To Copy or Not to Copy: Making In-Memory Databases Fast on Modern NICs

Aniraj Kesavan, Robert Ricci, and Ryan Stutsman

University of Utah School of Computing
{aniraj, ricci, stutsman}@cs.utah.edu

Abstract. When databases resided primarily on disks, the problem of data layout was focused on structures that enabled efficient reads and writes from that medium, as well the as effective use of main memory as a cache. With in-memory databases, this part of the I/O problem is largely gone, but that does not mean that I/O considerations can be completely ignored. We argue for the importance of considering the "other" side of the I/O equation—network communication with clients—when designing in-memory databases.
Modern NICs include a number of optimizations intended to improve I/O performance, including kernel bypass, zero-copy and scatter-gather DMA. Applied carefully, these features can reduce the involvement of the CPU in network transfers and can save memory bandwidth. However, our experiments show that some optimizations do not always provide a benefit in a database context, and using them can be tricky, as they affect strategies for processing updates and managing data lifetimes. In this paper, we explore the application of zero-copy NIC DMA to in-memory databases, and we explore how the NIC can influence and explicitly leverage data layout and concurrency control. We apply these results to the Bw-Tree structure by proposing a client-assisted design for transmitting large range scan results. Overall, the approach boosts throughput by 1.7× and reduces CPU overhead by 75% compared to simple zero-copy DMA.

1 Introduction

For decades, the performance characteristics of storage devices have dominated thinking in database design and implementation. From data layout to concurrency control, to recovery, to threading model, disks touch every aspect of database design. In-memory databases effectively eliminate disk I/O as a concern, and these systems now execute millions of operations per second, sometimes with near-microsecond latencies. It is tempting to believe that database I/O is solved; however, another device now dictates performance: the network interface (NIC).

As the primary I/O device in modern databases, NICs' performance characteristics should now be taken into consideration from the start in database designs. However, modern network cards have grown incredibly complex, partly in response to demands for high throughput and low latency. Their complexity makes understanding them difficult, and it makes designing software to take advantage of them even harder. To design for NICs effectively, database engineers must understand their characteristics.

Database designers encounter a number of trade-offs when trying to use the NIC effectively. For example, when can the NIC efficiently transmit data directly from database records via direct memory access (DMA)? Should records be pre-materialized in a way that makes transmission efficient? Or can the NIC efficiently assemble scattered data on-the-fly? Does concurrent access to data records by the NIC complicate memory management? To answer these questions, and to understand how NICs can impact database design, we profiled a modern kernel-bypass capable NIC with remote direct memory access (RDMA) with specific attention to large transfers and query responses. This paper details our findings and makes several concrete suggestions on how NICs might influence data layout and concurrency control.

Specifically, we have found three factors that make it especially challenging to design NIC-friendly database engines:

- *Zero-copy* DMA engines reduce server load for transferring large data blocks, but large, static blocks are uncommon for in-memory databases, where records can be small and update rates can be high.
- The performance gains from zero-copy DMA do not generalize to *finer-grained objects* for two reasons: (1) transmit descriptors grow linearly in the number of records; (2) NIC hardware limits descriptor length, limiting speed for small records. Despite this, we find that zero-copy can make sense for small objects under certain conditions, such as small “add-ons” to larger transmissions.
- Zero-copy introduces complications for *locking and object lifetime*, as objects must remain in memory and must be unchanged for the life of the DMA operation (which includes transmission time).

These factors make advanced DMA difficult to use effectively. In fact, we find that conventional *copy-out* that assembles responses in transmit buffers has some advantages over zero-copy transmission. Most surprisingly, explicit copy-out can outperform zero-copy in transmit throughput even for “heavy” responses like large range query results from secondary indexes. Explicit copy-out does use more CPU and memory bandwidth: our experiments show it adds a modest 5% to CPU load, but more significantly, consumes up to one-third of the server’s total memory bandwidth. In-memory databases are often capacity bound [4], not CPU or memory bandwidth bound; this may make copy-out preferable for some workloads, especially since the relative cost is likely to decrease over time.

To provide a concrete illustration of these issues, we investigate in the context of the Bw-Tree, an index whose properties give it flexibility in how it interacts with the NIC. Ultimately, we find that zero-copy can be a benefit. However, for significant gains, data layout has to be adjusted to suit the NIC, and the costs of maintaining that layout must be considered against the NIC performance benefits. Our findings yield a preliminary design for efficient transmission of heavy range query results that improves throughput by 1.7× (from 3.4 GB/s to 5.7 GB/s) and reduces CPU overhead by 75% compared to fully scattered zero-copy by relaxing the structure of the results that clients receive. We start with an overview of related work in this space, then describe a modern kernel-bypass NIC and the Bw-Tree. We continue with a set of microbenchmarks and draw conclusions for data structure/networking layer co-design.

2 Motivation and Related Work

Our analysis is inspired by in-memory database work that focuses on lock-freedom and/or no-update-in-place [13, 5, 11, 12, 10]. The unique features of these structures make them well positioned to leverage the sophisticated userlevel DMA capabilities of modern networking hardware (we explore the reasons for this in Section 4). A major goal of our work is to help in tuning these structures for low-overhead, high-throughput network transmission.

Several efforts are now underway to use userlevel NIC access and/or RDMA as a fabric for high-performance distributed databases [17, 6, 7, 20, 19]. Our findings should also aid in their design, especially those that build on the abstraction of a network-attached atomic record store [15, 11].

Small, fixed-size request/response cycles have been optimized by existing research [6, 8, 14, 9, 17], but the efficient transmission of larger responses like range query results or data migrations has been less well studied. Studies focused on large transmissions have so far been limited to relatively static block-oriented data. Our work focuses on optimizing transmission of large and complex query results, which differs from these two categories. Database queries do access and transmit many fine-grained records, but the transmissions can be comprised of many records sent together. This makes the transmissions large like block data, but the contents are much more dynamic. This is because the set of records returned varies, and because the records themselves may be rapidly updated.

A complementary study by Kalia, Kaminsky, and Andersen [9] provides an analysis of host interaction with Mellanox Infiniband network adapters, and they extract rules of thumb to help developers get the best performance from the hardware. The low-level nature of their analysis is especially suited for optimizing dispatch-heavy workloads with a high-volume of small request-response operations. Our focus is on operations that require heavy responses where the precise content of the response cannot be anticipated or is under heavy mutation.

3 User-level NIC Access

NICs have been rapidly adding features in the past decade in response to the demands of data center, cloud, and HPC networking. Direct application access to the NIC, or *kernel bypass*, is now available from several commodity NIC vendors [2, 1], along with a myriad of offload and packet steering features to reduce CPU and message dispatch overheads. Kernel bypass allows an application to send and receive data without OS involvement. This results in several efficiency gains. Latency is improved; in many cases kernel and syscall time is the dominant factor in round-trip time [17]. It can also improve throughput while reducing

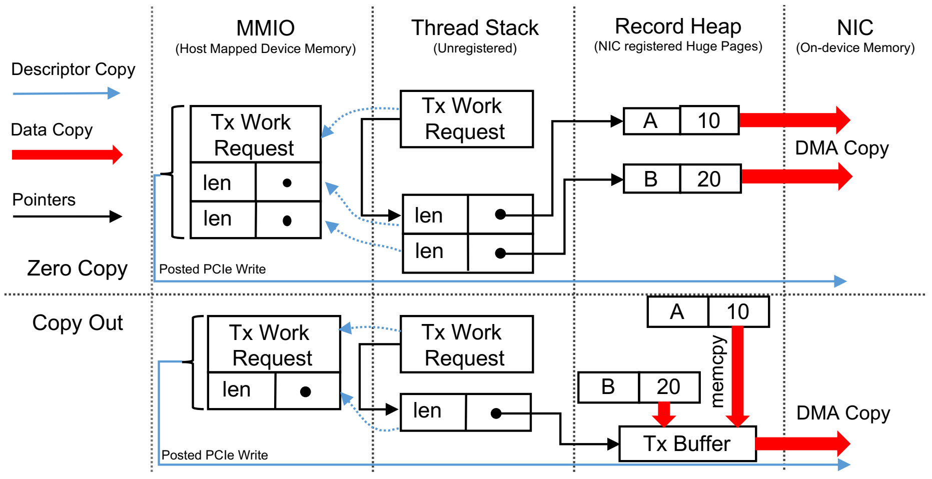

Fig. 1. Key structures involved in transmission for both zero-copy and copy-out. With zero-copy, transmit descriptors list several chunks of data for the NIC to DMA. With copy-out, all data to be transmitted is first explicitly copied into a transmit buffer by the host CPU; then, a transmit descriptor is posted that references just the transmit buffer rather than the original source data.

host load. One key feature is that the NIC can directly transmit application data rather than copying data from userspace to kernel buffers; this is called "zero copy". (Despite the colloquial zero-copy name, the NIC must still copy the data via DMA to on-device buffers and "copy" those bits onto the network cable.)

More interestingly, zero-copy isn't constrained to sending simple contiguous buffers. Because zero-copy already relies on the NIC's DMA capabilities, the application can provide a *gather list* to the NIC of chunks of data to be transmitted. The NIC assembles the chunks on-the-fly as it DMAs the data from host memory into NIC buffers. The rearrangement of the data happens "for free", since the NIC must perform DMA to get the data. This makes zero-copy especially promising for applications like databases where responses can be made up of near arbitrary arrangements of small or even variable size records.

Figure 1 details how the application interacts with a Mellanox ConnectX3, a modern 56 Gbps NIC that uses kernel bypass. Both zero-copy and the traditional copy-out approaches to transmission are shown. In both cases the same three key data structures are involved. The first important structure is the data to be transmitted, which lives in heap memory. For zero-copy, the memory where the records live must first be registered with the NIC. Registration informs the NIC of the virtual-to-physical mapping of the heap pages. This is required because the NIC must perform virtual-to-physical address translation since the OS is not involved during transmission and the application has no access to its own page tables. Registration is done at startup and is often done with physical memory backed by 1 GB hugepages to minimize on-NIC address translation costs.

The second key structure is the descriptor that a thread must construct to issue a transmission. With Mellanox NICs, a thread creates a work request and a gather list on its stack. The work request indicates that the requested operation is a transmission, and the gather list is a contiguous list of base-bound pairs that indicate what data should be transmitted by the NIC (and hence DMAed). For zero-copy, the gather list is as long as the number of chunks that the host would like to transmit, up to a small limit. The NICs we use support posting up to 32 chunks per transmit operation. Later, we find that this small limit bottlenecks NIC transmit performance when chunks are small and numerous (§5.1).

The key difference between zero-copy and copy-out is shown with the wide, red arrows in Figure 1. Copy-out works much like conventional kernel-based networking stacks: chunks of data are first copied into a single transmit buffer in host memory. Then, a simple, single-entry descriptor is posted to the NIC that DMAs the transmit buffer to an on-device buffer for transmission. As a result, copy-out requires an extra and explicit copy of the data, which is made by the host CPU. Making the copy uses host CPU cycles, consumes memory bandwidth, and is pure overhead. Surprisingly, though, copy-out has advantages including better performance when records are small and scattered. In those cases, complex gather descriptors bottleneck the NIC, and using the host CPU to pre-assemble the responses can improve performance.

The final important structure is the control interface between the NIC and the host CPU. When the NIC is initially set up by the application, a region of the NIC's memory is mapped into the application's virtual address space. The NIC polls this region, and the host writes new descriptors to it from the thread's stack to issue operations. The region is mapped as write-combining; filling a cacheline in the region generates a cacheline-sized PCIe message to the NIC. The NIC receives it, and it issues DMA operations to begin collecting the data listed in the descriptor. The PCIe messages are posted writes, which means they are asynchronous from the CPU's perspective. Even though PCIe latencies are much higher than DRAM access, the CPU doesn't stall when posting descriptors, so the exchange is very low overhead.

4 Bw-Tree: Lock-free Indexing

To show how these NIC features can be applied to in-memory database systems, we now examine a data structure that is well-positioned to take advantage of them. The Bw-Tree [13] is an atomic record store designed for extreme concurrency. It is an ordered index that supports basic record create, update, and delete (CRUD) operations in addition to search and range scans. It is fully lock-free and non-blocking, and is optimized for modern multicore workloads. It can store to flash, but is also intended for in-memory workloads; it serves as the ordered secondary index structure for in-memory SQL Server Hekaton [5] engine.

In many ways, the Bw-Tree is a conventional paged B-link tree, but it also has unique characteristics that interact with network-layer design choices. Its lock-freedom, its elimination of update-in-place, and its lazily consolidation of

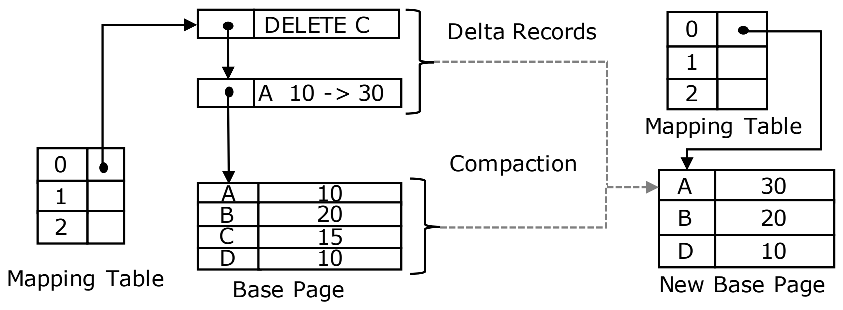

Fig. 2. The Bw-Tree avoids update-in-place by creating delta records "off to the side" that describe a logical modification to a page. Delta records are prefixed to a chain ultimately attached to a base page. When delta chains grow long they are compacted together with the base page to create a new base page.

updated records in virtual memory give it tremendous flexibility in how query results are transmitted by the NIC.

Records may be embedded within leaf pages, or the leaf pages may only contain pointers to data records. When used as a secondary index, typically leaf pages would contain pointers, since otherwise each record would have to be materialized twice and the copies would need to be kept consistent.

The key challenge in a lock-free structure is providing atomic reads, updates, inserts, and deletes without ever being able to quiesce ongoing operations (not even on portions of the tree). Bw-Tree solves this problem by eliminating update-in-place. All mutations are written to newly allocated memory, then the changes are installed with a single atomic compare-and-swap instruction that publishes the change. Figure 2 shows how this works. All references to pages are translated through a mapping table that maps page numbers to virtual addresses. This allows pages to be relocated in memory, and it allows the contents of a page to swapped with a single atomic compare-and-swap (CAS) operation.

One of the key innovations of the Bw-Tree is its use of *delta records*, which make updates inexpensive. Delta records allow the Bw-Tree to logically modify the contents of an existing page without blocking concurrent page readers, without atomicity issues, and without recopying the entire contents of the page for each update. Whenever a mutation is made to a page, a small record is allocated, and the logical operation is recorded within this delta record. The delta record contains a pointer to the page that it logically modifies. It is then atomically installed by performing a CAS operation on the mapping table that re-points the virtual address for a particular page number to the address of the delta record.

Some threads already reading the original page contents may not see the update, but all future operations on the Bw-Tree that access that page will see the delta record. As readers traverse the tree, they consider the base pages to be logically augmented by their delta records. Delta records can be chained together up to a configurable length. When the chain becomes too long, a new base page is formed that combines the original base page contents with the updates from the deltas. The new page is swapped-in the same way as other updates.

Read operations that run concurrent to update operations can observe superseded pages and delta records, so their garbage collection must be deferred. To solve this, each thread that accesses the tree and each unlinked object is associated with a current *epoch*. The global epoch is periodically incremented. Memory for an unlinked object can be recycled when no thread belongs to its epoch or any earlier epoch. The epoch mechanism gives operations consistent reads of the tree, even while concurrent updates are ongoing. However, there is a cost; if operations take a long time they remain active within their epoch and prevent reclamation of memory that has been unlinked from the data structure.

4.1 NIC Implications for Bw-Tree

Lock-freedom has major implications on the in-memory layout of the Bw-Tree. Most importantly, readers (such as the NIC DMA engine) can collect a consistent view of the tree without interference from writers, and holding that view consistent cannot stall concurrent readers or writers to the tree. This natural record stability fits with the zero-copy capabilities of modern NICs; because the NIC's DMA engine is oblivious to any locks in the database engine, structures requiring locking for updates would have to consider the NIC to have a non-preemtible read lock for the entire memory region until the DMA completes. Instead of copying records "out" of the data structure for transmission, records can be accessed directly by the NIC. Eliminating the explicit copy of the records into transmit buffers can save database server CPU and memory bandwidth.

Page consolidation can also benefit the NIC and improve performance. Records in the Bw-Tree are opportunistically packed into contiguous pages in virtual memory, but the view of a page is often augmented with (potentially many) small delta records that are scattered throughout memory. A database might save CPU and memory bandwidth by more aggressively deferring or even eliminating consolidation of records into contiguous regions of memory or pages. We show in §5.4 that highly discontinuous data can slow transmission throughput but that aggressive consolidation is inefficient; delta records can dramatically reduce consolidation overheads while keeping records sufficiently contiguous to make the NIC more efficient.

Overall, we seek to answer these key questions:

- When should records be transmitted directly from a Bw-Tree? Are there cases where explicitly copying records into a transmit buffer is preferable to gather DMA operations?
- How aggressive should a Bw-Tree be in consolidating records to benefit individual clients and to minimize database server load?
- How does zero-copy impact state reclamation in the Bw-Tree? Might long transmit times hinder performance by delaying garbage collection of stale records?

CPU	Intel Xeon E5-2450 (2.1 GHz, 2.9 GHz Turbo)
	8 cores, 2 hardware threads each
RAM	16 GB DDR3 at 1600 MHz
Network	Mellanox MX354A ConnectX-3 Infiniband HCA (56 Gbps Full Duplex)
	Connected via PCIExpress 3.0 x8 (63 Gbps Full Duplex)
Software	CentOS 6.6, Linux 2.6.32, gcc 4.9.2, libibverbs 1.1.8, mlx4 1.0.6

Table 1. Experimental cluster configuration. The cluster has 7 Mellanox SX6036G FDR switches arranged in two layers. The switching fabric is oversubscribed and provides about 16 Gbps of bisection bandwidth per node when congested.

5 Experimental Results

To explore how different designs trade-off database server efficiency and performance, we built a simple model of an in-memory database system that concentrates on data transfer rather than full query processing. In all experiments, one node acts as a server and transmits results to 15 client nodes. Our experiments were run on the Apt [18] cluster of the CloudLab testbed: this testbed provides exclusive bare-metal access to a large number of machines with RDMA-capable Infiniband NICs. Details of the configuration are given in Table 1. All of the experiments are publicly available online[1].

Experiments transmit from a large region of memory backed by 4 KB pages that contains all of the records. The region is also registered with the NIC, which has to do virtual-to-physical address translation to DMA records for transmission. In some cases, using 1 GB hugepages reduces translation look aside buffer (TLB) misses. The NIC can benefit from hugepages as well, since large page tables can result in additional DMA operations to host memory during address translation [6, 9]. For our experiments, the reach of the NIC's virtual-to-physical mapping is sufficient, and hugepages have no impact on the results.

5.1 Zero-copy Performance

The first key question is understanding how database record layout affects the performance of the transmission of query results. The transmission of large result sets presents a number of complex choices that affect database layout and design as well as NIC parameters. Range query results can be transmitted in small batches or large batches and either via copy-out or zero-copy.

To understand these trade-offs, we measure the aggregate transmission throughput of a server to its 15 clients under several configurations. In each experiment, the record size, s, is either 1024 B or 128 B. Given a set of records that must be transmitted, they are then grouped for transmission. For zero-copy, an n entry DMA gather descriptor is created to transmit those records where ns

[1] `https://github.com/utah-scs/ibv-bench`

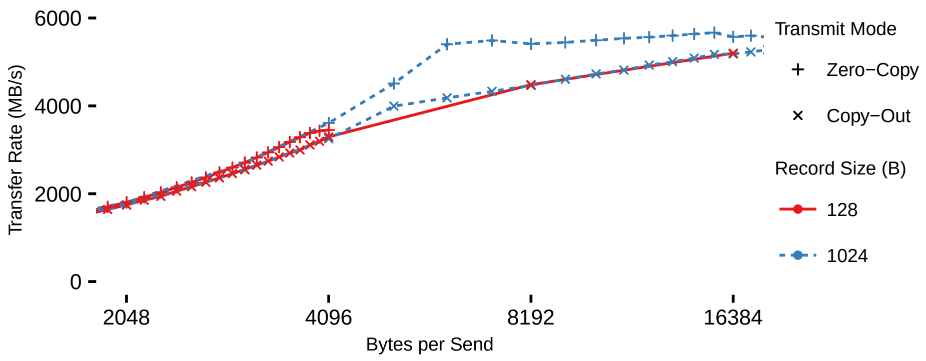

Fig. 3. Transmission throughput when using conventional copy-out into transmit buffers and when the NIC directly copies records via DMA (zero-copy). The line for 128-byte zero-copy stops at 4K, as this is the maximum size of a send with 32 records of this size.

bytes are transmitted per NIC transmit operation. For copy-out, each of the n records is copied into a single transmit buffer that is associated with a transmit descriptor that only points to the single transmit buffer. Each transmission still sends exactly ns bytes, but copy-out first requires ns bytes to be copied into the transmit buffer. Intuitively, larger groups of records (larger sends) result in less host-to-NIC interaction, which reduces host load and can increase throughput; the benefits depend on the specific configuration and are explored below.

Figure 3 shows how each of these configurations impact transmission throughput. For larger 1024 B records, using the NIC's DMA engine for zero-copy shows clear benefits (aside from CPU and memory bandwidth savings, which we explore in §5.2). The database server is able to saturate the network with zero-copy so long as it can post 6 or more records per transmit operation to the NIC (that is, if it sends 6 KB or larger messages at a time).

The figure also shows that using copy-out with 1024 B records, the NIC can also saturate the network, but records must be packed into buffers of 16 KB or more. This is significant, since it determines the transmission throughput of the database when range queries only return a few results. In this case, the DMA engine could provide a throughput boost of up to 29% over copy-out, but the benefit is specific to that scenario. If range scans return even just 16 records per query, the throughput benefits of zero-copy are almost eliminated.

Next, we consider 128 B records. The decreased access latency of in-memory databases makes them well-suited to smaller, finer-grained records than were previously common. One expectation is that this will drive databases toward more aggressively normalized layouts with small records. This seems to be increasingly the case as records of a few hundreds bytes or less are now typical [16, 3].

Figure 3 shows that for small 128 B records, the NIC DMA engine provides little throughput benefit. Our NIC is limited to gather lists of 32 entries, which is insufficient to saturate the network with such a small record size. Transmission

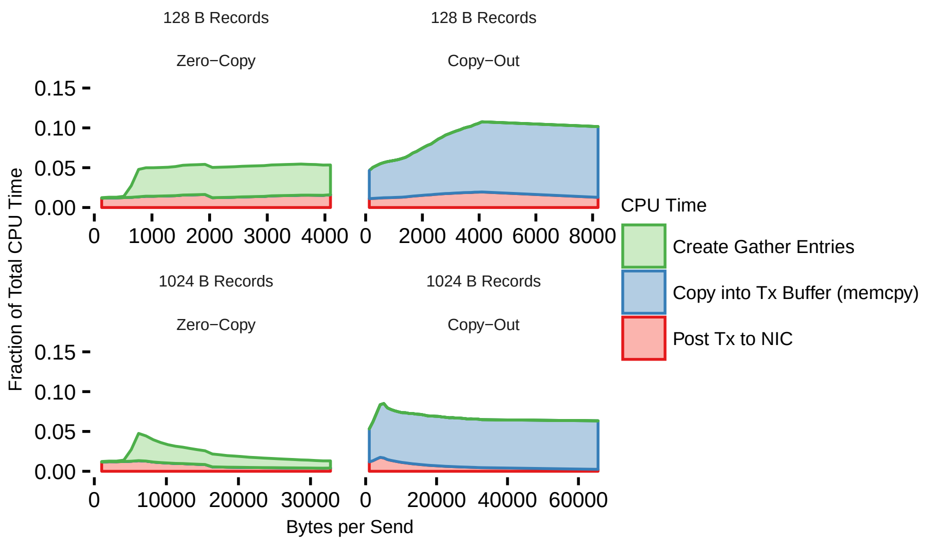

Fig. 4. Breakdown of server CPU time for transmission of small 128 B records and 1024 B records using copies into transmit buffers (Copy-Out) and zero-copy DMA.

peaks at 3.5 GB/s. Copying 128 B records on-the-fly can significantly outperform zero-copy transmission when there are enough results to group per transmission. In fact, copy-out can saturate the network with small records, and it performs identically to copy-out with larger 1024 B records.

5.2 Zero-copy Savings

In addition to improved performance, a goal of zero-copy DMA is to mitigate server-side CPU and memory load. Figure 4 breaks down CPU time for each of the scenarios in Figure 3: small and large records both with and without zero-copy. Most of the server CPU time is spent idle waiting for the NIC to complete transmissions. The results show that zero-copy always reduces the CPU load of the server, and, as expected, the effect is larger for larger records. With 1024 B records, the `memcpy` step of copy-out uses 6.8% of the total CPU cycles of the socket at peak transmission speed.

Zero-copy eliminates copy overhead, but it adds overhead to create larger transmit descriptors. Each gather entry adds 16 B to the descriptor that is posted to the NIC. These entries are considerable in size compared to small records, and they are copied twice. The gather list is first staged on the thread's stack and passed to the userlevel NIC driver. Next, the driver makes a posted PCIe write by copying the descriptor (including the gather entry) into a memory-mapped device buffer. For large records, constructing the list uses between 1 and 4% of total CPU time, so zero-copy saves about 3 to 6% of CPU time over copy-out.

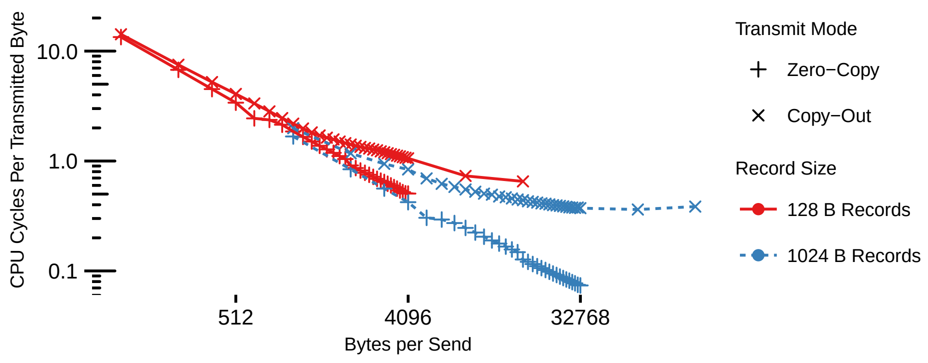

Fig. 5. Cycles per transmitted byte for large and small records with zero-copy and copy-out. Note the log-scale axes.

The memory bandwidth savings for zero-copy are more substantial. Figure 3 shows that copy-out transmit performance nearly matches zero-copy (5.6 GB/s versus 5.8 GB/s). Copy-out introduces exactly one extra copy of the data, and `memcpy` reads each cache line once and writes it once. So, copy-out increases memory bandwidth consumption by 2× the transmit rate of the NIC or 11.2 GB/s in the worst case. This accounts for about 32% of the available memory bandwidth for the machine that we used. Whether using zero-copy or copy-out, the NIC must copy data from main memory, across the PCIe bus, to its own buffers, which uses another 6 GB/s of memory bandwidth.

For smaller 128 B records, the CPU and memory bandwidth savings are nearly the same as for larger records. In this case, copy-out uses up to 10.8% of the CPU cycles on the socket, and zero-copy uses 5.5%. Eliminating the extra `memcpy` halves CPU overhead for transmission; a savings of about 5% of the total socket's CPU cycles. Just as before, the memory bandwidth savings is twice the transmission rate or 11.2 GB/s. Overall, this makes copy-out reasonable for small records scattered throughout memory, especially since zero-copy cannot saturate the network in these cases (§5.1).

The results break down where the CPU and memory bandwidth savings come from, but not all configurations result in the same transmit performance. For example, Figure 3 shows that when transmitting 128 B records, copy-out gets up to 53% better throughput than zero-copy. As a result, minimizing CPU overhead can come at the expense of transmit performance. The real efficiency of the server in transmission is shown in Figure 5. The figure shows how many cycles of work the CPU does to transmit a byte in each of the configurations, which reveals two key things. First, it shows that, though the absolute savings in total CPU cycles is small for zero-copy, it does reduce CPU overhead due to transmission by up to 76%. Second, the improvement is a modest 12% for small records, since copy-out can transmit in larger, more efficient batches.

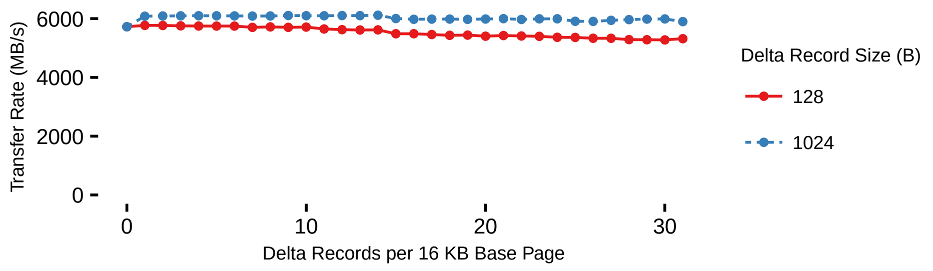

Fig. 6. Zero-copy transmit performance when a single 16 KB base page is transmitted at along with a varying number of delta records. Results are shown for delta records of both 128 B and 1024 B.

5.3 Extending the Delta Format to Clients

The experiments so far consider delivering a clean, ordered set of records to the client. That is, the server collects and transmits the precise set of records in the correct order, either via copy-out or zero-copy. Another option is to transmit base pages along with their deltas and have clients merge the results. This approach is attractive because it organically fits with the design of the Bw-Tree and it suits the DMA engine of the NIC well. The NIC can transmit the large contiguous base pages (16 KB, for example) with little CPU overhead. It also eliminates copy-out for small records, but avoids the transmission throughput ceiling longer gather lists suffer (§5.1). Merging records on the client side is cheap; the server can even append them to the response in a sort order that matches the record order in the base page for efficient $O(n)$ iteration over the records that requires no copies or sorting.

Figure 6 shows the benefits of this approach. In this experiment, each transmission consists of a single 16 KB base page while the number and size of the delta records attached to each transmission is varied. The NIC benefits from the large base page, and it manages to keep the network saturated. CPU overhead ranges from less than 1% when there are a few delta records up to about 2% when there are more. Compared to zero-copy of scattered small records this approach also yields a 1.7× throughput advantage; Figure 3 shows that throughput peaks at 3.4 GB/s when records are scattered, while the delta format achieves 5.7 GB/s. The main cost to this approach is the cost of consolidating delta records into new base pages when chains grow long; we consider this overhead in more detail in the next section.

5.4 Tuning Page Consolidation

Bw-Tree and many indexing structures are paged in order to amortize storage access latency, and paging can have similar benefits for network transmission as well. However, the userlevel access of modern NICs makes interacting with them much lower-latency than storage devices. This raises the question of whether paging is still beneficial for in-memory structures. That is, is the cost of preemptively consolidating updates into pages worth the cost, or is it better to transmit fine-grained scattered records via zero-copy or copy-out?

The results of our earlier experiments help answer this question. If copy-out is used to gain faster transmission of small records, then the answer is simple. Even if every update created a complete copy of the full base page, the extra copies would still be worthwhile so long as more pages are read per second than updated. This true for most update/lookup workloads, and read-only range queries make this an even better trade-off.

However, consolidation must be more conservative when using zero-copy to yield savings, since zero-copy can collect scattered records with less overhead than copy-out. Yet, there is good reason to be optimistic. Delta records reduce the cost of updates for paged structures. If each base page is limited to d delta records before consolidation, the number of consolidations is $\frac{1}{d}$. This means that allowing short delta chains dramatically reduces consolidation costs, while longer chains offer decreasing returns. This fits with the NIC's preference for short gather lists; allowing delta chains of length 4 would reduce consolidation by 75% while maintaining transmit throughput that meets or exceeds on-the-fly copy-out. The large 16 KB base pages also improve transmit throughput slightly, which improves efficiency.

For small records, transmitting compacted base pages that have a few deltas gives a total CPU savings of about 8%. For the same CPU cost, a server can perform about 5.2 GB/s of page compaction or about 340,000 compactions per second for 16 KB pages. Even in the worst case scenario where all updates are focused on a single page, delta chains of length 4 would tolerate 1.3 million updates per second with CPU overhead lower than copy-out. So, delta records can give the efficiency of zero-copy with the performance of copy-out.

5.5 Impact on Garbage Collection

Using zero-copy tightly couples the higher-level database engine's record allocation and deallocation, since records submitted must remain stable until transmission completes. Fortunately, existing mechanisms in systems that avoid update-in-place accommodate this well, like Bw-Tree's epoch mechanism described in Section 4. Normally, the Bw-Tree copies-out returned records. While an operation and its copy-out are ongoing, memory that contains records that have been unlinked or superseded cannot be reclaimed. Zero-copy effectively defers the copy until the actual time of transmission. As a result, transmissions completions must notify the higher level database of transmission completion to allow reclamation. This delay results in more latent garbage and hurts the effective memory utilization of the system.

In practice, this effect will be small for two reasons. First, zero-copy adds a transmission, which completes within a few microseconds; however, it also saves a `memcpy` which accounts for 1 to 2 µs for a 16 KB transmission. Second, the amount of resulting data held solely for transmission should generally be more than compensated for by eliminating the need for explicit transmit buffers.

With copy-out, the size of the transmit buffer pool is necessarily larger than the amount of data under active transmission, and internal fragmentation in the transmit buffers makes this worse.

5.6 Inlining Data Into Transmit Descriptors

Mellanox NICs allow some data to be *inlined* inside the control message sent to the NIC over PCIe. Our NICs allow up to 912 B to be included inside the descriptor that is posted to the NIC control ring buffer. Inlining can improve messaging latency by eliminating the delay for the NIC to DMA the message data from host DRAM, which can only happen after the NIC receives the descriptor. Inlining benefits small request/response exchanges, but it does not help for larger transmissions. This is because even though there is an extra delay before the NIC receives the actual record data, that delay can be overlapped with the DMA and transmission of other responses. Other researchers have shown that sending data to the NIC via MMIO also wastes PCIe bandwidth [9]. All experiments in this paper have inlining disabled. Enabling inlining gives almost identical throughput and overhead to copy-out, except it only works for transmissions of 912 B or less.

6 Conclusions

Our findings show that understanding NIC performance has significant consequences for modern in-memory databases, and that careful co-design of data layout, concurrency control, networking impacts both performance and efficiency. Summarizing the key findings of our experiments:

- The sophisticated DMA capabilities of modern NIC's can be used to transmit directly from database records without any intermediate copying.
- For workloads with small records of a few hundred bytes or less where no significant computation is done server-side, using conventional copy-out transmission will yield better overall throughput even when the result sets are large.
- Zero-copy yields only a few percent total CPU savings, but may reduce server memory bandwidth load by one-third.
- Increasing core counts mean zero-copy CPU savings will become less significant over time, as per-server network speeds are likely remain in the 100s of gigabits per second for some time.
- The memory bandwidth savings are likely to be more important, but memory bandwidth is still growing faster than network bandwidth.
- Zero-copy is highly efficient when records can be transmitted from large contiguous blocks.

Revisiting the key questions posed by the Bw-Tree's unique structure, we apply our findings.

▶ When should records be transmitted directly from a Bw-Tree? Are there cases where explicitly copying records into a transmit buffer is preferable to gather DMA operations?

Gather list length limitations means that copy-out yields better throughput than zero-copy for small, scattered records. For example, this makes copy-out preferable when the tree is a secondary index that only holds pointers to records. Zero-copy always results in less server-side load, so zero-copy is preferable if the server also hosts computationally heavy operations. However, the load reduction is small and may be offset by increased delays at the clients.

▶ How aggressive should a Bw-Tree be in consolidating records to benefit individual clients and to minimize database server load?

If clients can receive and apply delta records themselves, then using zero-copy to directly transmit base pages and their deltas results in the best throughput and the lowest CPU load. Section 5.4 shows that short delta chains of up to length 4 will sustain millions of updates per second to a database with lower CPU load than either copy-out or zero-copy for transmission.

▶ How does zero-copy impact state reclamation in the Bw-Tree? Might long transmit times hinder performance by delaying garbage collection of stale records?

Transmission delays garbage collection of unlinked records, but the amount of accumulated state will be less than the amount that traditional transmit buffers would consume. This effect is compounded by the number of ongoing client operations, so Bw-Tree's ability to leverage zero-copy could improve fan-out to large numbers of clients.

Overall, our client-assisted design for bulk returning small records with the Bw-Tree reduces server-side CPU overhead by 75% and increases throughput by 1.7× compared to zero-copy DMA. With care, modern NICs can saturate the network with low overhead while returning 10s of billions of records per second, even with difficult workloads returning dynamic sets of fine-grained records.

We are applying these techniques to a high-bandwidth, low-impact data migration mechanism for RAMCloud that works to evacuate data away from hot servers with minimal disruption to its already overloaded CPU. Looking forward, we also plan to apply these findings to build a network-attached high-performance access method using techniques similar to the Bw-Tree.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1338155.

References

1. DPDK documentation. `http://dpdk.org/doc/guides-16.04/`.
2. Mellanox ConnectX-4 product brief. `http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-4_VPI_Card.pdf`. Accessed: 2016-07-04.
3. B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload Analysis of a Large-scale Key-value Store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and*

Modeling of Computer Systems, SIGMETRICS '12, New York, NY, USA, 2012. ACM.

4. N. Bronson. Personal Communication, 2016.
5. C. Diaconu, C. Freedman, E. Ismert, P. Larson, P. Mittal, R. Stonecipher, N. Verma, and M. Zwilling. Hekaton: SQL server's memory-optimized OLTP engine. In *SIGMOD*, 2013.
6. A. Dragojević, D. Narayanan, M. Castro, and O. Hodson. FaRM: Fast Remote Memory. In *USENIX NSDI*, Seattle, WA, Apr. 2014. USENIX Association.
7. A. Dragojević, D. Narayanan, E. B. Nightingale, M. Renzelmann, A. Shamis, A. Badam, and M. Castro. No compromises: distributed transactions with consistency, availability, and performance. In *SOSP*, 2015.
8. A. Kalia, M. Kaminsky, and D. G. Andersen. Using RDMA Efficiently for Key-value Services. In *Proceedings of the 2014 ACM SIGCOMM Conference*, SIGCOMM '14, New York, NY, USA, 2014. ACM.
9. A. Kalia, M. Kaminsky, and D. G. Andersen. Design Guidelines for High Performance RDMA Systems. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, Denver, CO, June 2016. USENIX Association.
10. A. Kejriwal, A. Gopalan, A. Gupta, Z. Jia, S. Yang, and J. Ousterhout. SLIK: Scalable Low-Latency Indexes for a Key-Value Store. In *USENIX Annual Technical Conference*, Denver, CO, June 2016.
11. J. Levandoski, D. Lomet, S. Sengupta, R. Stutsman, and R. Wang. High Performance Transactions in Deuteronomy. In *Proc. of CIDR*, 2015.
12. J. Levandoski, D. Lomet, S. Sengupta, R. Stutsman, and R. Wang. Multi-Version Range Concurrency Control in Deuteronomy. *Proc. VLDB Endow.*, 2016.
13. J. J. Levandoski, D. B. Lomet, and S. Sengupta. The Bw-Tree: A B-tree for New Hardware Platforms. In *International Conference on Data Engineering (ICDE)*. IEEE, 2013.
14. H. Lim, D. Han, D. G. Andersen, and M. Kaminsky. MICA: A Holistic Approach to Fast In-Memory Key-Value Storage. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, Seattle, WA, Apr. 2014.
15. S. Loesing, M. Pilman, T. Etter, and D. Kossmann. On the Design and Scalability of Distributed Shared-Data Databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, New York, NY, USA, 2015. ACM.
16. R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling Memcache at Facebook. In *Symposium on Networked Systems Design and Implementation (NSDI)*, Lombard, IL, 2013. USENIX.
17. J. Ousterhout, A. Gopalan, A. Gupta, A. Kejriwal, C. Lee, B. Montazeri, D. Ongaro, S. J. Park, H. Qin, M. Rosenblum, S. Rumble, R. Stutsman, and S. Yang. The RAMCloud Storage System. *ACM Transactions on Computer Systems*, 33(3):7:1–7:55, Aug. 2015.
18. R. Ricci, G. Wong, L. Stoller, K. Webb, J. Duerig, K. Downie, and M. Hibler. Apt: A platform for repeatable research in computer science. *ACM SIGOPS Operating Systems Review*, 49(1), Jan. 2015.
19. W. Rödiger, T. Mühlbauer, A. Kemper, and T. Neumann. High-speed Query Processing over High-speed Networks. *Proc. VLDB Endow.*, Dec. 2015.
20. X. Wei, J. Shi, Y. Chen, R. Chen, and H. Chen. Fast In-memory Transaction Processing Using RDMA and HTM. In *Proceedings of SOSP*, SOSP '15, New York, NY, USA, 2015. ACM.