr.Deref(scope);
  //Compute over the Node object.
} //Scope is destroyed; Node object can be evicted.
```

- DerefScope constructor acquires RCU lock; deconstructor releases
- Deref() checks pointer status bits

"Reverse pointer": embedded in each object, pointing to the RemUniquePtr

- Enables object moving and evicting

[1] Ruan, Zhenyuan, et al. "AIFM: High-performance, application-integrated far memory." OSDI'20.

Back

Thread Synchronization

Application threads

1. Grabs an RCU read lock (ie, DerefScope)
2. If the **P**resent bit is not set, swap in
3. If **P** is set:
    a. But the **M** and **E** bits are unset, return object (local) address
    b. If **M** is set (by filtering threads), race to acquire the pointer's spinlock
        i. If winning, makes a copy of the object, and returns its address.
        ii. Otherwise, go to step 1.b
    c. If **E** is set (by eviction threads), ...

Filtering threads

1. Set **M** bit
2. Call `SyncRCU()` (ie, the RCU write waiting lock)
3. Race to acquire the pointer's spinlock
    a. If winning, move the object
    b. Otherwise, ignore the object

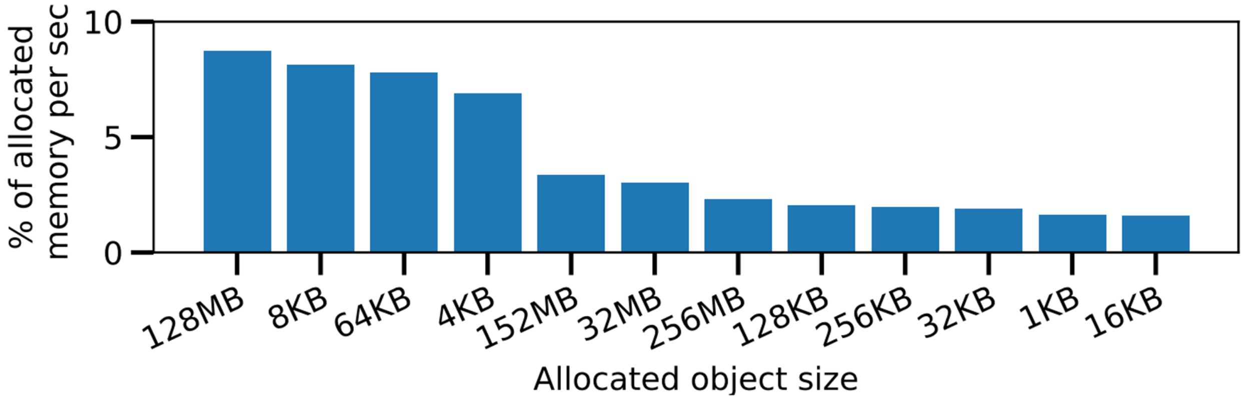
Eviction threads

1. Set **E** bit
2. ...

Back

Mitigating Swap-in Amplification

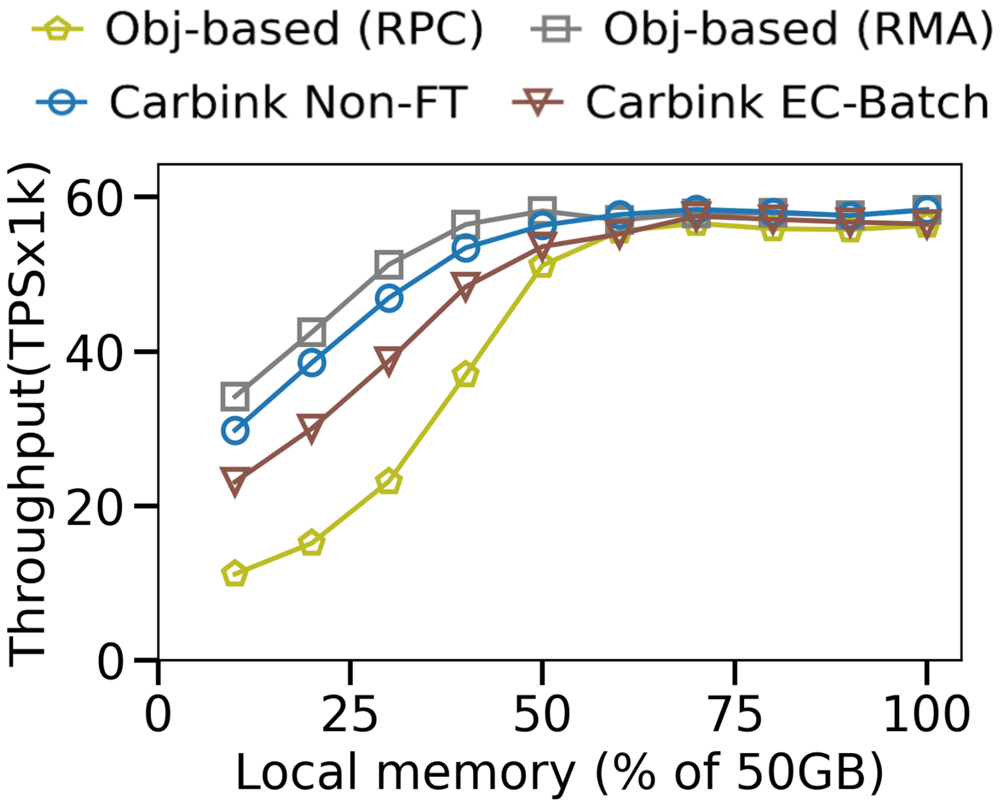
Prioritizing evicting spans containing large objects



- GWP: large objects occupy the majority of memory.
- Moreover, hot objects tend to be small: Spanner reports roughly 95% of accesses involve objects smaller than 1.8KB.

Back

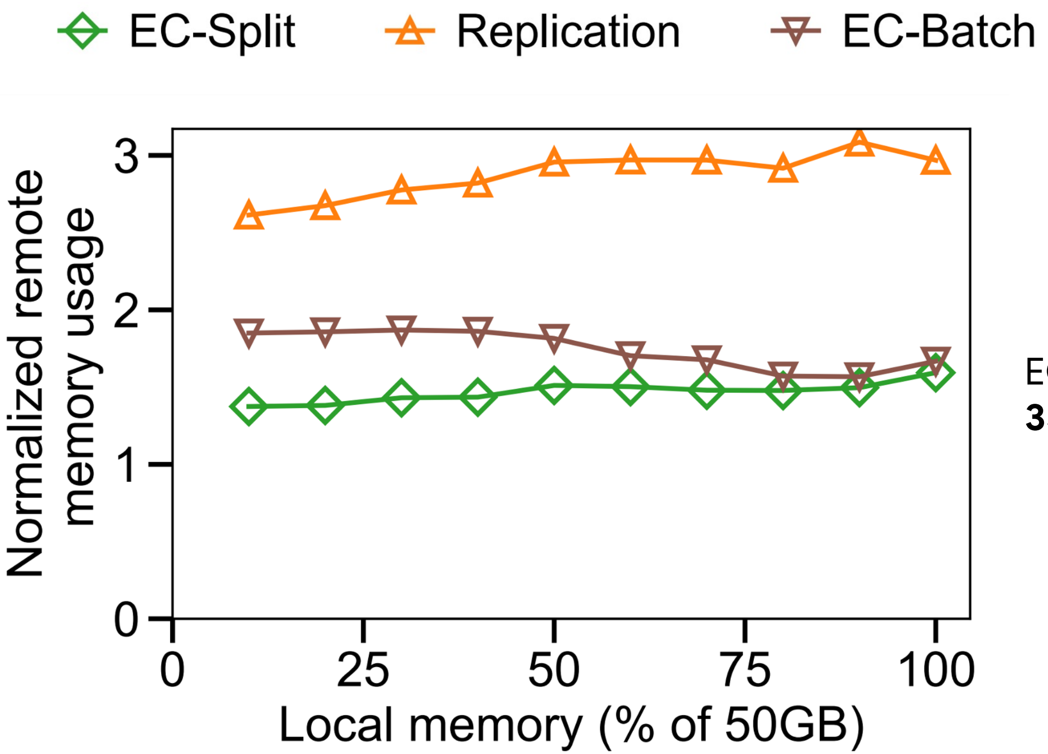
AIFM (Swapping Individual Objects) vs. Carbink



Carbink Non-FT: similar performance as AIFM.

[Back](#)

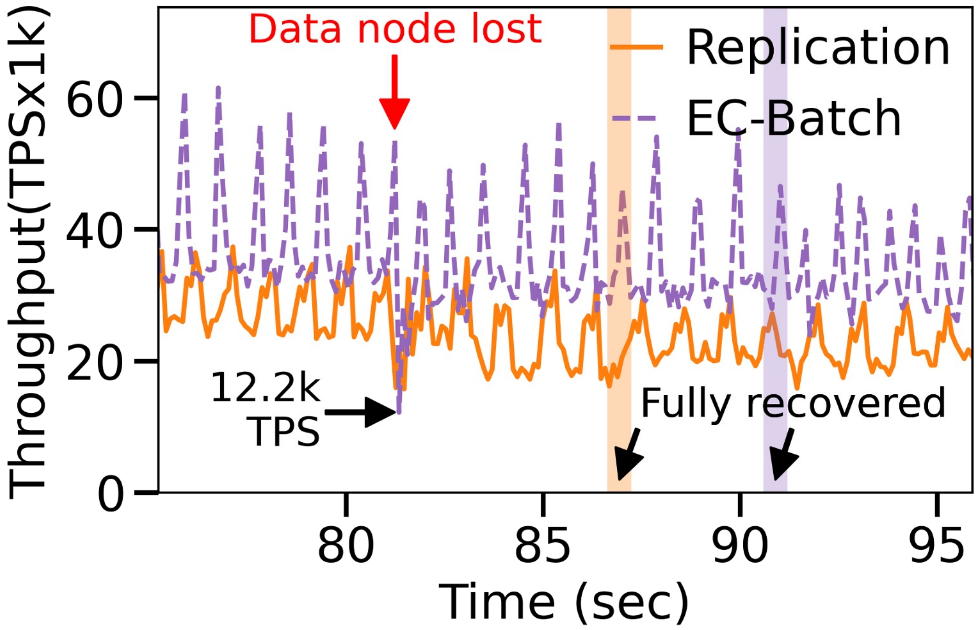
Remote Memory Usage (KV-store)



EC-Batch remote: at most **35%** more memory usage.

Back

Failure Recovery (KV-store)



EC-Batch: 0.6s to restore to normal vs. Replication 0.3s.

EC-Batch: 1.7x longer time for fully recovery.