Cornflakes: Zero-Copy Serialization for Datacenter Networks


* Stanford University ◊ Microsoft Research ◇ UC Berkeley

Platform Lab Retreat
Thursday, February 17, 2022
Microsecond applications can’t afford software data serialization

- Kernel bypass RTTs within 5 us
- Protobuf serialization + deserialization of a simple data structure, 1024-byte sized, takes 1.1 us

message Object {
    optional string msg = 1;
}
Serialization involves data movement

- Serialization steps:
  1. Add header
  2. Copy two fields together
  3. Copy into pinned memory accessible to the NIC, with packet header

- Key issue: Data movement is required to coalesce non-contiguous I/O regions together
Software serialization limits achievable throughput

- Data structure: 4096 byte-message divided into 2 fields
- Serialization libraries achieve 14% of single core throughput (zero-copy echo server)

```plaintext
message Object {
  optional bytes m1 = 1
  optional bytes m2 = 2
}
```

Is it possible to write a serialization library that removes some of these costs?

Single-core serialization echo server running over UDP DPDK networking stack
Research has proposed custom accelerators for serialization.

Is it possible to improve serialization with commodity hardware?
Scatter-gather offload enables zero-copy serialization with commodity NICs

- Scatter-gather allows networking stacks to construct and send packets from non-contiguous I/O regions
- This talk: *how do we build a serialization library using scatter-gather?*
Talk agenda

• Introduction

• Key challenge around using scatter-gather

• Performance tradeoffs behind scatter-gather

• Cornflakes library design

• Evaluation
How does scatter-gather work

- Scatter-gather avoids CPU copies but uses extra PCIe requests to fetch non-contiguous memory regions.
- **When are these extra PCIe requests worth it?**
  - We ran a series of experiments to understand these tradeoffs on Mellanox CX-5 NICs and Intel Xeon Silver CPUs.
Scatter-gather microbenchmark

- Server initializes large region of memory divided into segments
- Clients request non-contiguous segments
- Server coalesces segments using scatter-gather or copying
- Goal: see how performance metrics change across parameters
Microbenchmark metrics

• Metrics calculated
  • Varied offered load per configuration
  • Calculated highest achieved load
  • This ignores latency, but attempts to capture “knee” of the throughput latency curve
Fetching more segments limits scatter-gather performance

- Both approaches suffer from extra per-segment metadata
- Scatter-gather additionally suffers from:
  - CPU assembles larger work requests (16 additional bytes per segment)
  - Extra PCIe transfers when segments are < 256 bytes large
Increasing working set size doesn’t slow down scatter-gather transmission

- In the copying approach, program metadata and data segments share the cache as copying segments brings the data into the L1 cache.
- For scatter-gather, execution state (metadata) can remain in the L1 cache as the NIC performs copies.

4096 byte packet, 2 segments

4096 byte packet, 8 segments
Talk Agenda

• Introduction

• Background on scatter-gather

• Scatter-gather vs. memory copying measurement study

• Cornflakes

• Evaluation
Application memory used for zero-copy I/O must be accessible to the NIC

• Problem:
  • For zero-copy I/O, application memory must live in pinned pages accessible to the NIC (and registered in the case of Mellanox NICs)

• Solution:
  • Cornflakes provides block based memory allocators for applications to use
Zero-copy I/O introduces NIC/CPU data races

• Problem:
  • Application could free or write to a buffer while the NIC is still sending it before transmission completion notification

• Solution:
  • Reference counting
  • Disable writing to cornflakes-allocated buffers via the Rust API
Ordering requirements on serialization wire format

• Problem: The ordering of scatter-gather arrays given by the application could induce extra PCIe round trips

• Solution:
  • Cornflakes orders NIC accessible segments to go at the end
Cornflakes API: Abstraction

// Ref-counted buffer for scatter-gather element
pub struct RcBuf {
    data_pointer: *mut mbuf; // pointer to raw packet
    offset: usize; // offset into packet
}

// Methods on RcBuf
impl RcBuf {
    // allocate buffer
    pub fn alloc(conn: &mut Network) -> RcBuf;
    // increase ref count on clone
    pub fn clone(&self) -> RcBuf;
}

// Scatter-gather array element pointer
pub enum CfPtr<'a> {
    RawRef(&'a [u8]), // raw pointer if not zero copy
    RcRef(RcBuf) // ref-count if zero copy
}

**Listing 1:** Scatter-gather array interface used by Cornflakes.

For zero-copy serialization, applications use reference counted buffers.

These buffers are allocated by Cornflakes and increase the ref count on clone.

Applications pass a list of references (either raw or ref counted) to the networking stack.
message GetMsg
{
  optional uint32 id = 1;
  optional string key = 2;
  optional ref.counted.bytes val = 3;
}

impl GetMsg { // some functions omitted
  fn get_val(&self) -> RcBuf;
  fn set_val_rc(&self, buf: &RcBuf);
  fn alloc_hdr_buf(&self) -> Vec<u8>;
  fn serialize(&self, buf: &mut[u8]) -> Vec<CfPtr>;
  fn deserialize(&mut self, pkt: &Vec<CfPtr>);
}

fn handle_get(&self, pkt: &Vec<CfPtr>, conn: &Network) {
  let mut get_msg = GetMsg::deserialize(pkt);
  let value = self.map.get(get_msg.get_key());
  get_msg.set_val(value);
  let mut header = get_msg.alloc_hdr_buf;
  conn.send(get_msg.serialize(&mut header));
}

**[Written by app developer]**: Apps can use the protobuf syntax to define message schemas; mark values used for zero-copy

**[Generated by Cornflakes]**: Applications set and get reference counted buffers; serialize produces a scatter-gather array where the first entry is the object header.

**[Written by app developer]**: Applications pass scatter-gather arrays to the networking stack for transmission
Cornflakes API: Wire format

message GetMessage {
  optional int32 id = 1;
  optional ref_counted_string key = 2;
  optional ref_counted_bytes value = 3;
}

Fields ordered to put NIC accessible values at the end to minimize PCIe round trips

Fields ordered by size for potential networking stack optimizations
Zero-copy enables throughput gains

- Hardware scatter-gather enables 3x throughput gains over serialization baselines for 1 get
- Hardware scatter-gather throughput degrades with more entries
  - Increased cache misses on keys themselves
  - Cornflakes inefficiency during deserialization
  - More metadata processing per segment
Conclusion: scatter-gather enables zero-copy serialization without new hardware

- Cornflakes is the first system to enable **zero-copy serialization** without new hardware
- Cornflakes shows **1.8-3x throughput gains** over prior systems for a key value store

deeptir@cs.stanford.edu
https://deeptir.me
github.com/deeptir18