MAKING LARGE TRANSFERS FAST FOR IN-MEMORY DATABASES IN MODERN NETWORKS

by

Aniraj Kesavan

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

School of Computing

The University of Utah

May 2017

The University of Utah Graduate School

STATEMENT OF THESIS APPROVAL

The thesis of **Aniraj Kesavan**
has been approved by the following supervisory committee members:

Ryan Stutsman,	Chair(s)	**23 Feb 2017** Date Approved
Robert Ricci,	Member	**23 Feb 2017** Date Approved
Feifei Li,	Member	**23 Feb 2017** Date Approved

by **Ross Whitaker**, Chair/Dean of
the Department/College/School of **Computing**
and by **David B Kieda**, Dean of The Graduate School.

ABSTRACT

Efficient movement of massive amounts of data over high-speed networks at high throughput is essential for a modern-day in-memory storage system.

In response to the growing needs of throughput and latency demands at scale, a new class of database systems was developed in recent years. The development of these systems was guided by increased access to high throughput, low latency network fabrics, and declining cost of Dynamic Random Access Memory (DRAM).

These systems were designed with On-Line Transactional Processing (OLTP) workloads in mind, and, as a result, are optimized for fast dispatch and perform well under small request-response scenarios. However, massive server responses such as those for range queries and data migration for load balancing poses challenges for this design.

This thesis analyzes the effects of large transfers on scale-out systems through the lens of a modern Network Interface Card (NIC). The present-day NIC offers new and exciting opportunities and challenges for large transfers, but using them efficiently requires smart data layout and concurrency control.

We evaluated the impact of modern NICs in designing data layout by measuring transmit performance and full system impact by observing the effects of Direct Memory Access (DMA), Remote Direct Memory Access (RDMA), and caching improvements such as Intel® Data Direct I/O (DDIO).

We discovered that use of techniques such as Zero Copy yield around 25% savings in CPU cycles and a 50% reduction in the memory bandwidth utilization on a server by using a client-assisted design with records that are not updated in place.

We also set up experiments that underlined the bottlenecks in the current approach to data migration in RAMCloud and propose guidelines for a fast and efficient migration protocol for RAMCloud.

To my parents and teachers, Kindergarten to the U

CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I consider myself fortunate in more ways than I can count and I thank God profusely for that. Among all the people whom I could thank for every single achievement in my life, my gratitude for my parents, Kesavan Mundathode Pangottil and Sarala Kesavan, surpasses the rest. I have come to realize that the luckiest I have ever been was to be born to my parents who dedicated their entire lives to the betterment of mine in the face of financial hardships and social stigma. They instilled in me the value of a good education early on. No statement of acknowledgment would be complete without thanking them. I would also like to thank Sajini for her continued support and lifting me up whenever I felt like I could not continue. I am excited about our life together, and I look forward to growing old with you.

I owe everything I have learned and accomplished as a graduate student at the U to Ryan Stutsman. Ryan was the rock of my master's education, and if not for his steadfast mentoring and confidence in me despite my inherent skepticism, this work would not have been possible. I understand that it is a common sentiment among students to attribute their success to their advisors, but when I say that I could not have done any of this without Ryan, I mean it from the bottom of my heart. He has been the greatest mentor, advisor, and instructor and his positivity and unparalleled work ethic inspired me to go the extra mile to complete my tasks. He is humble despite his many accomplishments, and he has amazed me with his trust in me during testing times. Ryan's empathy and generousness went a long way in my success here. I consider myself privileged to be one of the first students to graduate under him, and I wish him continued stellar success at the U.

Robert Ricci is the sole reason I decided to attend the U, and no one has been more encouraging to me than Rob during the last two years. He showed me the ways of graduate school and was the forcing function that kept me on track of my progress. He taught me what to focus on while presenting research and I am a better student for it. I could always walk up to his office to get answers to questions on both life and research.

Since his response to my first email before my application up until my thesis defense, Rob was instrumental in my success here. I would also like to thank Feifei Li for sitting on my thesis committee. Despite his busy schedule, Feifei found time to discuss my work and gave indispensable advice both about graduate school and about pursuing a career in Computer Science in addition to imparting invaluable Database wisdom for the thesis.

I only wish that I had met Chinmay Kulkarni earlier. I couldn't have asked for a better labmate, and I have learned a lot from him in the short time we have worked together. I am sure Chinmay will continue to work wonders for our group, and I look forward to his successful career. I would also like to thank Tian Zhang (Candy) and QingKai Lu for their legwork that founded this research. Graduate school would have been a lot different if not for my roommates and longtime friends, Alex Karathra and Rony Gregory. I thank them for all their encouragement and support. To err on the side of caution when it comes to missing names, I would like to thank all my friends who were supportive of my decision to quit my job and come back to school.

I have made plenty of decisions in my brief academic and professional career and attending the University of Utah was the single most rewarding decision that I have made so far. Like any other student, I have had the pleasure of expanding my horizon of knowledge exponentially after every stage of my education. I would like to thank all my teachers from my early schooling days at Vijayamatha and T.H.S.S and later at Govt. Model Engineering College. I would like to thank my mentors at Zynga and Dreamworks Animation who taught me to push further, especially Shan Kadavil, Binu Philip, Vyas Thottathil, Manik Taneja, Blessan Abraham, and Satheesh Subramanian.

This material is based upon work supported by the National Science Foundation under Grant No CNS-1566175 and Grant No CNS-1338155. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

CHAPTER 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 Input/Output (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 database I/O is solved; however, another device now dictates performance: the Network Interface Card (NIC). As the primary I/O device in present-day databases, the NIC should be a first-class citizen in all phases of database design. However, modern network cards have grown incredibly complex, partly in response to demands for high throughput and low latency.

We set out to improve the efficiency of RAMCloud's [55] data migration mechanism. RAMCloud is a storage system that keeps all data in DRAM at all times and provides low latency access by leveraging advancements like kernel bypass that are present in a modern NIC. To thoroughly understand the complexities involved in the NIC in doing huge transfers of data, we profiled a modern NIC with a focus on massive data transfers. We made several discoveries that informed our design for RAMCloud, but they also generalize to other operations involving large transfers of data in a distributed in-memory database.

This thesis is structured into six key pieces. Chapter 1 gives an introduction to the problem space and the motivation for this work. It also outlines an overview of all the experiments and key contributions. Chapter 2 introduces our notion of "**Zero Copy**" and "**Copy Out**" and explains the lessons we learned when we developed a microbenchmark to profile a modern NIC with kernel bypass and RDMA doing bulk transfers of data. The chapter also provides a detailed study of the impact on the available system resources while we achieve near the line rate of network transmission performance. Where possible, recommendations for a database designer are provided, and takeaways from our experiments are explicitly listed. Chapter 3 enumerates a few scenarios where we can leverage our observations to improve server efficiency and network transmission using a client-assisted design. We quantify the benefits of such an approach and discuss future directions on how this design will apply to column-oriented databases. Chapter 4 discusses how the nuanced performance characteristics for the NIC can influence design for rapid reconfiguration in RAMCloud [55]. It discusses a breakdown of RAMCloud's existing migration mechanism to understand its bottlenecks and proposes the basics of a new design for migration that we believe will advance the state-of-the-art in the reconfiguration of distributed in-memory databases. Chapter 5 discusses prior work that laid the foundation for this research and the state-of-the-art in in-memory databases and data transfer in high-speed networks. Chapter 6 reiterates our conclusions and discusses future directions for this work.

While developing the microbenchmark to analyze a NIC, one thing was becoming increasingly clearer to us. Modern network cards have grown incredibly complex, partly in response to demands for high throughput and low latency. One key advancement these NICs made is the presence of some form of kernel bypass where the NIC Direct Memory Access (DMA) engine can be controlled by the application, and the network card can access application memory without invoking the CPU. This development led to the Remote Direct Memory Access (RDMA) [25] paradigm. The modern NIC with its complex architecture makes understanding its characteristics difficult, and it makes designing software to take advantage of them even harder. Database designers will encounter many trade-offs when trying to use the NIC effectively. This thesis helps navigate the complex trade-off space with concrete suggestions backed by measurements.

Our mission was to come up with a migration protocol that advances the state-of-the-art for in-memory storage systems. We realized the importance of making the data transfer NIC friendly and set out to measure what variables impact the transmission performance and system stability. We wanted to enumerate and underline the challenges in designing keeping NIC as a first-class citizen. We specifically looked at five 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* which are commonplace in in-memory stores for two reasons:

  1. Transmit descriptors grow linearly in the number of records;
  2. NIC hardware limits descriptor length, restricting 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.
- Absolute savings in *consumed memory bandwidth* is far greater than the CPU savings offered by Zero Copy.
- 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).
- *Direct Cache Access*, now rephrased as Intel®Data Direct I/O [4], provides direct access to last level cache from an I/O device. This effect impacts the consumed memory bandwidth significantly.

These factors make advanced NIC DMA difficult to use effectively. We set out to explore data transmission purely from the perspective of a modern NIC. We profiled a cluster of machines with Infiniband ConnectX®-3, a modern kernel bypass capable NIC with RDMA capabilities, with specific attention to large transfers and query responses. We published our findings and made some concrete suggestions around how NICs might influence data layout and concurrency control in a modern in-memory database [35].

Our primary results gave us ideas about how the Zero Copy paradigm will perform against the traditional approach of transmitting via larger buffers. We also made some optimizations to our benchmarks to better leverage the effects of DDIO [4], which are explained in Section 2.6. We also measured memory bandwidth and a few other metrics with the help of Intel®'s Performance Monitoring Unit (PMU).

Another operational burden while using these newer in-memory storage systems is to find a way to provide effective reconfiguration. Our results from the benchmark highlighted the importance of hardware considerations in data migration. State-of-the-art systems like RAMCloud have around 1000× lower latency than traditional storage systems, and these tight Service Level Agreements (SLA) make it harder to do migrations without interfering with regular requests. The need for rapid reconfiguration with minimal system impact is one of the key hurdles to overcome before the mainstream acceptance of large scale in-memory storage systems.

Our findings on modern NICs inform a new design for cluster reconfiguration in the RAMCloud storage system. Low-cost load balancing improves RAMCloud's efficiency since indexes [33] and transactions [37] benefit from collocation between servers. This is tricky to implement, and RAMCloud is designed with no locality within a server to provide high memory utilization [62] via efficient log compaction. The more interesting engineering detail is RAMCloud's exceedingly tight SLAs (median access time of 5 µs and 99.9% of reads within 10 µs). It would be great if we can saturate the available NIC bandwidth which is 8-12GB/s [5, 6], but the state-of-the-art transfer mechanism offers many orders of magnitudes lower than that at 10 MB/s throughput while operating at latencies that are 1000-10000× higher than that of RAMCloud [55]. We wanted our protocol to strike a balance between fast operation and minimal disruption without drastically affecting the latency and throughput available for the system. The following key principles guide our design for fast and efficient migration for RAMCloud:

- **High throughput**: State-of-the-art systems do migrations at 5 - 10 MB/s. Our proposed protocol should result in 100-1000x faster data migrations. We aspire to have rapid reconfiguration that will help with enabling frequent reconfiguration on live clusters.
- **Low impact on latency**: RAMCloud has 99.9 percentile read latency at < 10 µs and write latency at < 100 µs. We propose a protocol that will introduce only a minimal impact on the normal case operation, still keeping SLAs that are 100x tighter than other systems [21].
- **Judicious use of system resources**: In an in-memory store, the significance of resources such as DRAM capacity and memory bandwidth are paramount. Our

protocol should make the most efficient use of these resources.

- **Late-partitioning**: RAMCloud promises uniform access latency regardless of the level of data locality. RAMCloud also does not need to pre-partition data for normal operations. Most other systems diverge from this approach to make data movement faster. An ideal solution should provide uniform access latency and high DRAM utilization without needing early partitioning decisions.

We evaluate the current state-of-the-art approaches to migration in the RAMCloud storage system. We lay out some experiments that highlight issues with current approaches and motivates our new fast and efficient migration protocol. To motivate and demonstrate our approach, we explore the following experiments:

- Measuring impacts of *locality of data* in RAMCloud; we set up an experiment to show how the locality of data affects the collective load and throughput on the system.
- *Factor analysis of migration bottlenecks*; RAMCloud's current migration protocol, while orders of magnitude faster than existing state-of-the-art protocols, leaves more to be desired in terms of achievable performance. We analyze the various bottlenecks in the current protocol and the performance benefit we gain by enumerating savings on avoiding each one.

1.1 Contributions

"Scale-out in-memory stores are optimized for small requests under tight SLAs, and bulk data movement, for rebalancing and range queries, interfere; the thesis argues carefully leveraging data layout and advancements in modern NICs will yield gains in performance and efficiency for large transfers in these systems without disrupting their primary obligations."

We have shown that careful co-design of data layout, concurrency control, and networking can lead to 95% reduction of CPU overhead (25% absolute CPU savings) and a 50% reduction in consumed memory bandwidth, all the while getting a throughput boost of 1.6× compared to blindly using the kernel bypass available in the NICs. We also profiled the bottlenecks in the way data migration is set up in RAMCloud and in conjunction with the benchmarks. Though there were efforts to dismantle cost of a database in the network layer before, the server-side impact on mixed workloads including range scans and migrations have not been well understood until now. In the light of our discoveries, we make concrete suggestions on how to design a fast and efficient migration protocol which causes minimal disruption.

The following are the main contributions from this work which support and provide evidence for the thesis statement in the beginning of this section.

- **Evaluation of influence of modern NICs on data layout**: To our knowledge, this is the first published research exploring the scatter gather DMA engine of the NIC in contrast with the traditional approach of transmission in the context of bulk transfers in in-memory databases. We profiled the transmit performance of a modern NIC in depth to show the influence of NICs on data layout. We have found a few arguments against the blind use of the new kernel bypass approach considering the overhead of concurrency management and difficulty of implementation.
  - Developed a microbenchmark to assess the performance of Infiniband verbs.
  - Measured transmission throughput and CPU overhead in the face of varying data layouts.
  - Compared the performance of Zero Copy and Copy Out mechanisms of transmitting data from an RDMA capable modern NIC.
- **Design for client-assisted materialization for better transmit performance**: Our measurements inform a new design for migration in RAMCloud, but it also has impacts on other database operations. While evaluating the results, we saw that there are certain structures and circumstances that can uniquely take advantage of the hardware limited scatter gather entries of a modern NIC. Chapter 3 explores ways in which a modern B-tree index can exploit NIC hardware. Through the benchmark, we show 60% improvement in throughput using such a design with a 30$\times$ reduction in CPU overhead.
- **Evaluation of server impact during transmission**: We measure performance counters outside of CPU cores that reveal the impact on memory bandwidth, DDIO traffic, and Peripheral Component Interconnect Express (PCIe) traffic in a system that's churning out heavy responses. This is the first study to our knowledge that does an in-depth analysis of server impact using uncore events for large data

transmissions near the line rate in a kernel bypass capable modern NIC.

  - Profiled the number of cache lines read and written (including the pre-fetcher effect) to measure consumed memory bandwidth.
  - Profiled the number of DRAM accesses from Last Level Cache (LLC) that results from DDIO.
  - Profiled the number of DRAM accesses from Last Level Cache (LLC) that results from PCIe errors.

- **Experiments to motivate fast and efficient reconfiguration**: This is the first assessment of data migration on SLAs for a high-performance, in-memory store that leverages high throughput, kernel bypass networking fabrics. We laid out experiments that show how even the state-of-the-art systems for cluster reconfiguration becomes suboptimal with its locality constraints and overall slowness. RAMCloud assumes uniform access cost to all nodes. We quantify the gains from explicitly colocating records. Our experiments call for fast and efficient migration protocol especially suited for an in-memory database system with a decentralized log structured data model such as RAMCloud.
- **Preliminary design for a migration protocol**: After careful analysis of the current migration protocol, we discovered the critical bottlenecks in the current design that make it infeasible for fast and efficient migration. We propose the basics of a migration protocol that will effectively eliminate these bottlenecks and pave the way forward for fast and efficient transfer of huge chunks of data.

CHAPTER 2

DRAWING LESSONS FROM MODERN NICS

Network Interface Cards have grown complex and present opportunities and challenges as part of a modern distributed system. It is imperative that today's database system designer considers the network in design decisions. Previous research for optimizing databases was focussed more on processing time spent on a database query [11], but today, most of the end-to-end latency while processing a query is spent in the networking stack [54]. Since in-memory databases became popular and datasets started to fit in memory, the spinning disks that were the primary I/O bottlenecks before are not anymore. The bulk of I/O overhead in a distributed storage system today is in network transmission. Adding this to the fact that the modern NIC allows us to offload CPU from the data transfer path, the saved memory bandwidth and CPU cycles could be doing useful work.

Hardware overheads have thinned over time, and system researchers had to minimize CPU interrupts and come up with forms of kernel by-pass networking to extract more performance. Network cards have had Transmission Control Protocol (TCP) offloading baked in as a TCP Offload Engine for some time now. They helped reduce TCP protocol burden from the main CPU by moving that processing to network adapters. However, the ability to bypass copies was dependent on the TCP Offload Engine (TOE) design and in many cases did not support non-copying for incoming data streams [48]. RDMA [25, 30, 57] capable NICs support non-copying of incoming data streams over a wider variety of applications where TCP offloading was just one among the benefits involved. Development of advanced host-device interconnects such as PCI Express [14] made it possible to develop highly performant network devices offering throughputs and latencies orders of magnitude better than what was possible before.

The key consideration we explore in this chapter is how a database server can make the best use of the NIC's ability to receive and transmit data directly at the application level, especially for the cases when the system must move large chunks of data. We evaluated the use of various RDMA primitives for benchmarking large transfers and explored the various trade-offs and design choices while using advanced NIC features and drew lessons from our evaluation of those choices with the help of a microbenchmark.

2.1 Zero Copy and Copy Out

The key benefit of a modern NIC in contrast with earlier hardware is the ability to manipulate control structures of the NIC from userspace applications. Kernel bypass is a ubiquitous feature in the modern NIC that avoids CPU in the data transfer path. Literature defines kernel bypass as communication that does not involve the kernel in the critical path [68]. Newer network controllers also make use of vectored I/O, otherwise known as scatter/gather I/O where the NIC can receive and transmit directly from application vectors. The Mellanox Infiniband ConnectX®-3 NIC that we profiled can collect scattered chunks of data in memory via Direct Memory Access (DMA).

Before kernel bypass became commonplace, the steps for copying data over the network using socket programming required the host CPU to make additional copies of transmitted data. The application calls a send function in a socket library and the data for sending is assembled in a transmit buffer, this transmit buffer is copied on to the on-NIC buffers, and the data is sent over the wire. We could employ the same technique in the kernel bypass capable NIC as well. If we exploit the scatter/gather list to transmit data directly from where it lives in memory, we could reduce the number of times data is copied within the host before transmission. This style of transmitting data which minimizes the number of copies is what we will refer to as "**Zero Copy**" from now on. Instead of employing a multi-entry scatter/gather list, the data to be transmitted, which could be discontiguous in memory, could be assembled onto a temporary buffer and this buffer could be treated as a single entry scatter/gather list and then transmitted over the network. We will be calling this technique "**Copy Out**" for the remainder of the thesis.

Zero Copy DMA facilitates a scatter gather list of buffer descriptors that could be mapped to discontiguous locations in memory on demand. These provide the added benefit that network headers do not need to exist along with data and help bring more flexibility in deciding the transport layer. One subtle thing to note here is that Zero Copy only implies the absence of a copy of the records to a buffer. The data still needs to be copied to the on-NIC buffers before it is sent out via the network cable. This approach is different from what happens when we call socket send on a traditional stack where the data for transmission is pre-assembled in a large buffer in main memory and then copied across the network. We evaluate how Zero Copy performs in contrast with the more traditional Copy Out from now on.

2.2 Memory Bandwidth

Interestingly, the additional copy involved in Copy Out hurts memory bandwidth of the system more than contributing to additional CPU load purely from the perspective of network transmission. Our evaluation in Section 2.8.1 shows us that while the traditional copy adds only up to 18% increase in overhead of total CPU utilization, the effect on memory bandwidth is worse. If we were to make use of the high throughput available in NICs by transmitting near line rate using Copy Out, the transmission takes up 2$\times$ more memory bandwidth which accounts for half of the total available memory bandwidth in a modern server. We should read this in the context that network transmission is not the primary responsibility of a storage server in a distributed system and costs of communication should be treated as overhead. Most of the memory bandwidth is wasted in just aggregating the data before transmission could be used as part of actual computation or other useful rearrangements of data in the query response. Section 2.8 will fully discuss the impact on memory bandwidth and other parameters while the system is transmitting large amounts of data.

2.3 NIC Structures in Detail

Figure 2.1 details how an application interacts with a Mellanox ConnectX-3®, a modern 56 Gbps NIC that uses kernel bypass. 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 source data. 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 page tables. Registration is done when the beginning of the benchmark 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.

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 cache line in the region generates a cache line 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 does not stall when posting descriptors, so the exchange is very low overhead.

2.3.1 Structural Differences in Zero Copy and Copy Out

The key difference between Zero Copy and Copy Out is shown with the wide, red arrows in Figure 2.1. Copy Out works much like a conventional kernel-based networking stack: chunks of data are first copied into a single transmit buffer in host memory, usually by a `memcpy` call to copy from the record's virtual address to the transmit buffer. 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.

2.4 DDIO

One conflating factor in understanding the benefits of Zero Copy is Data Direct I/O (DDIO) [4]. DDIO is a performance-oriented enhancement to the DMA mechanism, which was introduced in Intel®Xeon E5 processors with Sandy Bridge-EP micro-architecture. With DDIO, CPU caches are used as the primary source and destination for I/O, allowing NICs to talk directly to the last level caches of local CPUs and avoiding costly fetching of the I/O data from system RAM. As a result, if we have enough locality of data to exploit, DDIO reduces the overall I/O processing latency, allows processing of the I/O to be performed entirely in cache, and prevents the available memory bandwidth from becoming a performance bottleneck. We fully investigate the effects of DDIO in Section 2.8 to conclude the impact of the Copy Out mechanism.

2.5 Inlining

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 in 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 [30]. All of our experiments have inlining disabled. Enabling inlining gives almost identical throughput, and overhead, except it only works for transmissions of 912 B or less.

2.6 Eliminating Contention

In our original implementation, we used a common shared buffer pool for transmission. We later found that the use of thread local transmit buffers to avoid contention on the buffer pool during Copy Out transmission shows considerably less overhead due to busy waiting to get free transmit buffers and accentuates the effect of DDIO. This differs from the published results [35]. The measurements in this thesis only include data collected after the optimization.

2.7 Evaluation

We ran numerous experiments profiling a modern NIC to expose its performance characteristics. We wanted to show the impact of NIC features on large databases, which often have to transmit significant amounts of data while responding to range scans or reconfiguring clusters. We set up our experiments that mimic the data transmitting module of an in-memory database. We evaluated various choices in the design space that a database designer encounters while developing a database that is NIC friendly. We set up the following experiments and found compelling evidence that will aid in co-designing databases for new hardware.

- Comparison of various RDMA primitives over Mellanox Infiniband verbs library; we find that `SEND` is best suited for returning scattered records.
- Comparison of different arrangements of small 128 B records to evaluate how to best exploit S/G DMA; we find Zero Copy, *when carefully tuned,* offers better throughput and efficiency. We outline precisely when Copy Out can be faster and more efficient than Zero Copy, which could be exploited for small gains.
- Similar comparison of larger 1024 B records; we observe Zero Copy is a clear winner in terms of throughput and efficiency across the board.
- Comparison of CPU cycles spent per transmitted byte; overheads at transmissions $<$384 B make Copy Out more efficient while Zero Copy is a clear winner for larger transmissions and larger records.
- A breakdown of CPU overhead in transmission; we see that the additional `memcpy` dominates CPU overheads to the tune of 95 %, transmission overhead for smaller transmissions are conspicuous.

2.7.1 Experiment Setup

We explored how the different designs trade-off database server efficiency and performance by building 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 [60] cluster of the CloudLab [65] testbed: this testbed provides exclusive bare-metal access to a large number of machines with RDMA-capable Infiniband NICs. Table 2.1 shows the Experiment Setup for the cluster where we profiled the Mellanox Infiniband ConnectX-3 ®NIC and impact of data layout and use of no update in place structures on transmission throughput. 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. All of our experiments are publicly available online[1]. Figure 2.2 shows the experiment setup. Each of the server threads transmits to one of the 15 clients from our 16 core machine. Thread affinity gives better performance and accurate score boarding of performance.

All our experiments transmit from a large region of memory backed by 4 KB pages that contain 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. Others have argued that in some cases, using 1 GB hugepages reduces translation lookaside buffer (TLB) misses [59]. We have realized that the NIC can benefit from hugepages as well since large page tables can result in additional DMA operations to host memory during address translation [19, 30]. For our experiments, the reach of the NIC's virtual-to-physical mapping is sufficient, and hugepages have no impact on the results. To err on the side of caution, we did enable 1 GB hugepages and registered those with the NIC to make sure we avoid as many TLB misses as possible.

2.7.2 Comparing RDMA Modes for Transmission

We started out by getting measurements of the transmission speeds using RDMA verbs. READ and WRITE are the most commonly employed RDMA verbs because they are one-sided and do not require both ends to be active in theory. RDMA read has the

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

additional incentive that it is client initiated. SEND/RECEIVE primitives, on the other hand, are two-sided and server initiated. We evaluated the performance of each of these using Infiniband verbs library. We implemented RDMA reads and writes as op codes on the `ibv_send_wr` request that transmit data via `ibv_post_send` and measured the transmit performance while enabling Zero Copy using `IBV_WR_SEND` as the opcode. `SEND` operations can transmit multiple entries in the S/G list. One issue we uncovered was that one-sided RDMA verbs are not well positioned to take advantage of the Zero Copy paradigm since it only supports a remote read from a registered location from the other side. READ also requires a secondary notification mechanism to work in practice negating the advantage of being one-sided.

Others have argued for the use of one-sided RDMA for key-value stores [47] and even for accelerating range queries [49]. We estimate that these would be suitable for large transmissions only when there is a strong correlation of queries with resulting values that are contiguous in memory. In the general case, the receiver needs to be active and you can only gather one remote location in one call to the `ibv_post_send`. It could work better if we were to pipeline multiple requests, but we are losing out on efficiency per operation.

Figure 2.3 shows a comparison of transmission performance of different verbs transmitting 200 Bytes(B) records varying the number of records that could be transmitted at a time for various operations. The green dot shows the performance for RDMA READ, which is also unfair since RDMA READ can only transmit from a single remote virtual address at a time.

The blue dot in Figure 2.3 shows RDMA write performance using a full scatter/gather list. This tells us that WRITE and SEND will offer similar performance. Transmission using larger data sizes such as 1000 B records showed a clearer benefit of using `SEND` than using 200 B records, as shown in Figure 2.4.

These figures that compare the performance of RDMA verbs implemented on top of ibverbs library draw interesting conclusions about use of RDMA for transmitting small and scattered records in memory. It is interesting to note that the library could provide RDMA READ and could leverage scatter gather entries retaining all the benefits of a one-sided operation in theory.

Takeaways: Verbs

1. `SEND` saturates NIC; can gather records and notify receiver when transmission completes.
2. `WRITE` avoids invoking receiver; server must induce notifications some other way.
3. `READ` is client initiated and avoids the server CPU in critical path; it's the lowest overhead on server, but can collect one contigous chunk per posted op, limiting its usefulness for returning complex responses.

Recommendation: Use `SEND` primitives for returning scattered set of records.

2.7.3 Record Sizes

In the world of in-memory databases, there is an argument to be made that record sizes will get smaller. The decreased access latency of in-memory databases makes them well suited to smaller, fine-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 hundred bytes or less are now typical even for companies that manage large volumes of data in memory [12, 51]. We choose 128 B records as a realistic sample of record sizes in the future and compare it against 1024 B records to see the performance impact due to bounded-disorder in our evaluations in the rest of the thesis.

2.7.4 Performance Impact of Data Layout

To explore how different design decisions affect trade-offs in 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.

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 many 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 set as 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* 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. We vary *n* from 1 to as much as the NIC can support DMAing in a single transmission. Intuitively, larger groups of records (larger sends) result in less host-to-NIC interaction, which reduces host load and can increase throughput; the actual benefits depend on the specific configuration and are explored below.

Figure 2.5 shows how each of these configurations impacts transmission throughput. For larger 1024 B records, using the NIC's DMA engine for Zero Copy shows clear benefits even when you are transmitting a smaller number of records. This is in addition to the CPU and memory bandwidth savings, which we explore in Section 2.8.1. The database server can come close to saturating the network with Zero Copy as long as it can post two or more records per transmit operation to the NIC (that is, if it sends 2 KB or larger messages at a time). For the Copy Out approach, you get roughly the same transmission throughput as Zero Copy when your transmission sizes get bigger at around 16 KB. In the impact study in Section 2.8, we explain why this is not a good value proposition. We see that the DMA engine could provide a throughput boost of up to 82% over Copy Out while transmitting a single 1024 B record. For small records, the biggest improvement of 67% comes while transmitting six records. It is also interesting to note that if range scans return even just 16 scattered, small(128 B) records per query, the benefits of Zero Copy purely from the perspective of transmission throughput are almost eliminated.

Figure 2.5 shows that for small 128 B records, Zero Copy provides little throughput benefit if you were only returning a few records at a time. Our NIC is limited to gather lists of 32 entries, which is insufficient to saturate the network with such a small record size. Transmission peaks at 3.8 GB/s. In fact, Zero Copy performs best when we transmit between 8-15 records or 1 to 2 KB per transmission, above which the transmit performance tapers off. Zero Copy gives around 55% improvement over Copy Out while dealing with transmissions around 1 KB or eight records. Throughput dips with sixteen records per send operation and it gradually moves up as we increase the transmission sizes further but never gets back to the peak transmission. This calls for careful attention to the length of S/G list in order for a database designer to get peak performance while using small record sizes and larger transmissions. We dig deeper into this anomaly and find evidence as to why this happens with the help of measuring traffic induced in the Memory controller due to LLC misses because of DDIO and PCIe traffic in Section 2.8. Owing to this aberration, copying 128 B records on-the-fly can significantly outperform Zero Copy transmission when there are enough results (more than sixteen records in our NIC, which is capable of transmitting thirty two at once) in a response. In fact, Copy Out can saturate the network with small records, and it performs identical to Zero Copy with larger 1024 B records.

Takeaways: Throughput

1. Bigger records can saturate the NIC; Zero Copy is significantly faster for bigger chunks.
2. Zero Copy is limited by the length of S/G list; Zero Copy can't saturate the NIC transmitting 128 B records.
3. Copy out can be faster for smaller records unless S/G length *is carefully tuned*; we examine Copy Out's efficiency in Section 2.8.1.

Recommendation: For large sets of small, scattered records, use Copy Out. If your record sizes are bigger, always use Zero Copy.

2.8 System Impact

We saw how the Zero Copy promises enhanced savings and performance in the previous section. To put things in the context of resource utilization and to quantify performance gains, we profiled our benchmark code for CPU utilization, memory bandwidth, and some other metrics such as memory bandwidth utilization due to Last Level Cache (LLC) misses from DDIO and PCIe. We used cycle counters present in the original RAMCloud [55] code base for nonintrusive profiling of CPU time. We wanted to make sure our profiling code should not interfere with the timing of our highly performant benchmark, which churns out millions of operations per second and microsecond latencies. We searched for lightweight profiling methods that add minimal overheads. We ended up using Intel®'s Performance Counter Monitoring module [3]. An interesting thing about measuring the performance of a modern CPU is that almost all of the metrics of significance exist outside of the cores. These measurements are classified as uncore and offcore measurements in the Performance Monitoring Unit (PMU) moniker. The newer generation CPUs that we used had the Sandy Bridge microarchitecture that provides uncore performance monitoring unit and we measured their performance counters to gauge impact accurately.

2.8.1 Breakdown of CPU Costs

While we just showed in Section 2.7.4 that judicious use of Zero Copy results in enhanced transmission performance, the goal of zero-copy DMA is also to mitigate the server-side CPU cost. Figure 2.6 breaks down CPU time for all the scenarios: small and large records using Zero Copy and Copy Out. In the case of Zero Copy, most of the server CPU time is spent idling waiting for the NIC to complete transmissions at smaller record sizes.

For 128 B records, smaller transmissions take up a larger fraction of CPU owing to the overhead of more descriptors per transmission. Zero Copy always reduces the CPU load on the server in all record sizes except for the cases where overheads dominate for a small number of small records and, as expected, there is a bigger benefit for larger record sizes. With 1024 B records, the `memcpy` step of Copy Out uses a maximum of 19% of all available CPU cycles. What's alarming is the fact that this constitutes 98% of all the time spent busy waiting. This is a pure overhead that we could eliminate by using Zero Copy.

While Zero Copy eliminates the `memcpy` overhead, it adds an overhead of its own to create and copy 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 user level NIC driver. Next, the driver makes a posted PCIe write by copying the descriptor (including the gather entry) into a memory-mapped device buffer. We will see later in Section 2.8.4 that memory bandwidth savings for Zero Copy are more substantial. Figure 2.5 shows that Copy Out transmit performance nearly matches Zero Copy (5.6 GB/s versus 5.8 GB/s) for larger records at larger transmissions. Copy Out introduces exactly one extra copy of the data, and `memcpy` reads each cache line once and writes it once. So, Copy Out is expected to increase memory bandwidth consumption by $2\times$ the transmit rate of the NIC or 11.2 GB/s in the worst case. This accounts for about 45% of the available memory bandwidth for the server 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 could end up using another 6 GB/s of memory bandwidth. With these estimates in mind, we extensively profiled our benchmark with Intel's PMU module and the details of that are given in Section 2.8.4.

Takeaways: Breakdown of absolute CPU costs

1. `memcpy` dominates the CPU overheads in transmission; Zero Copy offers 8x savings for small records and 45x savings for larger records.
2. Considerable CPU overhead means Copy Out is better at transmitting <384 B of data; For smaller records, overhead of adding and copying descriptors can't be ignored.

Recommendation: If CPU overhead is the likely concern, use Zero Copy; adding an exception for this rule for transmissions <384 B might help minimising overheads.

2.8.2 CPU Efficiency of Transmission

The results from the previous section break down where the CPU savings come from, but not all configurations result in the same transmit performance. For example, Figure 2.5 shows that when transmitting 128 B records, Copy Out gets up to 15% better throughput than Zero Copy while transmitting 32 records at once. As a result, minimizing CPU overhead can come at the expense of transmit performance. CPU overheads as a total fraction of CPU tell us what makes up for the overheads in transmission where CPU cost per transmitted byte will tell us the CPU efficiency of the transmission. The real CPU efficiency of the server in transmission is shown in Figure 2.7. The figure shows how many cycles of work the CPU needs to do for every byte transmitted 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 95% for larger records and around 72% for smaller records. If you were to transmit at least two small records, Zero Copy becomes more CPU efficient.

Takeaways: CPU Efficiency of Transmission

1. Most efficient CPU utilization for transmission is obtained while using Zero Copy on large batches; 72% savings for 128 B records and 95% savings for 1024 B records.
2. Zero Copy is 2x costlier than Copy Out in transmitting a single 128B record; results like equality searches on small records might be better served by Copy Out.
3. If the results to be transmitted involve 2 or more records, Zero Copy is always more CPU efficient.

Recommendation: Zero Copy effectively offloads CPU and cost per byte goes down drastically when you transmit larger batches of records.

2.8.3 Memory Bandwidth

In a modern In-Memory Database server, DRAM utilization is the most precious resource [62]. Available memory bandwidth is also at a premium since there is a direct correlation between memory bandwidth utilization and DRAM latencies and starvation. Since uncore events will give the most accurate measurements, we explored all uncore events that induce pressure on the memory controller. We found that `LLC_MISSES X 64` (cache line size) is an interesting metric, but there was a problem that it wouldn't account for prefetch misses. We went for the more accurate `MEMORY_BW_READS` and `MEMORY_BW_WRITES` metrics that are derived from `CAS_COUNT.RD` and `CAS_COUNT.WR` metrics, which are the number of cache lines read and written in the sampling interval. We used an interactive reporting tool written on top of perf with named mappings for these events to measure this metric. We profiled the total memory bandwidth consumed by the system while our benchmark was running with fixed record sizes, a number of records, and modes of copying (Zero Copy and Copy Out). We took the median of our measurements to account for consumption only while the system was transmitting data.

Figure 2.8 shows the memory bandwidth consumed by the system while transmitting data at various modes of copying varying record sizes and transmission sizes. We can see that using Zero Copy on larger transmissions involving larger record sizes is beneficial. The savings are highlighted along largest transmissions of 32 records of sizes 1 KB each. We had measured that a userspace application could achieve a peak bandwidth of around 24 GB/s in our system. This would mean that Copy out might take up a half of all available bandwidth in order to saturate the NIC while transmitting. This should raise alarms for a database designer since this is pure overhead and the wasted memory bandwidth could be used for data manipulation or other useful work in the database server. This is even more interesting considering that the modern-day in-memory stores are not tuned for large responses. When the DRAM latency and accesses go up, the performance of the whole system could degrade to the point at which the system might become unusable. For smaller data sizes, it is even more interesting. Zero Copy shows less memory utilization until we get to the point where its transmit performance drops below that of Copy Out. This would imply that there are times when Copy Out outperforms Zero Copy in terms of transmit performance and memory bandwidth utilization. It is interesting to note that these data points overlap the anomalous transmit performance degradation we saw while observing transmission throughput in Figure 2.5. Interestingly, after this region in the graph where you transmit 16 or more 128 B records where Zero Copy offers better transmission performance, the memory bandwidth consumption by Zero Copy is reduced, and Copy Out starts to appear more expensive in terms of memory bandwidth. These anomalies are better explained in Section 2.9, but we can conclude from this figure that there is a linear correlation between transmit performance and consumed memory bandwidth for Zero Copy.

Takeaways: Impact of Memory Bandwidth

1. Copy Out might occupy significant portion of available memory bandwidth; Transmissions of 32 records of 1 KB size consumed **half of all available bandwidth**.
2. Smaller records have a smaller memory footprint and may not affect available bandwidth; The trend follows transmission throughput and there are areas where Zero Copy is costlier.

Recommendation: Zero Copying larger records saves the most memory bandwidth while giving stellar throughput.

2.8.4 Memory Bandwidth and Transmission Throughput

Figure 2.9 shows the ratio of Memory pressure exerted per bytes transmitted. We can clearly see the value proposition of Zero Copy mode of transmitting here. For Zero Copy, we can see that the system is consistent and predictable and only incurs a small overhead of copying descriptors other than DMAing the data onto the NIC. The cost of the descriptors is evident from the fact that 128 B records show a bigger multiplier on transmission throughput than 1024 B records. For 128 B records, the ratio appears tumultuous across varying number of records, but still shows Zero Copy as the clear winner. It is interesting to note that transmitting a single record via Copy Out, introducing an unnecessary copy, makes your transmission 3x less efficient in terms of memory bandwidth utilization. If we were to compare larger 1 KB records across the two modes of copying, we can see that there is a 2X memory bandwidth cost on copying out 1024 B records. Like in the case of smaller records, the additional copy reduces the transmission efficiency more in the case of a small number of records.

Takeaways: Efficiency of Memory Bandwidth Utilization

1. Any use of Copy Out is pure overhead; smaller records doing Zero Copy have larger memory footprint owing to the descriptor overhead.
2. There are cases where Zero Copy and Copy Out show similar memory bandwidth efficiency of transmission; Copy Out of smaller records have bigger footprint though inconsistent across transmission sizes.
3. Chapter 3 discusses a hybrid approach yielding better throughput without consuming as much memory bandwidth.

Recommendation: If efficient use of memory bandwidth is a concern, always use Zero Copy.

2.8.5 DDIO Traffic

Newer generations of Intel Processor's have a portion of their Last Level Cache dedicated for I/O. This was initially called Direct Cache Access and more recently named DDIO. This changes our perception of memory consumption quite a bit. Network transmission occurs at this part of the Last Level Cache and even the destination for a packet over the network becomes this dedicated portion of LLC. This has a significant impact in the number of LLC misses the system has to take while dealing with network traffic. The key limitation to DDIO is that usually, it is allocated as only 10% of LLC, which is around 2 MB in a modern system. However, as long as a single transmission does not surpass this size, we can expect higher efficiency in transmission due to DDIO. In all of our experiments, we were doing transmission of at most 32 records of 1 KB each, and hence, the effect of DDIO will be visible in our benchmarks. To measure the effect of DDIO, we measured the number of bytes read from memory by LLC due to misses from the DDIO marked region.

Figure 2.10 shows the reads issued to the memory controller from LLC due to misses from the DDIO region. We can clearly see that when we use Copy Out method to transmit data, we do not see much traffic in this area. When we issue a `memcpy` to copy data into the transmit buffer, it is already loaded in the LLC. Unless that data is ejected from LLC, a miss from the DDIO region need not necessarily involve a read from DRAM. On the other hand, if we were employing Zero Copy technique, and we were transmitting randomized chunks of data, DRAM is almost always involved. Randomisation plays an important role in cache invalidation, and we had to tune our benchmark so that it is always transmitting randomized data. We can see that the DDIO misses are following the same trend as transmission throughput we showed in Figure 2.5. Since our data sets are limited in size and do not cause much cache pollution, we see that the memory reads are only due to misses in DDIO region. For transmissions using Zero Copy, the read requests to memory happen because of misses from the DDIO region and account for most of the memory bandwidth when it is less so for Copy Out. Section 2.8.6 explains this in detail.

Takeaways: Intel®Data Direct I/O

1. While doing Zero Copy, almost all of the misses from DDIO region result in reads issued to memory controller.
2. While doing Copy Out, since the data is already in memory, A miss from the DDIO region may not always result in a memory read.

Recommendation: Keeping your transmission sizes under the size of a modern LLC (20MB) can lower your memory bandwidth consumption considerably.

2.8.6 DDIO Savings in Copy Out

The previous sections showed how Copy Out could be costly and degrade system performance because of the consumed memory bandwidth. We also examined how DDIO traffic plays a big role in modern CPU architecture in limiting I/O costs. We are going to examine the interesting side effect of the additional `memcpy`. Figure 2.11 shows what percentage of consumed memory bandwidth results from misses from the DDIO region. It tells us that while we pay the price of `memcpy` to bring data to the LLC, it helps in limiting further DRAM access while transmitting from the region reserved for DDIO. If not for this effect, both Copy Out and Zero Copy might have added memory bandwidth consumption in bringing data from DRAM for transmission.

Takeaways: Copy Out side effects of DDIO

1. The additional `memcpy` step prefetches data to LLC at a cost and improves the number of DDIO misses that result in DRAM accesses.
2. If not for the DDIO effects, we would have seen a bigger drop in throughput and memory efficiency for Copy Out.

Recommendation: DDIO paints a slightly different picture of memory consumption favoring Copy Out; in the bigger scheme of things, total memory bandwidth consumed still makes us prefer Zero Copy.

2.9 Anomaly in Transmission Throughput

While evaluating performance and efficiency of Zero Copy in previous sections, we saw that for 128 B records, the transmission throughput does not linearly progress on increasing the transmission size. If you refer back to Figure 2.5, throughput trends upward until we reach 1024 B (eight records), the transmission throughput remains consistent for about eight to sixteen records, and then the performance tapers off. There is a huge dip in transmission throughput going from fifteen to sixteen records and even though the performance gradually recovers, it does not quite catch up with Copy Out until we get to the maximum number of records possible and even at that point, Copy Out edges Zero Copy performance by a bit. After some investigation, we discovered that LLC bytes read from memory due to misses due to PCIe jumps to the peak measured reads and then saturates and the anomaly corresponds to these data points. It is also interesting to note that this phenomenon is completely specific to the NIC. In Figure 2.6, the cycles spent posting transmissions to the NIC drops suddenly corresponding to this region.

Figure 2.12 shows the read requests issued by LLC because of misses from PCIe traffic. For 128 B records, we can clearly see that after 8 records, this metric is nearly saturated and at 16 records, the number of bytes read peaks causing the dip in transmit performance. The memory reads due to PCIe traffic are saturated at this point and we can see a similar trend for 1024 B records too. Since larger records pack more data per record, the dip in transmit performance is not evident in that case. The other interesting thing to realize is that for Copy Out, the amount of memory traffic induced by PCIe errors follows a similar trend as transmission throughput, which is expected since NIC itself is connected via PCIe interface. Our intuition is that either the prefetcher mispredicts or the maximum possible size of a PCIe transaction (256 B) causes the jump and the overhead is compensated by more and more data transmitted as we get to bigger and bigger transmissions. Since 1024 B records also show a slight dip in transmission performance around 16 records, the DMA overhead is less evident there since we are already operating at a higher throughput owing to large record sizes.

2.9.1 PCIe Induced DRAM Accesses and Throughput

Figure 2.13 shows the ratio of DRAM accesses induced by PCIe traffic to the obtained transmission throughput for various data points. When we transmit 16 small records (2048 B), the PCIe effectiveness goes down and the ratio peaks. This is strictly in line with transmission throughput shown in Figure 2.5 where at 2048 B, we see a sharp drop in transmission throughput from which it slowly recovers as we increase the number of S/G entries per transmission. There is an inverse correlation between the ratio of PCIe-induced DRAM accesses and transmission throughput.

Takeaways: DRAM accesses due to PCIe traffic and anomaly in Throughput

1. The memory utilization because of PCIe peaks as transmission throughput drops for small records doing Zero Copy.

2.10 Conclusions

- For large sets of small records, Copy Out offers better performance; if your record sizes are larger, use Zero Copy.
- Zero Copying larger records will give you good memory bandwidth utilization.
- DDIO compensates for some of the memory bandwidth cost to Copy Out data.
- Careful tuning of parameters is required if you were to use Zero Copy; PCIe-induced memory accesses may explain the anomalous throughput increase for Copy Out.

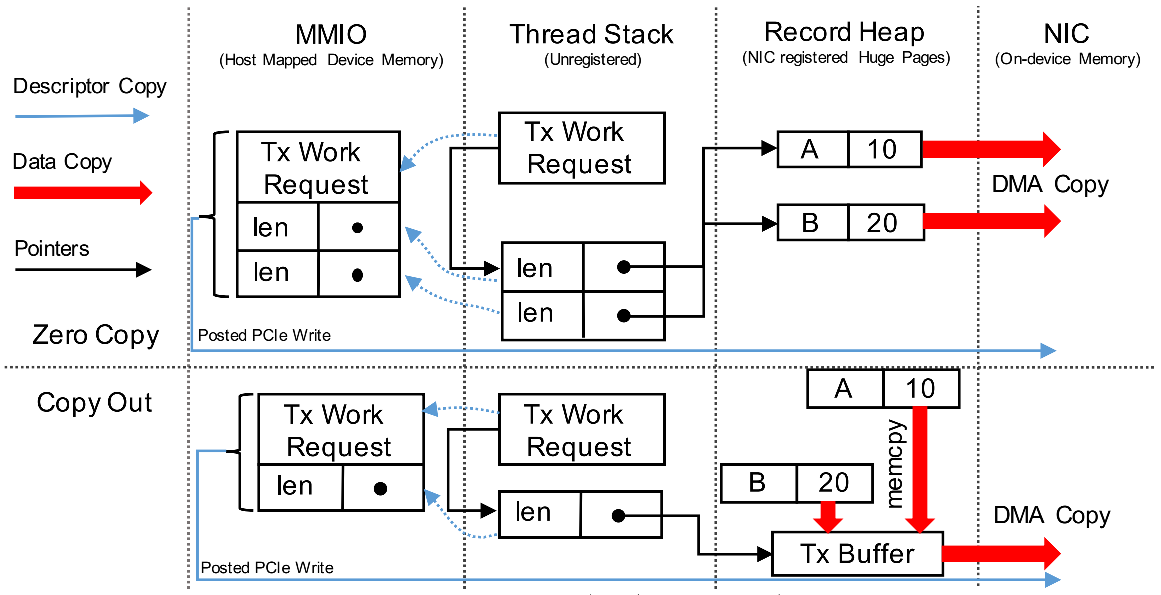


Figure 2.1: Key structures involved in network transmission.

Table 2.1: Experimental cluster configuration.

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

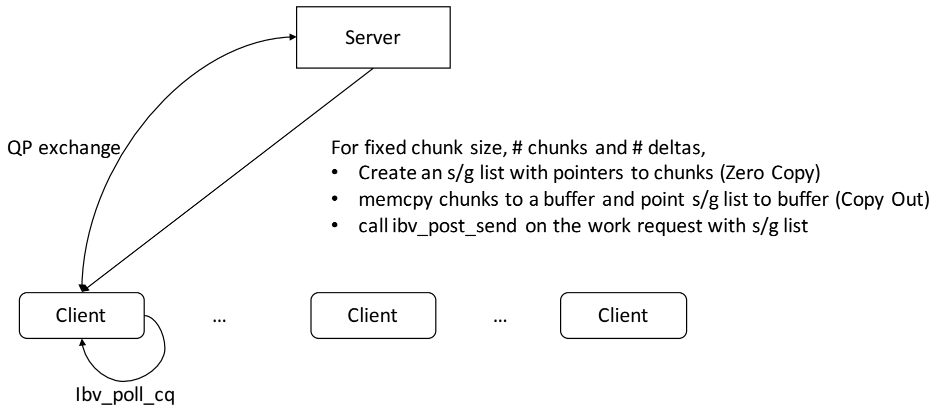

Server
QP exchange
For fixed chunk size, # chunks and # deltas,
• Create an s/g list with pointers to chunks (Zero Copy)
• memcpy chunks to a buffer and point s/g list to buffer (Copy Out)
• call ibv_post_send on the work request with s/g list
Client
...
Client
...
Client
Ibv_poll_cq

Figure 2.2: Experiment setup. Each of the 15 clients communicate to a pinned thread on the server.

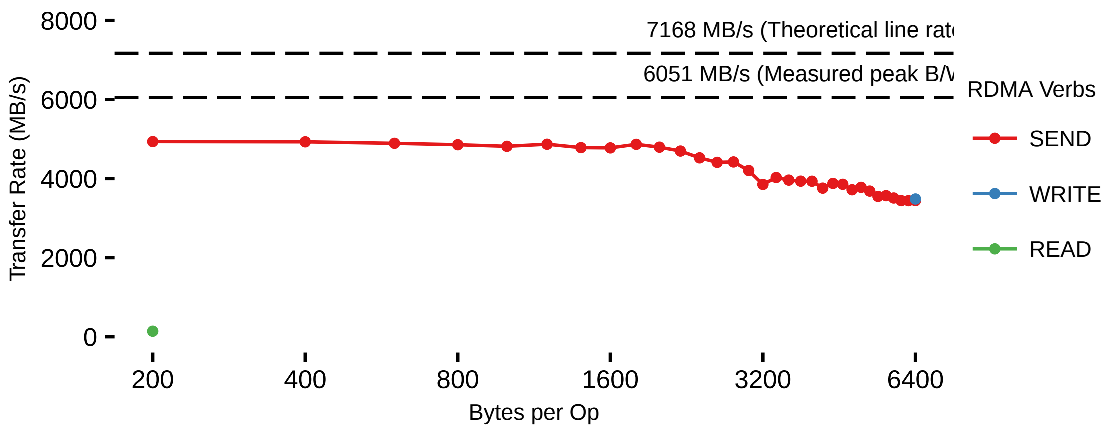


Figure 2.3: Transmission rate for 200 B records over different RDMA verbs and varying S/G lengths. We only show `WRITE` at full capacity and `READ` at one read per transmission.

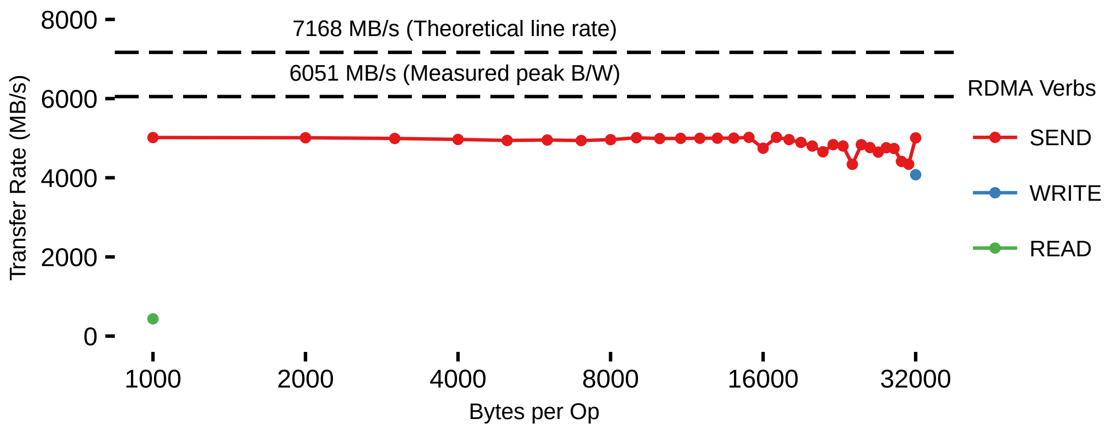


Figure 2.4: Transmission rate for 1000 B records over different RDMA verbs and varying S/G lengths. We only show `WRITE` at full capacity and `READ` at one read per transmission.

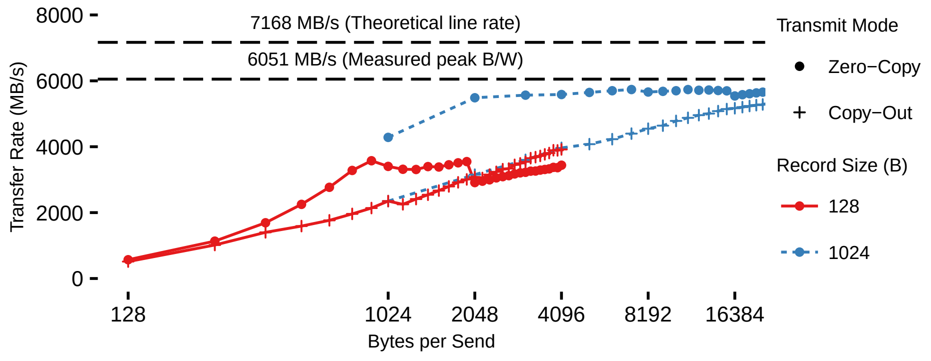


Figure 2.5: Transmission throughput for Zero Copy and Copy Out.

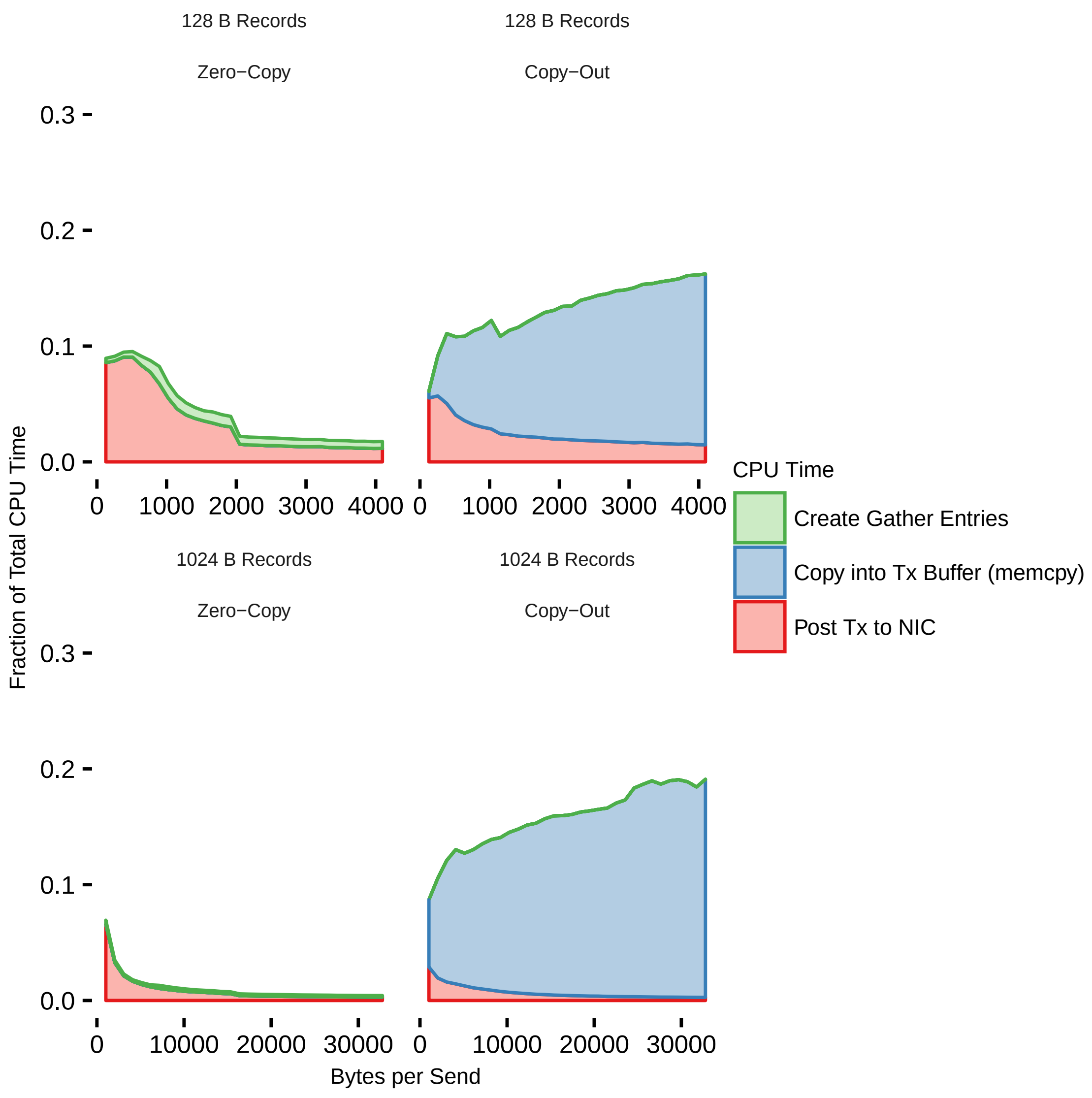


Figure 2.6: Breakdown of absolute CPU overheads for transmission.

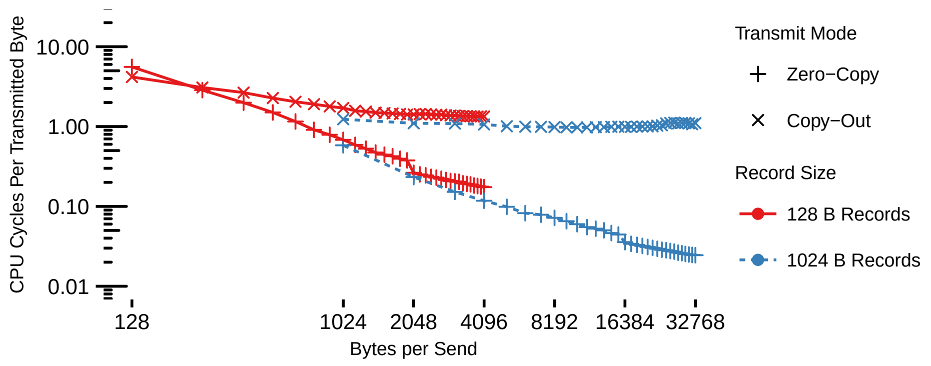


Figure 2.7: Cycles per transmitted byte for large and small records with Zero Copy and Copy Out. Note the log-scale axes.

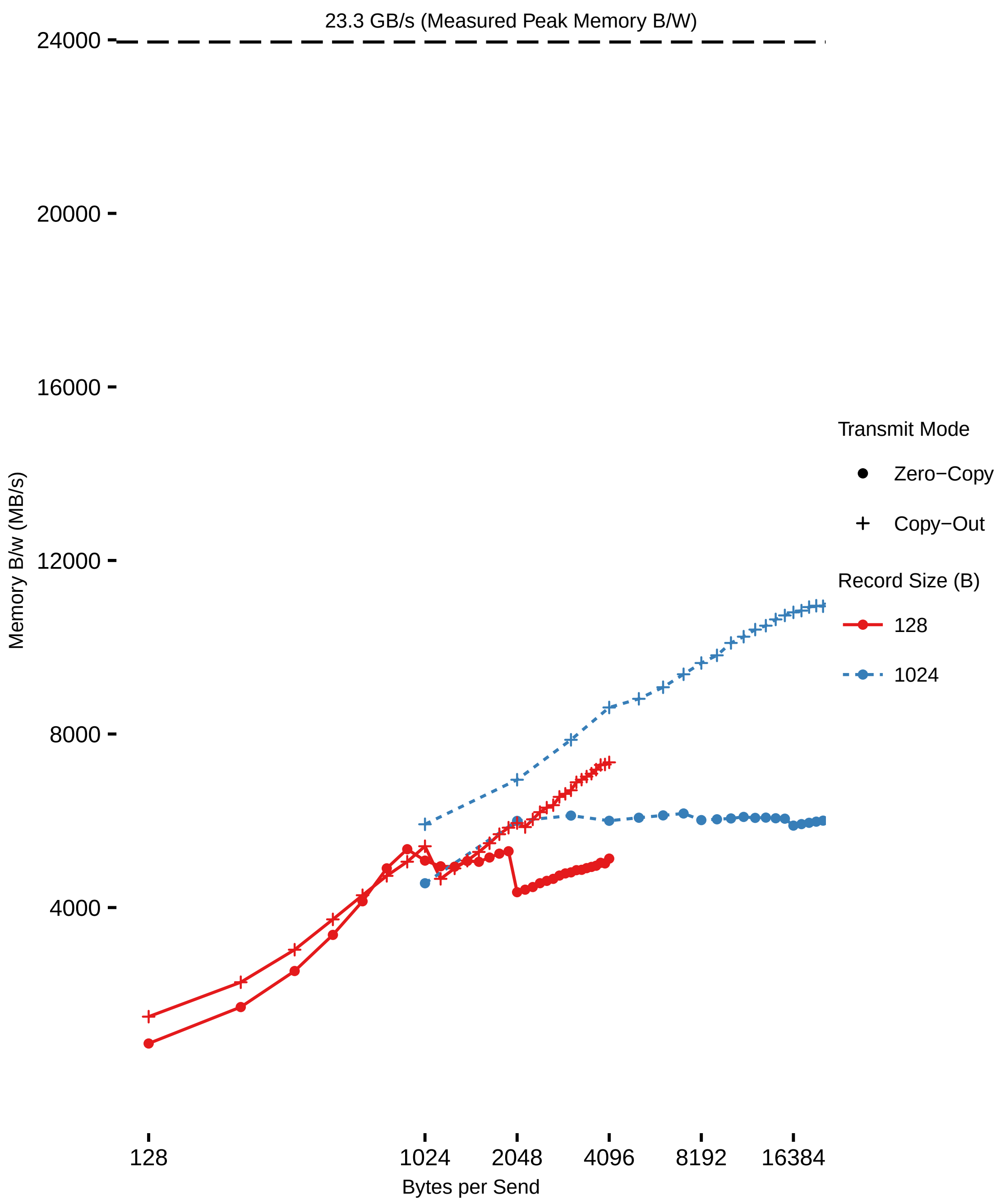


Figure 2.8: Measured memory bandwidth impact on the system during transmission.

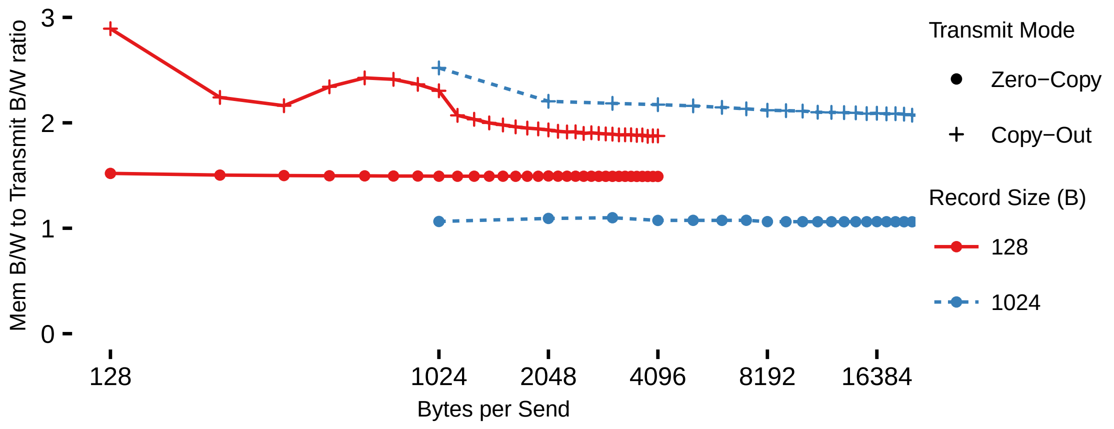


Figure 2.9: Ratio of memory bandwidth to transmission throughput.

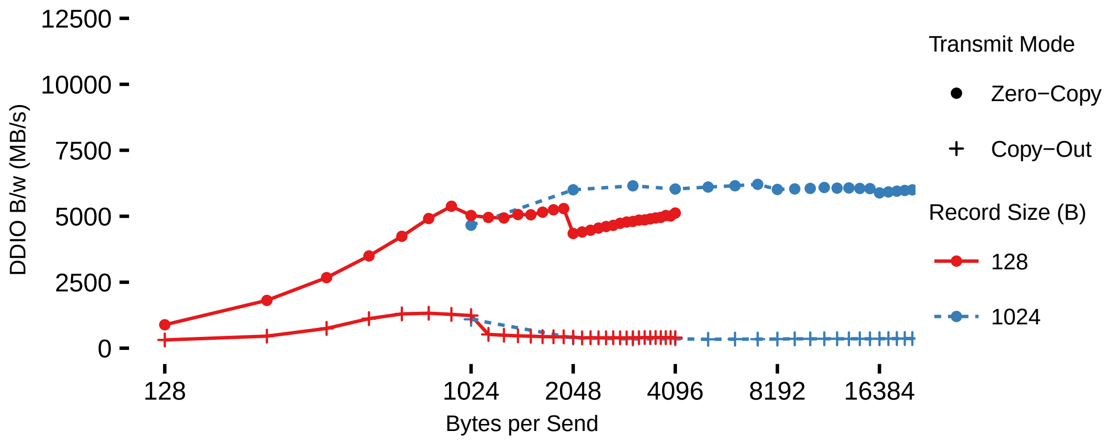


Figure 2.10: DRAM accesses (MB/s) LLC misses from DDIO. vs throughput.

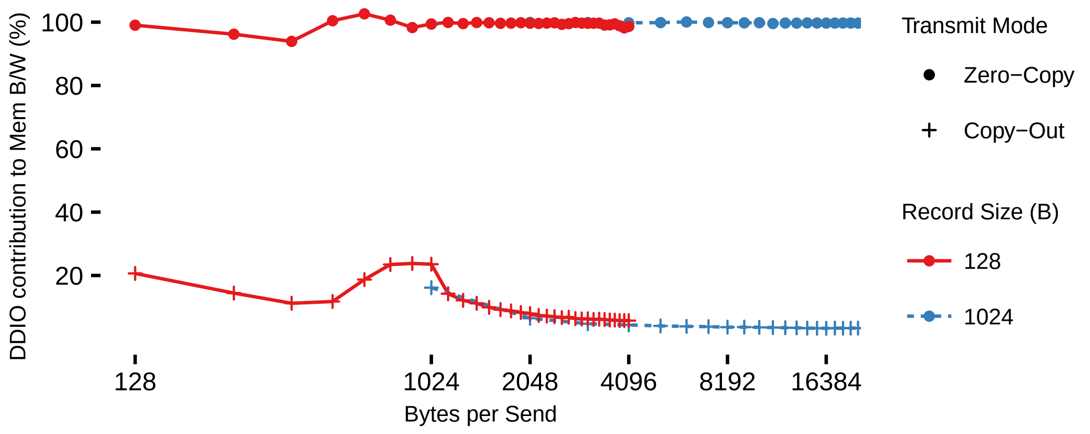


Figure 2.11: Percentage of LLC misses that contribute to total memory bandwidth consumed.

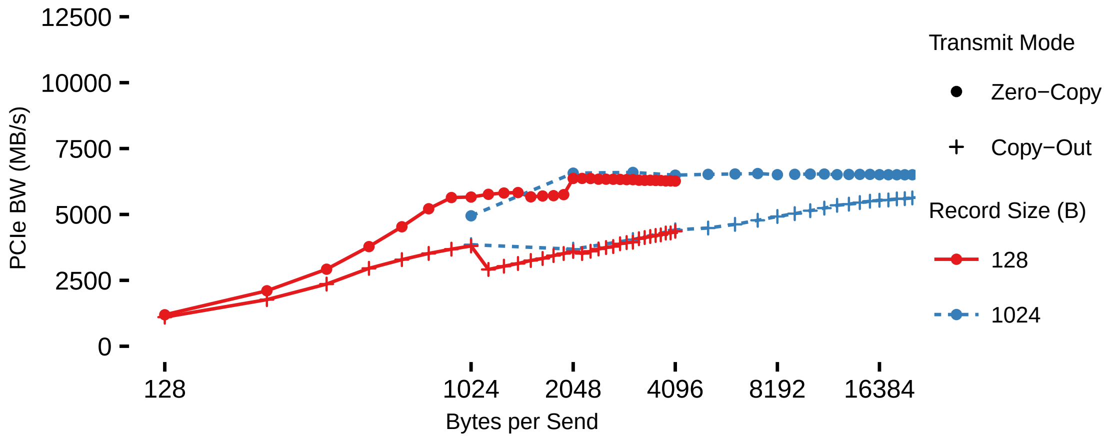


Figure 2.12: DRAM accesses (MB/s) due to LLC misses due to PCIe.

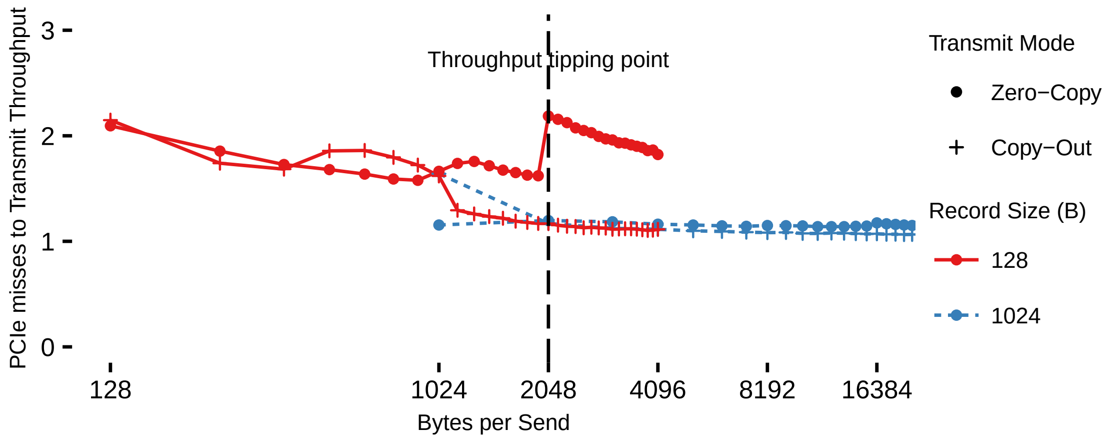


Figure 2.13: Ratio of DRAM accesses because of PCIe to transmission throughput.

CHAPTER 3

APPLICATIONS FOR A CLIENT-ASSISTED DESIGN

We have seen the impact of Zero Copy on network transmission and memory bandwidth and how that compares to using Copy Out to transmit data over the network. The difference in throughput is explicitly seen for smaller 128 B records, where owing to the descriptor overhead and anomalies discussed in Chapter 2, Copy Out gives better throughput performance especially when the transmissions get larger. We have already established that 128 B records show a more realistic depiction of record sizes in the future and that seems to be the growing consensus in the community [12, 51].

At first, the throughput results from comparing Zero Copy and Copy Out for 128 B records might tempt us to think that it is always better to use a Copy Out get maximum throughput with smaller records. The complications of implementation and the cost of overheads are also rampant when using small records with Zero Copy. The bigger problem with Copy Out might get overlooked in the face of all these issues surrounding Zero Copy. While transmitting at around 5.9 GB/s (near line rate for the Connect-X3®NIC), we saw that the memory pressure on the system is close to 12 GB/s, which is half of what we evaluated our server's total available memory bandwidth. This implies that network transmission will bottleneck available memory bandwidth for doing useful work such as processing data in a hybrid or analytical workload.

On-Line Analytical Processing (OLAP) and hybrid workloads composed of transactional and analytical queries have fuelled a renewed interest in column stores [9, 63] in recent years. Column stores are uniquely positioned to take advantage of the NIC because of their layout and compression benefits. The impact of modern NIC while designing column stores should also be discussed for completeness.

This chapter introduces a client-assisted design with bounded-disorder for using Zero Copy with no-update-in-place row stores as well as column stores. We evaluate the improvement in throughput and efficiency to provide evidence to back this approach. We also make informed opinions on how Zero Copy might perform on column store designs for in-memory databases with co-operation from clients for materializing data.

3.1 Bw-Trees

Chapter 2 showed us that Copy Out might offer better transmission throughput for a large set of scattered small records. It also showed the perils of using Copy Out where it consumes a lot of memory bandwidth, starving the server of memory bandwidth that could be utilized for doing useful work. Zero Copy promises memory bandwidth savings in addition to eliminating CPU overhead, while Copy Out can saturate the NIC throughput. We wanted to extract the best of both worlds, and we wanted to analyze structures that can uniquely take advantage of the Zero Copy paradigm. This would imply an arrangement of small records where the total transmission size goes up, helping the transmission throughput while being able to do Zero Copy to help reduce memory footprint. For this to work, we would also need a way of efficient update and synchronization so that data being fetched for transmission remain stable throughout the transmission.

Microsoft®Research came up with a modern B-tree structure called the Bw-Tree [39] which is their primary indexing structure for SQL Server Hekaton [18]. Bw-Tree is a highly performant, ordered index employing lock and latch free concurrency and effective utilization of caches in modern multicore processors. In the following sections, we explore how a client-assisted design involving a no-update-in-place structure such as Bw-Trees can take advantage of Zero Copy and gain maximum throughput without much system impact. We discuss how its lock freedom and delta records make it particularly suitable to exploit Zero Copy in the following sections. We then present our evaluation of transmissions involving small records tacked on to bigger base records to show the gains in throughput without consuming more memory bandwidth.

3.2 Lock-free Indexing

The Bw-Tree [39] 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 searches and range scans. It is fully lock-free and nonblocking 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 the in-memory SQL Server Hekaton [18] 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 lazy consolidation of 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 3.1 shows how this works. In place updates are avoided 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. 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.

3.3 Delta Records

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 are 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.

3.4 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 hold that view consistent without stalling 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-preemptible 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 our evaluation 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 Zero Copy?
- 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?

3.5 Evaluation

3.5.1 Extending the Delta Format to Clients

The experiments in Chapter 2 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 NIC's DMA engine 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, avoiding the ceiling of transmission throughput. 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.

3.5.2 Throughput and Efficiency of Delta Records

Figure 3.2 shows the benefits of delta records. In our experiment, each transmission consists of a single 16 KB base page while the number (one to thirty one) and size (128 B and 1024 B) 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 0.3% when there are a few delta records up to about 0.6% when there are more. This offers a 30× reduction in CPU overhead in comparison with our evaluation of the Copy Out approach in Section 2.8.1. Compared to Zero Copy of scattered small records, this approach also yields a 1.6× throughput advantage; Figure 2.5 shows that throughput peaks around 3.4 GB/s when records are scattered, while the delta format achieves around 5.6 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.

3.5.3 Tuning Page Consolidation

Bw-Tree and many indexing structures are paged to amortize storage access latency, and paging can have similar benefits for network transmission as well. However, the user-level 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 is 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 a 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 performance slightly, which improves efficiency.

For small records, transmitting compacted base pages that have a few deltas gives a total CPU savings of about 10%. For the same CPU cost, a server can perform about 6.5 GB/s of page compaction or about 425,000 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 four would tolerate 1.7 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.

3.5.4 Impact on Memory Bandwidth

Figure 3.3 shows the total consumed memory bandwidth while transmitting records. It is interesting to note that regardless of the variation in delta record sizes, memory bandwidth stays roughly the same. Comparing Copy Out transmissions of similar size, we get a 46% reduction in the consumption of memory bandwidth in the system. Another interesting factor is that the transmission involved only a few consolidated delta records, which we showed in Section 3.5.3 is beneficial; there is no added memory bandwidth pressure if we decide to transmit smaller or larger records. The bigger base page gets us to peak transmission, and a limited number of delta records is favored for aggregate CPU costs. We have arrived at the sweet spot of transmitting large amounts of data in a modern NIC.

3.5.5 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 3.2. Normally, the Bw-Tree copies-out returned records. While an operation and its Copy Out are ongoing, the 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 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` that 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.

3.6 Zero Copy for Column Stores

Column-oriented in-memory databases pose unique challenges in interacting with a modern NIC. Column stores use contiguous memory locations to store adjacent columns. They have also been known to use Run-Length Encoding, Dictionary Encoding, Bit-Vector Encoding, and heavier schemes like Lempel-Ziv Encoding for yielding better compression, which is one of the many value propositions of using a column-oriented layout.

We observed in the previous sections that Zero Copy offers promising performance if we had stable entries and used Zero Copy to transmit the results while responding to a range query. However, the fundamental problem lies in the fact that end-users expect query processing and analytics on data to materialize results in row-order. It has been argued before that early materialization of data helps performance in column-oriented databases [10]. This makes the application of a scatter gather list difficult, and we might have already used a `memcpy` or an alternative to materialize the data.

We argue for the possibility of a different materialization strategy with more engagement from the clients. If the database server were to sent filtered columns and required metadata such as the dictionary encoding as scatter gather entries, we could leverage Zero Copy's performance. We believe this approach would work even for hybrid layouts [26] that use row-oriented or column-oriented formats varying the width of columns according to the type of workload. Figure 3.4 shows how row stores and column stores could transmit data using Zero Copy using a client-assisted design. In practice, there are numerous challenges associated with this design:

- *Client co-operation*; this scheme of transmitting results would need smart clients that are capable of materializing data given the encoding scheme and the columns. We assume that the client has enough resources to spend in finessing the results.
- *Low selectivity in queries*; we have seen packing more data yields more savings and performance from Zero Copy. We also know that the compression schemes in column stores can limit transfer sizes. As a result, queries with low selectivity are a natural fit for Zero Copy.
- *Compressed metadata*; the client-assisted design would need the encoding to be transmitted along with the results; this would need efficient compression and lean data structures for metadata.

There needs to be further evaluation of how we can extract maximum performance and efficiency from a server equipped with a modern NIC.

Takeaways: Client-Assisted Design with Delta Records

1. Use of a large enough base page and updates as delta records helps in saturating NIC's transmission throughput.
2. We get a 50% reduction in memory bandwidth consumption using this approach without compromising transmission throughput.
3. No updates in place and latch and lock freedom makes this approach clearly favorable for the modern NIC.

Recommendation: **Novel data structures can significantly influence the performance of a modern NIC**; To extract maximum performance from the NIC without compromising system resources, make use of client-assisted designs using lock-free structures that offer no updates in place.

3.7 Conclusion

Zero Copy and Copy Out provides different optimizations for scattered, small records. A hybrid approach that leverages the high throughput that we obtain as a result of preassembling scattered records and using Zero Copy to transmit will yield good transmission throughput without consuming a lot of memory bandwidth.

- With the assumption of *co-operative clients* and queries with low selectivity, we can continue to use Zero Copy and get better results while dealing with small, scattered records that are common in in-memory databases.
- Advanced structures such as the Bw-Tree that allow no update in place and complete lock and latch freedom can be exploited for their suitability with the NICs. Further evaluation is required for these to work in practice.
- Column stores could also benefit from the hybrid approach; materializing results in row-order makes Zero Copy more complicated in modern NICs and one possibility is to send the encoding metadata along with the results with client-assisted materialization.

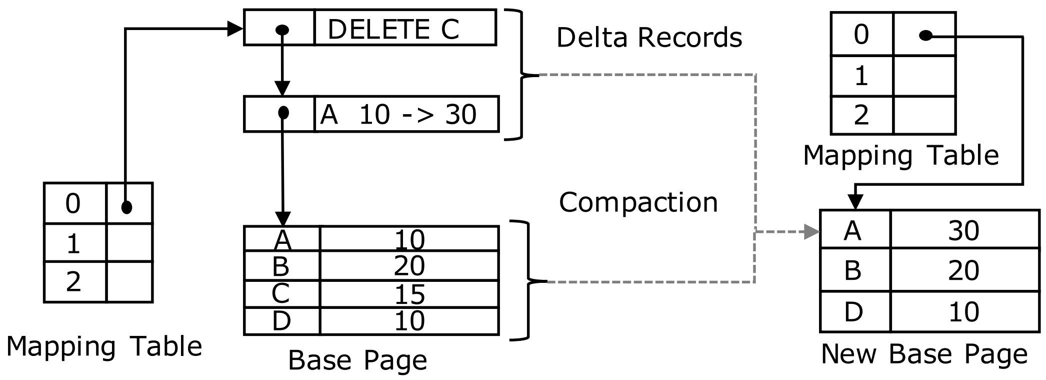


Figure 3.1: The structure of a Bw-Tree.

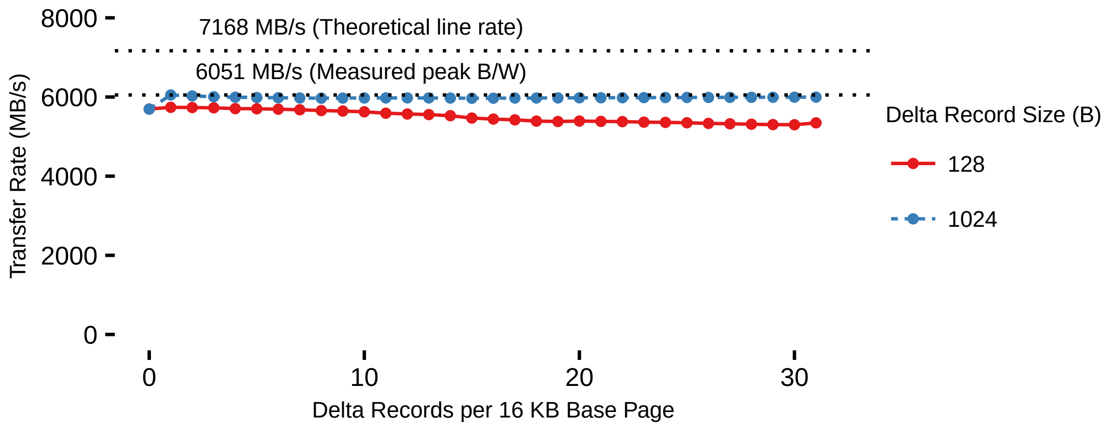


Figure 3.2: Delta throughput

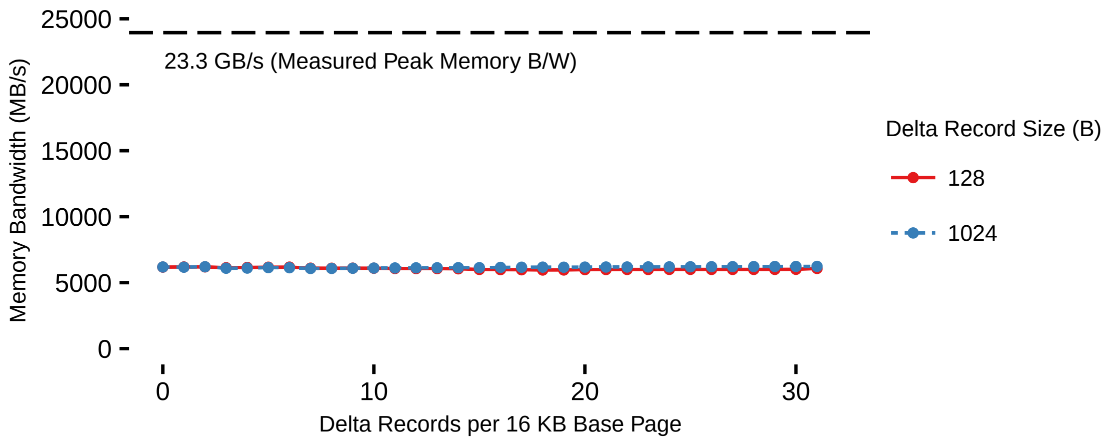
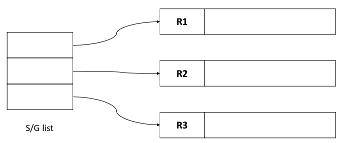
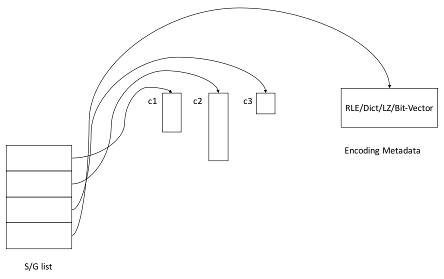


Figure 3.3: Observed memory bandwidth impact varying delta records on a 16 KB base page.





(a) Row-oriented Zero Copy (b) Column-oriented Zero Copy

Figure 3.4: Figure showing a possible design for Zero Copy on hybrid data layouts.

CHAPTER 4

NIC AWARE RECONFIGURATION

We have seen that the current generation of NICs are capable of transferring Gigabytes of data per second in latencies of the order of a few microseconds. We have reasons to believe that the next generation of networking hardware will have the ability to process hundreds of millions of messages per second at bandwidth of the order of tens of Gigabytes per second and sub-microsecond latencies [1]. We saw from the previous chapters how the nuanced performance characteristics of a modern NIC will influence overall transmit performance and server resource utilization for large data transfers such as responses to range scans. Another pressing concern for a modern in-memory database operating at scale is quick reconfiguration. Since DRAM is the most expensive resource in a modern cluster, efficient rebalancing involves fast reconfiguration that is frugal on resources and keeps the impact on their SLAs minimal. Informed with our lessons about the NIC, it is reasonable to assume that protocols for reconfiguration today can benefit from these optimizations and armed with the changes for proposed protocol, data migration might come closer to what the hardware is capable of doing today.

In this chapter, we discuss the state-of-the-art migration protocols and their performance, do an analysis of the existing migration protocol in RAMCloud, and armed with the lessons learned by profiling a modern NIC, we make concrete suggestions on how a new protocol might be well poised to take advantage of the new hardware.

4.1 State-of-the-Art

Fast reconfiguration of in-memory data is something that has been overlooked largely by the research community. There have been novel techniques for live migration of key value stores such as Albatross [17] that provides unavailability windows as short as 300ms, Zephyr [22] that does on-demand pulls, and asynchronous data pushes to minimize failed transactions and unavailability for migration while operating at transactional latencies of 50 ms. Squall [21] went further with fine-grained partitioning with optimizations such as prefetching. Squall also offers minimal overhead to their transactional latencies, which are of the order of tens of milliseconds and migrate data overall at the rate of around 5 MB/s (Refer to Squall paper, Figure 10 a). While these efforts offer reconfiguration speeds better than other systems and minimal addition to their millisecond latencies, that is far from what a modern NIC is capable of delivering. From our benchmark, we knew that the NIC can deliver transmission throughputs of around 5 GB/s while operating at microsecond latencies. There are important lessons to take from these recent research efforts on migration, but without devoting attention to the NIC and its complex performance characteristics, today's cluster of storage servers cannot offer migration performance order of magnitudes better than the state-of-the-art.

4.2 Migration in RAMCloud

We saw from the previous section what the latest reconfiguration protocols are capable of doing and how they can only offer 1000$\times$ less throughput than the latest NICs operating under millisecond latencies. There is a different class of systems [19, 29, 40, 55] that were developed recently that optimizes for end-to-end latency and has the ability to perform millions of operations per second per server. RAMCloud [55] is a prime example of such a system that promises median read latencies of 5µ s and responds to workloads up to a million operations per second. These characteristics result from the fact that these systems were designed from the ground up keeping low latencies in mind and any significant disruption to their operating latencies might deem them less useful. RAMCloud has a migration protocol that offers around 100MB/s data transfer speeds albeit synchronous and unpipelined. While this is faster than other systems, we believe that carefully leveraging the NIC, we can attain better throughput without compromising on latency.

4.2.1 Current Protocol

Figure 4.1 shows the steps involved in the existing protocol. The key drawback to the current approach is that most of the steps involved are synchronous and blocking. The current protocol could make use of Zero Copy since migration usually involves a large amount of data and sets of large data and the presence of `safeVersion` [37] makes Zero Copy an interesting proposition for migration in RAMCloud. While doing migrations, we found that the migration throughput that RAMCloud's existing protocol provides to be around 150 MegaBytes per second (MB/s). While this is at least ten times better than the state-of-the-art, there is definitely room for improvement in the protocol. We profiled the time spent in migration doing each of the steps mentioned in Figure 4.1.

Figure 4.2 shows an analysis of how each of the bottlenecks affects RAMCloud's migration performance. We made versions of migration protocol bypassing each of the bottlenecks and measured RAMCloud's migration performance in each of those scenarios. We migrated a table of a hundred million records with record sizes of 100 Bytes, which comes close to 9 GB including the key and metadata.

The first bottleneck we bypassed was the re-replication of data to the backups. This is a configurable parameter and removing this, we observed a speedup from around 150 MB/s to around 200 MB/s. This explains that while disk bandwidth was indeed a bottleneck, avoiding it does not give significant performance boost while migrating a small data set.

The second bottleneck we tried to avoid was to skip replaying the migrated segments on the receiver. From here on, the safety of the protocol is heavily compromised even under nonfailures. Since the point of the experiment was to show how much each step contributes to overall performance, we discard safety. Skipping replaying segments took us from 200 MB/s to 600 MB/s.

Since we exhausted all steps involved in the receiver during migration, we skipped the actual transmission of data from source to destination and not surprisingly, the throughput only went up from 600 MB/s to 700 MB/s.

The final step is the most interesting with respect to the lessons we have learned by analyzing the NIC. Skipping the copying step before transmission took us from 700 MB/s to 1150 MB/s. This underlines our earlier observation that any extra copy involved in a transmission is pure overhead and significantly affects performance. The analysis tells us that there is clearly room for improvement in migration performance in RAMCloud. While sequential replay of migrated segments of data seems to give the most benefits, it is safe to assume that use of Zero Copy while transmitting data from the source will enhance network transmission throughput without hurting the available memory bandwidth.

4.2.2 Locality in RAMCloud

RAMCloud's take on data locality is unique. RAMCloud promises uniform median latency of around 5μs [55]. RAMCloud does not have a strong notion of collocating data across servers. It is known that transactions [37] and secondary indices [33] can benefit with collocation of data. RAMCloud's data is hash partitioned, and within a server, the in-memory log is organized in a temporal order. We might be tempted to think that storing data in contiguous memory location in some order based on primary or secondary keys or even some other predetermined order will give better NIC performance. The key obstruction to doing this is RAMCloud's log cleaning mechanism [62]. RAMCloud's design philosophy rightly highlights the importance of DRAM utilization when all data is in memory. As a result, the log cleaner is designed in such a way that after cleaning, the log is reorganized so that records that are accessed in the same frequency are placed together; this makes sure that the records in the same segment get garbage collected together freeing up more space than otherwise. To keep this property intact, we cannot force a deterministic order of data in RAMCloud and this might turn out to be a problem if we want to pre-aggregate data to get higher transmission throughput.

We set up an experiment to demonstrate the effect of collocating data in RAMCloud. RAMCloud is slated to provide a throughput of one million operations per second. We set up a cluster of 10 RAMCloud servers and had ten clients issue "multiget" requests to the servers that return values for multiple keys in one operation. The operations were done on a table of 10 keys that were placed in various servers in different configurations.

Figure 4.3 shows the results of this experiment. When we had all 10 keys per table localized in each of the servers, the whole cluster is expected to produce a throughput of 100 million objects read per second. Owing to the overhead of multiget, we observe around 45 million combined operations on the server. The dotted line shows the throughput of the server if we were to use a regular "get" request to extract keys one by one. We see a great benefit in the total throughput when all keys are collocated in a single server for all tables. The x-axis indicates the server spread. When we move along the x-axis, the second point indicates a configuration where for every table, 9 keys belong in one server and another belongs to a different server. The right most point is the configuration where each server hosts one key of all the 10 tables across the server. The bottom half explains what fraction of total CPU burned was spent doing useful work (worker) and what fraction went into handling the communication between servers (dispatch). We can clearly see that the amount of useful work done takes a great hit while the communication overhead dominates even if each server in the cluster had to do one cross-machine extraction of data. Add this to the fact that if all the keys were evenly spread across the cluster, the "multiget" request does more harm than good and getting the keys one by one would have resulted in better aggregate throughput of the cluster. We can conclude that the effect of locality of data is paramount, and any movement of data to achieve better locality of data between servers will yield great performance later.

4.3 Design Guidelines for a New Protocol

Based on the analysis of the bottlenecks in the current protocol and the motivation that comes from having data localized to servers, some guidelines for developing a protocol for migration in RAMCloud are given below that have ties to the lessons we learned from benchmarking a modern NIC.

- Make use of *Zero Copy*; RAMCloud supports a slew of network transport protocols including TCP and Infiniband. Since kernel by-pass is essential in the design philosophy that calls for minimal interrupts, more popular packet processing oriented transports like DPDK [2] are also supported. No matter what protocol is used, we have seen from our previous chapters that using Zero Copy yields superior performance and efficient utilization of resources. The new protocol should minimize the number of copies involved in transmission.
- Layout your data such that there is *at least one large enough chunk* before transmission; Assuming we are doing a live migration while servicing data, it makes sense to reapply the guidelines from our evaluation of delta records in Bw-Trees here. Data could be pre-aggregated into a buffer and updates and some set of scattered records (limited by the network card) may be transmitted, giving near line rate throughput and minimal overhead to consumed memory bandwidth. Refer to Section 3.5.2 for how a 16KB base page can enhance transmission throughput by 1.6 times.
- *Defer re-replication* to the end; Our analysis of experiments only showed a minor increase of performance when we avoided re-replication. Larger migrations would

need higher *disk I/O bandwidth*, which gets bottlenecked at around 300MB/s for regular spinning disks. It would be ideal to defer re-replication to the end or use RDMA to achieve re-replication provided there is some way to ensure the safety of data in the face of failures during migration.

- *Defer partitioning* decisions before migration; We mentioned that ordering data within a RAMCloud server is not feasible because of utilization. Any new migration scheme should continue to adhere to the principle of assuming no locality of data within the server before migration unless significant changes are made to the log cleaning mechanism.
- *Parallelise or pipeline* steps in migration; We saw from our analysis that the sequential nature of the migration is detrimental to its performance. A new protocol should be able to parallelise the transfer of data as well as replaying at the receiver in order to attain maximum performance.

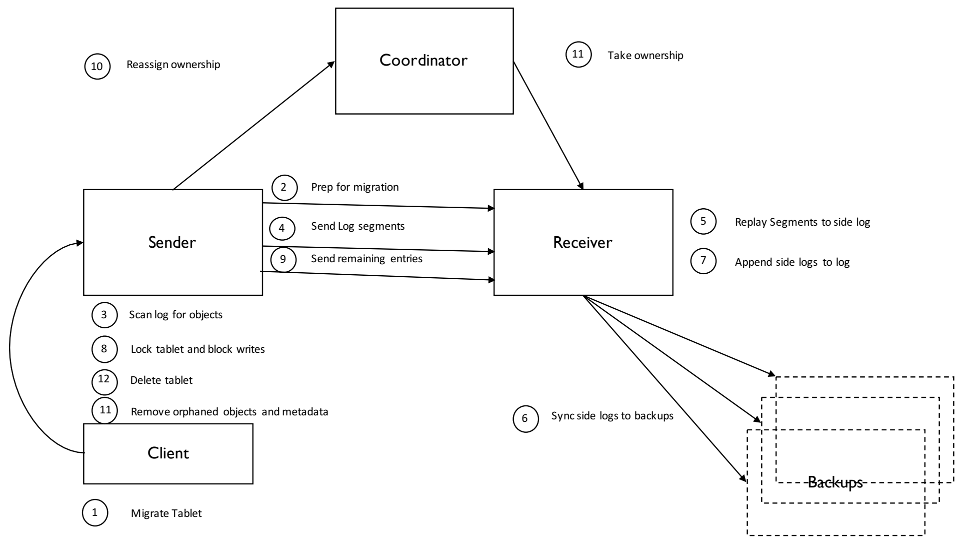


Figure 4.1: Diagram showing steps involved in the current migration protocol of RAMCloud.

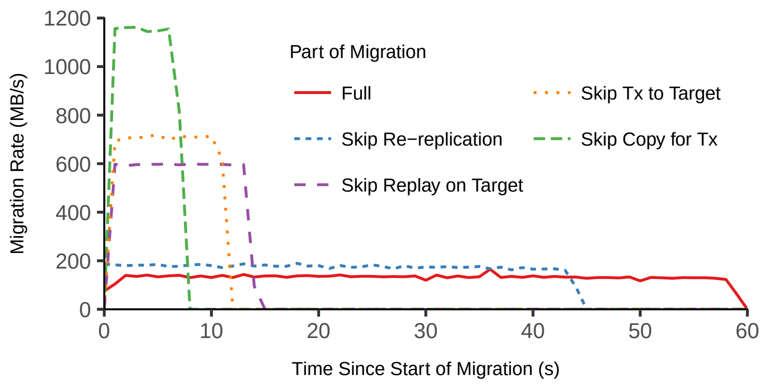


Figure 4.2: Analysis of bottlenecks in RAMCloud's current migration protocol.

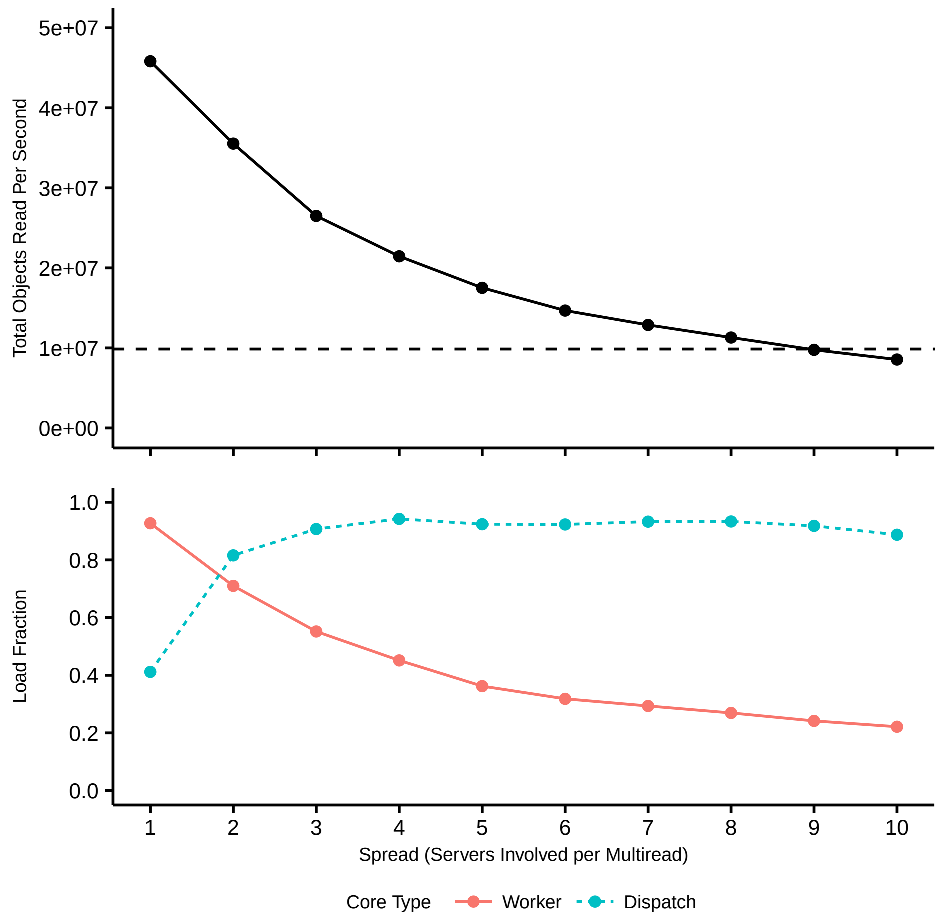


Figure 4.3: Colocation plot. Collective throughput and fraction of useful work and communication overhead over server spread.

CHAPTER 5

RELATED WORK

In-memory databases and low latency networks are very heavily researched areas these days. Our work borrows concepts from numerous efforts both current and prior from the databases, networking, and operating systems research communities. RAMCloud [55] itself was a research product built on top of other work on log-structured file systems and kernel by-pass networking. RAMCloud gave the world a distributed fast crash recovery, scalable log structured DRAM storage, and a new consensus algorithm [52, 53, 62, 64] in addition to many other functionalities that makes it one of the most feature complete low latency storage systems to come out of research.

5.1 In-memory Databases

In-memory databases sparked a new wave of systems research in the networking and database communities. For a long time while developing storage systems, researchers did not diverge from the tried and tested model for the locality of reference. Conventional "secondary storage" devices such as disks were the final resting place of ever-growing amounts of managed data. With the enhanced requirements of scale by the popularization of Web 2.0 in the early 2000s, database designers created a caching layer of RAM in front of disks to combat abysmal disk access times that were dragging down performance at scale. Memcached [23] is one such popular system and its further development enabled it to operate at a scale of billions of operations per second on trillions of items [50].

Another orthogonal area of research was to completely store all data in main memory. This was made possible in the late 2000s by the evolution of DRAM technologies and declining cost of main memory. Questions about durability and cost of replication were responded to by means of distributed recovery [53] and advances in defragmentation techniques [62] ensured high DRAM utilization. Another recent development was in the acceptance of high throughput and low latency networks. Pioneered by the high performance computing community, these advanced networking fabrics such as Infiniband [56] became more commonplace in data center networks, and kernel by-pass networking was made ubiquitous by packet processing frameworks such as DPDK [2], which enabled them to operate over universally accepted protocols such as Ethernet. These developments have led to a deluge of storage systems research [36] exploring multiple verticals. Among the particularly notable ones are RAMCloud [55], FaRM [19], and H-Store [32] (scale-out memory stores) and MICA [43], HyPer [61], and HERD [29] (focused on single node performance). While each of these systems has been driven by different dimensions of research, they all share the same concerns of in-memory storage systems operating on high-speed networks.

Today's main memory database systems offer a plethora of features that are at par with traditional database systems including structured storage and indexing, concurrency control and transactions, durability and recovery techniques, query processing and compilation, support for high availability, and ability to support hybrid transactional and analytics workloads [36]. FaRM [19] is an in-memory, distributed computing platform that leverages lock-free reads over RDMA [30], collocating objects and function shipping to provide performance characteristics of the highest order for an in-memory database. FaRM scales distributed transactions to hundreds of millions of operations with recovery times of the order of milliseconds by relying on full replication in DRAM. There are other systems such as Silo [67] that effectively utilize multicore CPUs.

5.1.1 Column Stores

Column-oriented databases have generated interest in recent years owing to the growing need of running complex analytical queries over a large amount of data. Most enterprise database solutions today ship with the column stores. This new focus has resulted in advanced materialization strategies and compression schemes [7, 9, 10, 63]. In-memory databases with hybrid layouts [58] and other recent work on in-memory column stores [28] focus a lot on analytical workloads and techniques like database cracking [27] for adaptive indexing. MonetDB [8] made significant milestones in OLAP processing of in-memory column stores. HyPer [34] is an example of a system that underlines the importance of RDMA in the context of a column store.

5.1.2 Lock Freedom

Lock freedom is important for Zero Copy and RDMA since the NIC also introduce contention since records are transmitted directly from their original memory location without an additional copy. Lock and latch freedom are design philosophies perfected on previous work on fine-grained concurrency primitives [15] and lock-free hash tables [46]. Lock freedom is commonly used in databases these days [45] since they gained practicality by the common use of epochs-based reclamation [24] . More recent iterations of SQL engines such as Hekaton [18] are examples for in-memory optimized database engines. Bw-Tree [39], a data structure that is lock and latch free, follows a no-update-in-place philosophy to indexing. Others have worked on exploiting Hardware Transactional Memory [42] for in-memory transaction processing [69] as well as lock-free indexing [45].

5.2 RDMA in Databases

The merits of RDMA have been widely accepted by database researchers by now. The ability to offload CPU from transmission make it appealing while the necessity for multiple round trips to fetch data made people hesitant. The ability of Infiniband NICs to use scatter gather to assist remote memory access has also been discussed before [66]. From the early arguments [57] calling for the need for offloading CPUs to research aiming for a billion key value operations on key value servers [40], RDMA seems to be a promising paradigm for the future of large-scale databases. Others have suggested a decoupled server I/O architecture using Programmable I/O (PIO) and re-architecting virtual memory systems for network transfers [16]. Key value stores such as HERD [29] and MICA [43] and systems that provide more complex data models such as RAMCloud [55] use RDMA effectively and there have been published guidelines to maximize performance for dispatch-heavy workloads [30].

5.3 Data Migration

Research efforts focussing on reconfiguration for in-memory databases have seen a lot of interest lately. Albatross [17] offered live reconfiguration with unavailability windows as low as 300ms. Zephyr [22] focuses on mitigating violations to Service Level Agreements

(SLAs) and limits change in average latency to 10-20% but like Albatross, operates on latencies of the order of tens of milliseconds. Squall [21] employs reactive pulls and pre-partitioning of data for fast migrations and is the state-of-the-art for low impact migrations that specifically work well for heavily skewed workloads. Like others, Squall also operates on platforms with throughput and latencies orders of magnitude less than what is possible with today's hardware. The key challenge in implementing migration in a system like RAMCloud is to maintain the latencies that are $1000\times$ tighter than these systems.

5.4 Data Transfer in Fast Networks

As pointed out by various researchers in the past, the days of slow networks are over [13] and network no longer remains the primary bottleneck that database designers should take for granted. This thesis argues heavily in favor of that fact. Several efforts in the past have exploited user-level NIC access as well as RDMA for highly performant distributed database systems [19, 20, 55, 61, 69] as well as block I/O for file systems [41]. The closest analysis that we have found on range queries and in-memory databases [49] argues for use of RDMA reads and multisocket parallelisation of copying of data. Our analysis is more detailed and quantifies the results with memory bandwidth and effects of DDIO. Our findings should also complement other efforts such as transactional key value stores [38] and shared-data databases [44]. Small, fixed-size request/response cycles have been optimized in existing research [19, 29, 30, 43, 55], 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.

A complementary study by Kalia, Kaminsky, and Andersen [30] provides a different 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. While most of RDMA research prefers one-sided RDMA verbs, more recent efforts have shown the benefits of two-sided

RDMA and datagram RPCs [31]. We believe that two-sided RDMA and RPC semantics are the most promising in today's architecture. In the future, richer semantics that allows the gathering of remote data might prove beneficial for one-sided RDMA verbs.

5.5 Summary

Our work is built on the premises of a lot of research ideas and its unique take on heavy transfers in the presence of tight SLAs (latencies of the order of 10s of microseconds) make it stand out among the related work. Until now, mixing workloads such as range scans and data migration, especially in the context of modern networks, has not been well understood from the server side. We present an in-depth analysis of resource utilization, the impact of DDIO and how data layout affects performance and efficiency in a modern in-memory database.

CHAPTER 6

CONCLUSION

Section 1.1 remarked the following thesis statement:

"Scale-out in-memory stores are optimized for small requests under tight SLAs, and bulk data movement, for rebalancing and range queries, interfere; the thesis argues carefully leveraging data layout and advancements in modern NICs will yield gains in performance and efficiency for large transfers in these systems without disrupting their primary obligations."

We have found conclusive evidence for this statement by evaluating the transmit performance of a NIC and the resource utilization of a server while transferring large amounts of data. We made numerous recommendations and observations from benchmarking the NIC. We presented a client-assisted approach that will get better throughput without compromising resources. We analyzed the effect of collocating data in RAMCloud and benchmarked the key bottlenecks in the current migration protocol. We also proposed a set of design guidelines for a new migration protocol for RAMCloud, drawing lessons from observing the NIC closely while it was transferring gigabytes of data.

While evaluating the various RDMA primitives, we found that one-sided RDMA verbs, though they offload destination CPU, require additional acknowledgment mechanisms. We also found that client-initiated `READ`s cannot use the scatter gather DMA engine effectively. We arrived at the conclusion that `SEND` primitives are the most suited for returning scattered set of records.

We analyzed the transmission throughput of Zero Copy and Copy Out for 128 B records and found that bigger records can saturate the NIC. Zero Copy is significantly faster for bigger chunks. Zero Copy is also limited by the length of S/G list for small 128 B records.

We also found that Copy Out can perform better if you have more records. We arrived at the following conclusion that unless your records are small and scattered, always use Zero Copy; for small and scattered records, you might need to tune your layout to get Zero Copy advantage.

Zero Copy with two-sided RDMA promises CPU savings by offloading CPU from the critical data transfer path of the sender. We found that CPU overheads in transmissions are largely due to the additional copies involved in Copy Out and Zero Copy benefits from avoiding the extra copy.

CPU efficiency of transmission is better measured in cycles spent per transmitted byte. We see that even transmitting two records at a time makes Zero Copy more CPU efficient in transmission and the gains get bigger for bigger transmissions.

Memory bandwidth is a crucial resource in an in-memory database server bound by DRAM capacity and latency. We found that for transfer of large records, the pressure on the memory controller is approximately 2X the transmission throughput and while transmitting at the line rate of our NIC, the server consumed half of all available memory bandwidth. If memory bandwidth is not an expendable resource, we ascertain that the designer should always use Zero Copy.

Intel®'s recent micro-architectures have support for DDIO that treats LLC as the source and destination for I/O transfers. This has a significant impact on the memory pressure exerted to the system, especially for Copy Out. DDIO paints a slightly different picture favoring Copy Out since the cost of prefetching the data is paid up front; we do not incur additional misses from DDIO. This is less relevant in the bigger scheme of things and Zero Copy still wins in terms of total consumed memory bandwidth in the server

The transmission throughput anomaly where the throughput of Zero Copy is limited while transferring a large set of small scattered records is best explained to our knowledge with PCIe errors tapering off with 16 records per transmission.

We argued for a client-assisted design with no-update-in-place records that can gain the benefits of Zero Copy while retaining the transmission throughput of larger records. We get a 50% reduction in memory bandwidth consumption using this approach without compromising transmission throughput. Having no updates in place and latch and lock freedom make this approach clearly favorable for the modern NIC.

We also laid out the basic design for a new migration protocol in RAMCloud that makes use of Zero Copy, smart layout for transmission and bypassing re-replication using RDMA.

6.1 Future Directions

1. Investigate and build more hybrid data structures for Zero Copy; we saw how a lock and latch free data structure that can support no-update-in-place fits well with the NIC's Zero Copy structure. While we employed a similar structure in our benchmark to show the benefits, there is a lot of complexity associated with a full implementation of such a structure. It should also be mentioned that there may be other structures that are uniquely capable of exploiting the modern NIC's performance characteristics.

2. Evaluate impact on column stores; we discuss the possible impact of column stores and how the row-oriented result format might cause problems doing Zero Copy. We need to actually measure the transmit performance and resource impact for a complete implementation of a column oriented, in-memory database.

3. Analyze throughput anomaly for small records in depth; while DRAM traffic resulting from PCIe errors correlates with the performance dip observed in Zero Copy, they do not explain why this happens for exactly when we transmit 2048 B (16 records). There needs to be a more detailed analysis around this dimension.

4. Build an actual migration protocol for RAMCloud; the set of guidelines provided in the thesis only serves as a reference point; there would be significant challenges in the actual implementation of a migration protocol for a complex system such as RAMCloud where we may also need to generalize the protocol for older network hardware for backward compatibility.

5. Evaluation of a wider variety of NICs; we based all of our experiments on Mellanox Infiniband ConnectX-3®NICs. At the time of writing, Mellanox has already released plans of releasing NICs that offer 4× maximum throughput and latencies of a few hundred nanoseconds. The scatter gather length of thirty two was an important constant in all our observations; it would be interesting to note how the behavior of the system will change with a scatter gather list of a different length. We also need to

draw parallel conclusions for other advanced NICs that may not explicitly support S/G DMA like the Mellanox hardware.

6.2 Lessons Learned

We started out to measure the impact of data layout and pre-aggregation on network transmit performance. It was rewarding to explore the complexities of handling kernel by-pass over the InfiniBand library and scoreboarding in a multithreaded environment to arrive at the right measurements for transmission. It was also challenging to profile the code for additional metrics reflecting system resource consumption without disrupting the actual transmissions.

I learned a lot while developing many portions of this benchmark. My most important takeaway from this work is the virtue of having truly repeatable experiments. Having hardware environments that are set up to encourage repeatable research and the seemingly disproportionate amount of time spent on developing the tooling of the framework certainly helped in this regard, especially for comparing results after code modifications and root-cause analysis. I got first-hand experience in concurrent programming, use of InfiniBand verbs, RDMA, and accurate cycle counting for profiling wall time. I have significantly improved my scripting and documentation skills over the course of this work. I have also learned valuable lessons on teamwork and collaboration and received a lot of help from my advisor and peers over the course of the work. I believe the guidelines that we drew from this work will help researchers of the future develop better performing in-memory databases for large transfers.

6.3 Parting Thoughts

Today's database designer has to navigate a complex, multidimensional space of trade-offs and network transmission is a critical aspect of performance for the distributed, scale out stores of tomorrow. Smart layout of data and co-designing for hardware are paramount in extracting maximum network performance without consuming unnecessary system resources. The use of Zero Copy offers better transmission throughput and memory bandwidth utilization for large transfers if we carefully co-design the software keeping the network hardware in mind. Research on how to extract more transmission performance from column-oriented layouts and advanced indexing structures should continue until we bridge the gap between what a modern distributed database server can deliver and what the hardware, especially fast networks, is capable of doing.

REFERENCES

[1] *ConnectX®-6 en single/dual-port adapter asic supporting 200gbe*. http://www.mellanox.com/page/products_dyn?product_family=268&mtag=connectx_6_en_ic. Accessed: 01-18-2017.

[2] *DPDK documentation*. http://dpdk.org/doc/guides-16.04/.

[3] *Intel performance counter monitor*. http://www.intel.com/software/pcm. Accessed: 01-07-2017.

[4] *Intel®Data Direct I/O technology*. http://www.intel.com/content/www/us/en/io/data-direct-i-o-technology.html. Accessed: 10-19-2016.

[5] *Mellanox ConnectX-3 product brief*. https://www.mellanox.com/related-docs/prod_adapter_cards/ConnectX3_EN_Card.pdf.

[6] *Mellanox ConnectX-4 product brief*. http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-4_VPI_Card.pdf.

[7] D. Abadi, S. Madden, and M. Ferreira, *Integrating compression and execution in column-oriented database systems*, in Proceedings of the 2006 ACM SIGMOD international conference on Management of data, ACM, 2006, pp. 671–682.

[8] D. J. Abadi, P. A. Boncz, and S. Harizopoulos, *Column-oriented database systems*, Proceedings of the VLDB Endowment, 2 (2009), pp. 1664–1665.

[9] D. J. Abadi, S. R. Madden, and N. Hachem, *Column-stores vs. row-stores: How different are they really?*, in Proceedings of the 2008 ACM SIGMOD international conference on Management of data, ACM, 2008, pp. 967–980.

[10] D. J. Abadi, D. S. Myers, D. J. DeWitt, and S. R. Madden, *Materialization strategies in a column-oriented dbms*, in Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on, IEEE, 2007, pp. 466–475.

[11] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood, *Dbmss on a modern processor: Where does time go?*, in VLDB" 99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK, no. DIAS-CONF-1999-001, 1999, pp. 266–277.

[12] 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.

[13] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, and E. Zamanian, *The End of Slow Networks: It's Time for a Redesign*, Proceedings of the VLDB Endowment, 9 (2016), pp. 528–539.

[14] R. Budruk, D. Anderson, and T. Shanley, *PCI express system architecture*, Addison-Wesley Professional, 2004.

[15] M. J. Carey, M. J. Franklin, and M. Zaharioudakis, *Fine-grained sharing in a page server OODBMS*, vol. 23, ACM, 1994.

[16] M. Chan, H. Litz, and D. R. Cheriton, *Rethinking network stack design with memory snapshots.*, in HotOS, 2013.

[17] S. Das, S. Nishimura, D. Agrawal, and A. El Abbadi, *Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration*, Proceedings of the VLDB Endowment, 4 (2011), pp. 494–505.

[18] 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.

[19] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, *FaRM: Fast Remote Memory*, in USENIX NSDI, Seattle, WA, Apr. 2014, USENIX Association.

[20] 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.

[21] A. J. Elmore, V. Arora, R. Taft, A. Pavlo, D. Agrawal, and A. El Abbadi, *Squall: Fine-grained live reconfiguration for partitioned main memory databases*, in Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, ACM, 2015, pp. 299–313.

[22] A. J. Elmore, S. Das, D. Agrawal, and A. El Abbadi, *Zephyr: live migration in shared nothing databases for elastic cloud platforms*, in Proceedings of the 2011 ACM SIGMOD International Conference on Management of data, ACM, 2011, pp. 301–312.

[23] B. Fitzpatrick, *Distributed caching with memcached*, Linux journal, 2004 (2004), p. 5.

[24] K. Fraser, *Practical lock-freedom*, PhD thesis, University of Cambridge, 2004.

[25] K. Gildea, R. Govindaraju, D. Grice, P. Hochschild, and F. C. Chang, *Remote direct memory access system and method*, Aug. 30 2004. US Patent App. 10/929,943.

[26] M. Grund, J. Krüger, H. Plattner, A. Zeier, P. Cudre-Mauroux, and S. Madden, *Hyrise: a main memory hybrid storage engine*, Proceedings of the VLDB Endowment, 4 (2010), pp. 105–116.

[27] S. Idreos, M. L. Kersten, S. Manegold, et al., *Database cracking.*, in CIDR, vol. 7, 2007, pp. 68–78.

[28] S. Idreos, S. Manegold, H. Kuno, and G. Graefe, *Merging what's cracked, cracking what's merged: adaptive indexing in main-memory column-stores*, Proceedings of the VLDB Endowment, 4 (2011), pp. 586–597.

[29] 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.

[30] 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.

[31] A. Kalia, M. Kaminsky, and D. G. Andersen, *Fasst: fast, scalable and simple distributed transactions with two-sided (rdma) datagram rpcs*, in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), USENIX Association, 2016, pp. 185–201.

[32] R. Kallman, H. Kimura, J. Natkins, A. Pavlo, A. Rasin, S. Zdonik, E. P. Jones, S. Madden, M. Stonebraker, Y. Zhang, et al., *H-store: a high-performance, distributed main memory transaction processing system*, Proceedings of the VLDB Endowment, 1 (2008), pp. 1496–1499.

[33] 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.

[34] A. Kemper and T. Neumann, *Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots*, in Data Engineering (ICDE), 2011 IEEE 27th International Conference on, IEEE, 2011, pp. 195–206.

[35] A. Kesavan, R. Ricci, and R. Stutsman, *To copy or not to copy: Making in-memory databases fast on modern NICs*, in Proceedings of the Fourth International Workshop on In-Memory Data Management and Analytics (IMDM), Sept. 2016.

[36] P.-Å. Larson and J. Levandoski, *Modern main-memory database systems*, Proceedings of the VLDB Endowment, 9 (2016), pp. 1609–1610.

[37] C. Lee, S. J. Park, A. Kejriwal, S. Matsushita, and J. Ousterhout, *Implementing linearizability at large scale and low latency*, in Proceedings of the 25th Symposium on Operating Systems Principles, ACM, 2015, pp. 71–86.

[38] J. Levandoski, D. Lomet, S. Sengupta, R. Stutsman, and R. Wang, *High Performance Transactions in Deuteronomy*, in Proc. of CIDR, 2015.

[39] 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.

[40] S. Li, H. Lim, V. W. Lee, J. H. Ahn, A. Kalia, M. Kaminsky, D. G. Andersen, O. Seongil, S. Lee, and P. Dubey, *Architecting to achieve a billion requests per second throughput on a single key-value store server platform*, in ACM SIGARCH Computer Architecture News, vol. 43, ACM, 2015, pp. 476–488.

[41] S. Liang, W. Yu, and D. K. Panda, *High performance block i/o for global file system (gfs) with infiniband rdma*, in 2006 International Conference on Parallel Processing (ICPP'06), IEEE, 2006, pp. 391–398.

[42] S. Lie, K. Asanovic, B. C. Kuszmaul, and C. E. Leiserson, *Hardware transactional memory*, (2004).

[43] 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.

[44] 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.

[45] D. Makreshanski, J. Levandoski, and R. Stutsman, *To lock, swap, or elide: on the interplay of hardware transactional memory and lock-free indexing*, Proceedings of the VLDB Endowment, 8 (2015), pp. 1298–1309.

[46] M. M. Michael, *High performance dynamic lock-free hash tables and list-based sets*, in Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures, ACM, 2002, pp. 73–82.

[47] C. Mitchell, Y. Geng, and J. Li, *Using one-sided rdma reads to build a fast, cpu-efficient key-value store.*, in USENIX Annual Technical Conference, 2013, pp. 103–114.

[48] J. C. Mogul, *Tcp offload is a dumb idea whose time has come.*, in HotOS, 2003, pp. 25–30.

[49] D. H. Nguyen, M. T. C. Van An Le, T. V. Pham, and N. Thoai, *Accelerating range query execution of in-memory stores: A performance study*.

[50] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, et al., *Scaling memcache at facebook*, in Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13), 2013, pp. 385–398.

[51] 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.

[52] D. Ongaro and J. K. Ousterhout, *In search of an understandable consensus algorithm.*, in USENIX Annual Technical Conference, 2014, pp. 305–319.

[53] D. Ongaro, S. M. Rumble, R. Stutsman, J. Ousterhout, and M. Rosenblum, *Fast crash recovery in ramcloud*, in Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, ACM, 2011, pp. 29–41.

[54] J. Ousterhout, P. Agrawal, D. Erickson, C. Kozyrakis, J. Leverich, D. Mazières, S. Mitra, A. Narayanan, G. Parulkar, M. Rosenblum, et al., *The case for ramclouds: scalable high-performance storage entirely in dram*, ACM SIGOPS Operating Systems Review, 43 (2010), pp. 92–105.

[55] 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 (2015), pp. 7:1–7:55.

[56] G. F. Pfister, *An introduction to the infiniband architecture*, High Performance Mass Storage and Parallel I/O, 42 (2001), pp. 617–632.

[57] J. Pinkerton, *The case for rdma*, RDMA Consortium, May, 29 (2002), p. 27.

[58] H. Plattner, *A common database approach for oltp and olap using an in-memory column database*, in Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, ACM, 2009, pp. 1–2.

[59] R. Rex, F. Mietke, W. Rehm, C. Raisch, and H.-N. Nguyen, *Improving communication performance on infiniband by using efficient data placement strategies*, in 2006 IEEE International Conference on Cluster Computing, IEEE, 2006, pp. 1–7.

[60] 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 (2015).

[61] W. Rödiger, T. Mühlbauer, A. Kemper, and T. Neumann, *High-speed Query Processing over High-speed Networks*, Proc. VLDB Endow., (2015).

[62] S. M. Rumble, A. Kejriwal, and J. Ousterhout, *Log-structured memory for dram-based storage*, in Proceedings of the 12th USENIX Conference on File and Storage Technologies (FAST 14), 2014, pp. 1–16.

[63] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, et al., *C-store: a column-oriented dbms*, in Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, 2005, pp. 553–564.

[64] R. S. Stutsman, *Durability and crash recovery in distributed in-memory storage systems*, PhD thesis, Stanford University, 2013.

[65] The CloudLab Team, *Cloudlab web site*. `https://cloudlab.us`.

[66] V. Tipparaju, G. Santhanaraman, J. Nieplocha, and O. Panda, *Host-assisted zero-copy remote memory access communication on infiniband*, in Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International, IEEE, 2004, p. 31.

[67] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden, *Speedy transactions in multicore in-memory databases*, in Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, ACM, 2013, pp. 18–32.

[68] T. Von Eicken, A. Basu, V. Buch, and W. Vogels, *U-net: A user-level network interface for parallel and distributed computing*, in ACM SIGOPS Operating Systems Review, vol. 29, ACM, 1995, pp. 40–53.

[69] 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.