LARGE TRANSFERS OF IN-MEMORY DATA MADE FAST AND DISCREET

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

November 2016

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

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

Ryan Stutsman ,	Chair(s)	**21 Nov 2016** Date Approved
Robert Ricci ,	Member	**21 Nov 2016** Date Approved
Feifei Li ,	Member	**21 Nov 2016** Date Approved

by **Ross Whitaker** , Chair/Dean of
the Department/College/School of **Computing**
and by **David 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 DRAM.

These systems were designed with OLTP workloads in mind and, as a result, are optimized for fast dispatch and performs well under small request-response scenarios. However, heavy responses such as 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 NICs. The present-day NIC offers exciting opportunities and challenges for large transfers, and we can make them efficient by leveraging smart data layout and concurrency control.

We evaluated the impact of modern NICs on data layout by measuring transmit performance while tuning the effects of DMA, RDMA and caching advancements. We proposed guidelines leveraging a smart co-design of data layout, concurrency control, and recent improvements to the NICs that result in increased overall efficiency and throughput. We also set up experiments that underlined the bottlenecks in current approaches to data migration in RAMCloud. We propose guidelines for a fast and efficient migration protocol with minimal disruption to the normal case operation.

We have found evidence to prove that performing large data transfers in a live cluster of systems optimized for smaller responses has a detrimental effect to their primary intent. We have analyzed the new possibilities that open up as a result of an enhanced migration protocol. We propose the guidelines for a new migration protocol which will will supplement the mainstream acceptance of these systems.

TIMELINE

This section gives an overview of the timeline associated with the work involved in the thesis. Chapter 5 explains the proposed work in detail .

Task 1	Develop a microbenchmark to measure performance of infiniband verbs	Feb 2016	Done
Task 2	Analyze performance of RDMA verbs over ibverbs library	Mar 2016	Done
Task 3	Measure Transmit performance and CPU cycles for Zero Copy and Copy Out	Apr 2016	Done
Task 4	Compare Peformance of Delta Records	May 2016	Done
Task 5	Write up and submit results to IMDM-2016	Jul 2016	Done
Task 6	Follow up measurements on Connect-IB®and hugepages	Aug 2016	Done
Task 7	Researching state of the art protocols for in-memory migration	Oct 2016	Done
Task 8	Write up Abstract	Oct 2016	Done
Task 9	Finish Writing Chapter on Introduction	Nov 2016	Done
Task 10	Analyse effects of DDIO® on memory bandwidth	Nov 2016	In Progress
Task 11	Finish the chapter on modern NICs	Nov 2016	In Progress
Task 12	Defend the thesis proposal	Nov 2016	In Progress
Task 13	Get uncore and offcore performance measurements from the microbenchmark	Dec 2016	TBD
Task 14	Analysing bottlenecks in RAMCloud migration protocol	Dec 2016	TBD
Task 15	Finish writing migration chapter	Jan 2017	TBD
Task 16	Come up with guidelines for new protocol	Jan 2017	TBD
Task 17	Finish writing Conclusion	Jan 2017	TBD
Task 18	Final thesis defense	Feb 2017	TBD

Table 0.1: Timeline of tasks involved in thesis.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

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 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 (NIC). As the primary I/O device in present-day databases, 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 [37] 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 which are present in a modern NIC. To thoroughly understand the complexities involved in NIC in doing huge transfers of data, we profiled a modern NIC with a focus on heavy data transfers. We made several discoveries that informed our design for RAMCloud, but they also generalize to other transfer-heavy database operations.

This thesis is structured into three key pieces. The first microbenchmarks a modern NIC with kernel bypass and RDMA and uses the findings to inform an efficient migration design for RAMCloud. The second piece is a breakdown of RAMClouds existing migration mechanism to understand its bottlenecks and to assess why it is incapable of fully exploiting modern server hardware. Finally, our forward-looking objective is a new design for migration that we believe will advance state of the art in terms of time taken to

migrate large amounts of data within a cluster of in-memory storage systems with minimal disruption to the normal case operation.

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 an on-device DMA engine from the network card can access application memory without invoking the CPU. This development led to the Remote Direct Memory Access (RDMA) [15] 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 encounter many trade-offs when trying to use the NIC effectively.

Our mission was to come up with a migration protocol that advances state of the art for in-memory storage systems. We realized the importance of the making data transfer NIC friendly and set out to measure what variables impact the transmission performance. We wanted to enumerate and underline the challenges in designing keeping NIC as a first class citizen. We specifically looked at three factors that make it especially challenging to design NIC-friendly database engines:

- *Zero-copy* DMA engines reduce server load for transferring large data blocks, but large, static blocks are uncommon for in-memory databases, where records can be small, and update rates can be high.

- The performance gains from zero-copy DMA do not generalize to *finer-grained objects* 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.

- 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 [1], provide direct access to last level cache from an I/O device. This effect impacts the memory pressure 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 [22].

Our primary results gave us ideas about how the Zero Copy paradigm will perform against the traditional approach of transmitting via larger buffers. These also pointed to how the use of thread local storage and effective network aware caching such as DDIO [1] somehow refutes the argument for paging records together.

Another burden of using these newer storage systems are how they would provide effective reconfiguration. Our results from the benchmark led us to believe in the importance of hardware considerations in data migration. State of the art systems like RAMCloud has around 1000x lower latency than traditional storage systems, and these tight SLAs make it harder to do migrations without interfering with regular requests. This is also one of the key hurdles to overcome before mainstream acceptance of these systems. Our findings about modern NICS inform a new design for cluster reconfiguration in the RAMCloud storage system. Low-cost load balancing improves RAMClouds efficiency since indexes [21] and transactions [24] benefit from collocation, but, at the same time, RAMClouds exceedingly tight SLAs (5 μs median access times and 99.9th of reads within 10μs) mean any disruption to performance has a serious impact. For an in-memory database, we would ideally like to saturate the available NIC bandwidth which is 8-12GB/s [2, 3] while state of the art transfer mechanisms offer many orders of magnitudes lower than that at 10MB/s throughput and only offer 1000-10000x higher latency than RAMCloud [37]. 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. Our design for fast and efficient migration for RAMCloud is guided by the following key principles:

- **High Throughput**: State of the art systems do migrations at $<$100 Mb/s. Our plan should result in 100-1000x faster data migrations

- **Low Impact**: RAMCloud has 99.9 percentile read latency at $<10\mu$s and write latency at $<100\mu$s. Our proposed protocol will be of minimal impact to the normal case operation, still keeping up with SLAs that are 10-100x tighter than other systems [12].

- **Late-partitioning**: RAMCloud promises uniform access latency regardless of the level of data locality. Most other systems diverge from this approach to make data movement faster. We would like to stick with deferring partitioning decisions as much as possible and this adds a challenge in preparing data for migration.

We evaluate the current state of the art approaches to migration in 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 throughput at *source of migration*; Load balancing is a key problem that could be addressed with migration and measuring the overhead introduced by migration is helpful here.

- Analyzing the effect of splitting and migrating *hot indexes*; Indexed reads add value to in-memory stores in utilizing them as a complete database.

- Exploring effects of *locality of data*; RAMCloud's unique take on locality makes its distributed recovery [36, 42] interesting because of its effect of distributing recovered data. We will explore the impact of recovery on locality and how reconfiguration can repair to restore locality.

- *Scaling clusters up and down* will show the effects of data movement in many-to-one and one-to-many data redistributions.

These will explore the themes of data locality, source impact and the effect of numerous workloads which include scaling up and down the cluster, splitting and migrating hot index tables, the impact on locality by the distributed recovery which sprays data across

the cluster and responding to changes in access patterns taking into account the multi-dimensional resource contentions that could occur in a practical system.

We will also propose the outline for an effective migration protocol that leverages the log structured memory of RAMCloud as well as the distributed crash recovery mechanism to provide fast and jitter-free data movement in a cluster for reconfiguration and improved performance for range queries and distributed transactions. We will try to do so while deferring partitioning decisions as much as possible.

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; We hypothesize that carefully leveraging data layout and advancements in modern NICs will yield to 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 up 75% reduction of CPU overhead and get a throughput boost of 1.7x 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 an fast and efficient migration protocol which causes minimal disruption.

The following are the main contributions from this work which supports and provides evidence for the thesis statement.

- **Evaluation of Impact of Modern NICs on data layout**: This is the first published research exploring the scatter gather DMA engine of the NIC in contrast with the traditional copy out approach in the context of bulk transfers. 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 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.

- **Experiments to motivate fast and efficient data transfer**: This is the first assessment of the impact 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 relatively uniform access cost to all nodes. We quantify the gains from explicitly colocating records and indexes. 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.

- **The design of client assisted responses 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 indices can exploit NIC hardware. Through the benchmark, we show 70% improvement in throughput using such a design.

- **Preliminary design for a migration protocol**: After careful analysis of the current migration protocol we discovered the most important bottlenecks in the current design that makes 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

MODERN NICS

Network Interface Cards have grown complex over time and present opportunities and challenges as part of a modern distributed system. It is imperative that today's database system designer treats network transmission as a first class citizen in every design decision. Previous research for optimising databases were focussed more on processing time spent on a database query [4], but today's truth is that lion's share of the end-to-end latency while processing a query is spent in network. Since we are not reeling from slow disks anymore, the bulk of I/O bottleneck in a distributed 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 utilised for useful work

Network cards have had TCP offloading baked in as TCP Offload Engine for some time now, but it was getting more and more ineffective because of performance issues and complexities involved [33]. RDMA [15, 19, 39] was proving to be a much better alternative with TCP offloading where transport offloading was just one among varios benefits involved. Development of advanced switching interconnects such as PCI express [7] made it possible to develop highly performant network devices offering throughputs and latencies a couple of orders of magnitudes better than what was possble with prior transports such as TCP.

2.1 Zero Copy and Copy Out

Kernel bypass is a ubiquitous feature in the modern NIC and key to accessing remote memory directly from across the network. While the literal definition just defines any data movement that doesn't involve copying between user space and kernel space, even packet capturing libraries [17] could fall under those definitions. Newer network controllers make use of vectored I/O, otherwise known as scatter/gather I/O where a single procedure

reads data from multiple buffers and writes it to single buffer. The Mellanox Infiniband ConnectX®-3 NIC that we profiled provides a scatter/gather list which powers it's on NIC DMA engine. This specific kernel bypass mechanism what we are going to colloqially refer in the thesis as Zero Copy.

Zero Copy should be christened in the context of traditional copy. Before kernel bypass became commonplace, the steps for copying data over the network using socket programming was as follows. The application calls something similar to a socket send 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 send out of the door. We could employ the same technique in the kernel bypass capable NIC as well. Instead of employing a multi entry scatter/gather list, the records could be assembled on to 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" from now on.

Zero copy DMA facilitates a scatter gather list of buffer descriptors which could be mapped to non contigous locations in memory on demand. These provide the added benefit that network headers don't need to exist along with data and helps bring more flexibility in deciding the transport layer. One subtle thing to note here is that Zero Copy only implies the absence of a memcopy from 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 is different from what happens when we call socket send on a traditional tcp stack where the data for transmission is pre assembled in a huge buffer and then copied across the network. We evaluate how Zero Copy performs in contrast with the more traditional network copy approach reffered as Copy Out from now on.

2.2 Memory Bandwidth

Interestingly, the additional copy involved in the traditional approach hurts memory bandwidth of the system more than contributing to additional CPU load purely from the perspective of network transmission. Our evaluation show us that while the traditional copy just adds a modest 5% increase in absolute CPU utilisation, making use of the high throughput available in NICs by transmitting near line rate takes up one third of the total available memory bandwidth in a modern server. We should read this in the context that

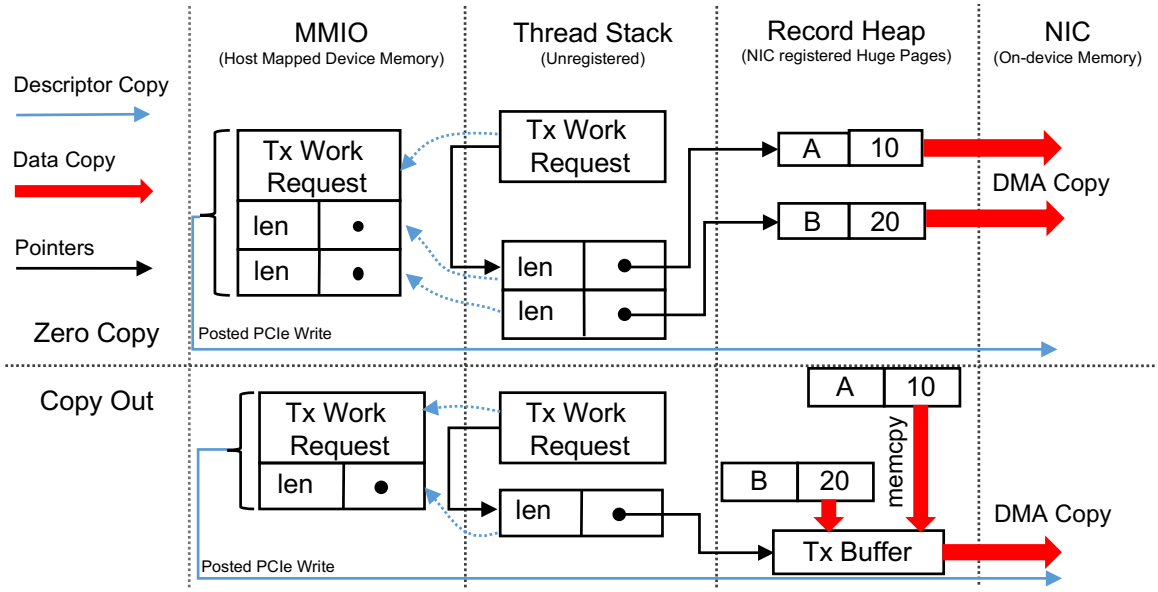

Figure 2.1: Key structures involved in network transmission

network transmission is not the primary responsibility of a server in a distributed system. Most of the memory bandwidth wasted in just aggregating the data before transmission could be used as part of actual computation or other useful rearrangement of data in the query response.

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 original 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 own page tables. Registration is done at startup and is often done with physical memory backed by 1 GB hugepages to minimize

on-NIC address translation costs.

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

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

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

2.4 DDIO

performance-oriented enhancements to the DMA mechanism have been introduced in Intel®Xeon E5 processors with their Data Direct I/O (DDIO) [1] feature, allowing the DMA "windows" to reside within CPU caches instead of system RAM. As a result, CPU caches are used as the primary source and destination for I/O, allowing network interface controllers (NICs) to talk directly to the caches of local CPUs and avoid costly fetching of the I/O data from system RAM. As a result, DDIO reduces the overall I/O processing latency, allows processing of the I/O to be performed entirely in-cache, prevents the available memory bandwidth from becoming a performance bottleneck.

We need to fully investigate the effects of DDIO in reducing memory bandwidth in order to conclusively comment on the impact of traditional copy mechanisms.

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 inside the descriptor that is posted to the NIC control ring buffer. Inlining can improve messaging latency by eliminating the delay for the NIC to DMA the message data from host DRAM, which can only happen after the NIC receives the descriptor. Inlining benefits small request/response exchanges, but it does not help for larger transmissions. This is because even though there is an extra delay before the NIC receives the actual record data, that delay can be overlapped with the DMA and transmission of other responses. Other researchers have shown that sending data to the NIC via MMIO also wastes PCIe bandwidth [19]. Almost all of our experiments have inlining disabled. Enabling inlining gives almost identical throughput and overhead to copy-out, except it only works for transmissions of 912 B or less.

2.6 Evaluation

We explored hows 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 [40] cluster of the CloudLab [43] testbed: this testbed provides exclusive bare-metal access to

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

Table 2.1: Experimental cluster configuration.

a large number of machines with RDMA-capable Infiniband NICs. The proposed work that is pending will give more conclusive evidence for the 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 the experiments are publicly available online[1].

2.6.1 RDMA modes

We started out by getting measurements of transmission speeds using RDMA Reads. We implemented rdma operations using infinibands ib verbs library. We implemented RDMA reads and write as op codes on the `ibv_send_wr` request which transmit data as `ibv_post_send` and also tried to measure the transmit performance while using zero copy using `IBV_WR_SEND` as the operation. We found that send operations can transmit multiple transmit buffer It became evident quickly that one sided RDMA reads are not well positioned to take advantage of the zero copy paradigm since it only supports a remote read from a given location from the source. The rest of the experiments concerning the impacts of NIC's impact on data layout were all done with send as the primary operation.

Figure 2.2 shows a comparison of transmission peformance of various verbs transmitting 100 Byte records varying the number of records that could be transmitted at a time for various operations.The green dot shows the performance for RDMA reads which is also unfair since RDMA reads can only transmit a single record at a time. The yellow dot shows RDMA write performance. The modes shown was an enum of the format

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

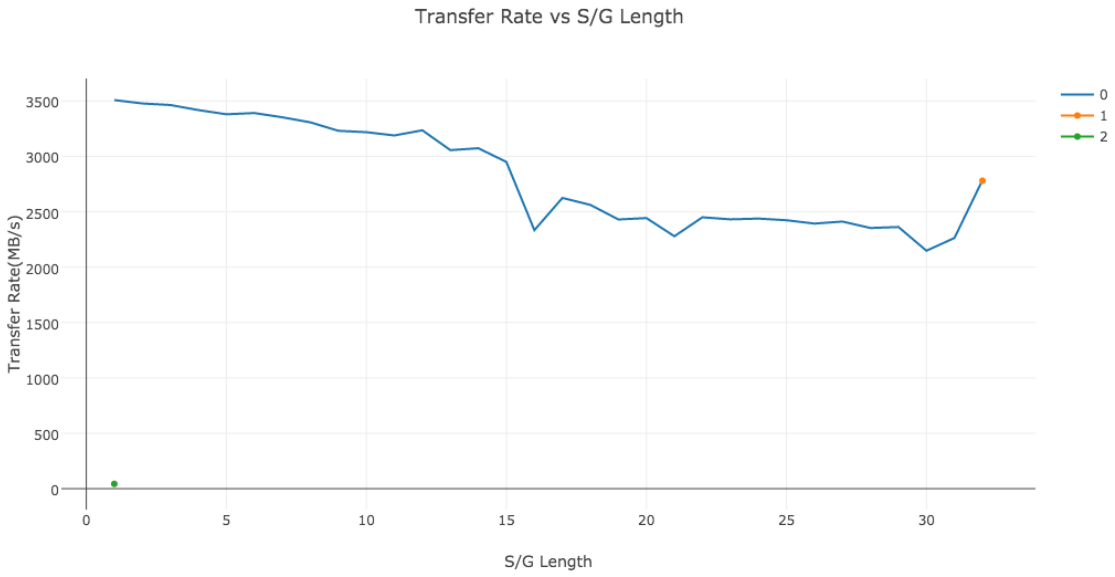

Figure 2.2: Transmission rate for 100 B records over different RDMA verbs and varying S/G lengths.

`enum Mode { MODE_SEND, MODE_WRITE, MODE_READ}` and infiniband (`<infiniband/verbs.h>`) verbs library provides the The record sizes were also an interesting measure to arrive at. It was obvious even from an earlier stage that higher data sizes such as shown in Figure 2.3 almost always results in better throughput measurements. In the world of In-memory databases, there is an argument to be made that record sizes will be getting smaller since database designers can aggresively normalise and this has proven increasingly to the liking of companies that manage large volumes of data in memory [5, 35].

2.6.2 Paging

All our experiments transmit from a large region of memory backed by 4 KB pages that contains all of the records. The region is also registered with the NIC, which has to do virtual-to-physical address translation to DMA records for transmission. In some cases, using 1 GB hugepages reduces translation look aside buffer (TLB) misses. We have

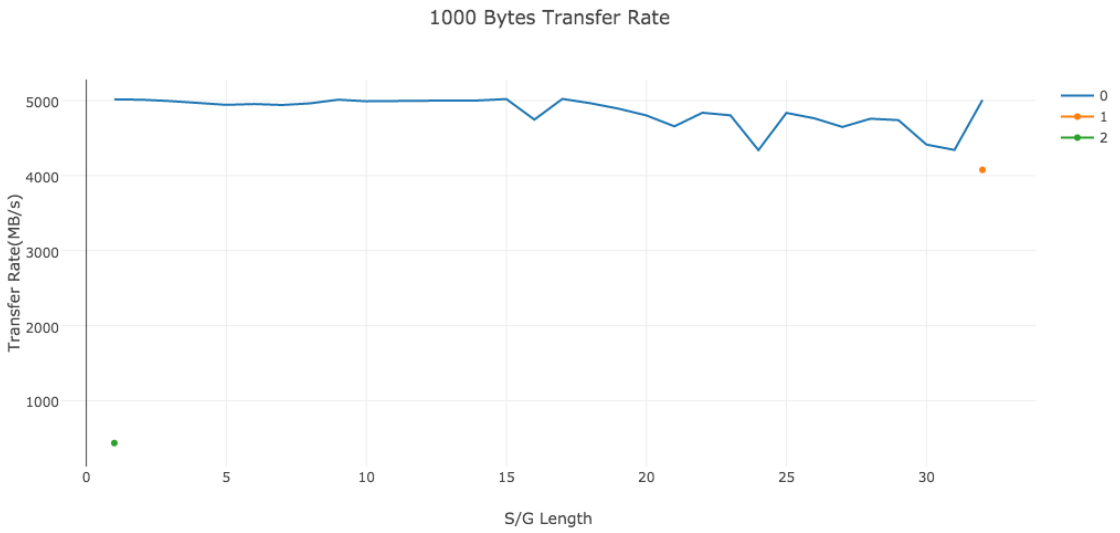

Figure 2.3: Transmission rate for 1000 B records over different RDMA verbs and varying S/G lengths.

realised 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 [10, 19]. For our experiments, the reach of the NIC's virtual-to-physical mapping is sufficient, and hugepages have no impact on the results.

To explore how different designs trade-off database server efficiency and performance, we built a simple model of an in-memory database system that concentrates on data transfer rather than full query processing. In all experiments, one node acts as a server and transmits results to 15 client nodes.

Experiments transmit from a large region of memory backed by 4 KB pages that contains all of the records. The region is also registered with the NIC, which has to do virtual-to-physical address translation to DMA records for transmission. In some cases, using 1 GB hugepages reduces translation look aside buffer (TLB) misses. The NIC can benefit from hugepages as well, since large page tables can result in additional DMA

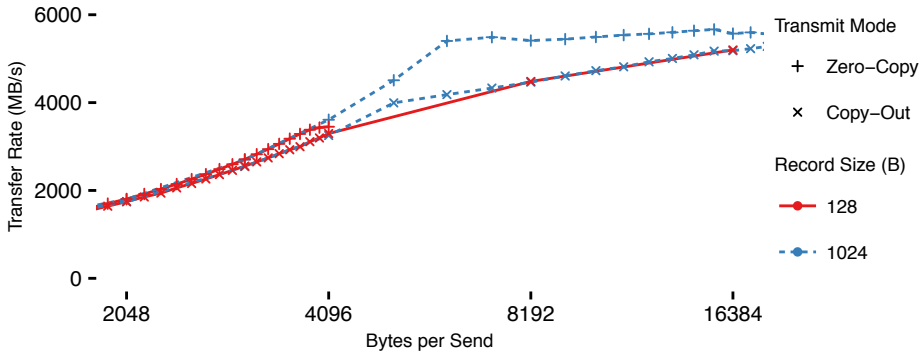

Figure 2.4: Transmission throughput when using conventional copy-out into transmit buffers and when the NIC directly copies records via DMA (zero-copy). The line for 128-byte zero-copy stops at 4K, as this is the maximum size of a send with 32 records of this size.

operations to host memory during address translation [10, 19]. For our experiments, the reach of the NIC's virtual-to-physical mapping is sufficient, and hugepages have no impact on the results.

2.6.3 Zero-copy Performance

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

To understand these trade-offs, we measure the aggregate transmission throughput of a server to its 15 clients under several configurations. In each experiment, the record size, s, is either 1024 B or 128 B. Given a set of records that must be transmitted, they are then grouped for transmission. For zero-copy, an n entry DMA gather descriptor is created to transmit those records where ns bytes are transmitted per NIC transmit operation. For copy-out, each of the n records is copied into a single transmit buffer that is associated with a transmit descriptor that only points to the single transmit buffer. Each transmission still sends exactly ns bytes, but copy-out first requires ns bytes to be copied into the transmit buffer. Intuitively, larger groups of records (larger sends) result in less host-to-NIC interaction, which reduces host load and can increase throughput; the benefits depend on the specific configuration and are explored below.

Figure 2.4 shows how each of these configurations impact transmission throughput. For larger 1024 B records, using the NIC's DMA engine for zero-copy shows clear benefits (aside from CPU and memory bandwidth savings, which we explore in §2.6.3.1). The database server is able to saturate the network with zero-copy so long as it can post 6 or more records per transmit operation to the NIC (that is, if it sends 6 KB or larger messages at a time).

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

Next, we consider 128 B records. The decreased access latency of in-memory databases makes them well-suited to smaller, finer-grained records than were previously common. One expectation is that this will drive databases toward more aggressively normalized layouts with small records. This seems to be increasingly the case as records of a few hundreds bytes or less are now typical [5, 35].

Figure 2.4 shows that for small 128 B records, the NIC DMA engine provides little throughput benefit. Our NIC is limited to gather lists of 32 entries, which is insufficient to saturate the network with such a small record size. Transmission peaks at 3.5 GB/s. Copying 128 B records on-the-fly can significantly outperform zero-copy transmission when there are enough results to group per transmission. In fact, copy-out can saturate the network with small records, and it performs identically to copy-out with larger 1024 B records.

2.6.3.1 Zero-copy Savings

In addition to improved performance, a goal of zero-copy DMA is to mitigate server-side CPU and memory load. Figure 2.5 breaks down CPU time for each of the scenarios in Figure 2.4: small and large records both with and without zero-copy. Most of the server CPU time is spent idle waiting for the NIC to complete transmissions. The results show

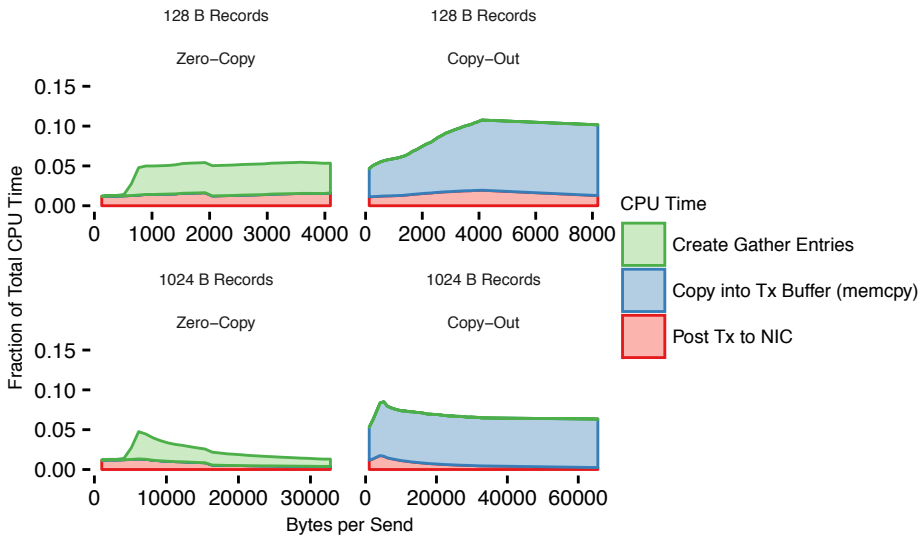

Figure 2.5: Breakdown of server CPU time for transmission of small 128 B records and 1024 B records using copies into transmit buffers (Copy-Out) and zero-copy DMA.

that zero-copy always reduces the CPU load of the server, and, as expected, the effect is larger for larger records. With 1024 B records, the `memcpy/` step of copy-out uses 6.8% of the total CPU cycles of the socket at peak transmission speed.

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

The memory bandwidth savings for zero-copy are more substantial. Figure 2.4 shows that copy-out transmit performance nearly matches zero-copy (5.6 GB/s versus 5.8 GB/s). Copy-out introduces exactly one extra copy of the data, and `memcpy/` reads each cache line once and writes it once. So, copy-out increases memory bandwidth consumption by 2× the transmit rate of the NIC or 11.2 GB/s in the worst case. This accounts for about 32% of the available memory bandwidth for the machine that we used. Whether using zero-copy or copy-out, the NIC must copy data from main memory, across the PCIe bus, to its own

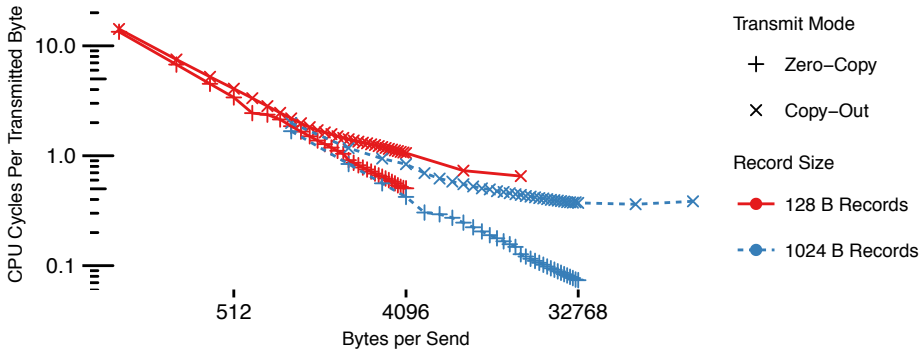

Figure 2.6: Cycles per transmitted byte for large and small records with zero-copy and copy-out. Note the log-scale axes.

buffers, which uses another 6 GB/s of memory bandwidth.

For smaller 128 B records, the CPU and memory bandwidth savings are nearly the same as for larger records. In this case, copy-out uses up to 10.8% of the CPU cycles on the socket, and zero-copy uses 5.5%. Eliminating the extra `memcpy`/ halves CPU overhead for transmission; a savings of about 5% of the total socket's CPU cycles. Just as before, the memory bandwidth savings is twice the transmission rate or 11.2 GB/s. Overall, this makes copy-out reasonable for small records scattered throughout memory, especially since zero-copy cannot saturate the network in these cases (§2.6.3).

The results break down where the CPU and memory bandwidth savings come from, but not all configurations result in the same transmit performance. For example, Figure 2.4 shows that when transmitting 128 B records, copy-out gets up to 53% better throughput than zero-copy. As a result, minimizing CPU overhead can come at the expense of transmit performance. The real efficiency of the server in transmission is shown in Figure 2.6. The figure shows how many cycles of work the CPU does to transmit a byte in each of the configurations, which reveals two key things. First, it shows that, though the absolute savings in total CPU cycles is small for zero-copy, it does reduce CPU overhead due to transmission by up to 76%. Second, the improvement is a modest 12% for small records, since copy-out can transmit in larger, more efficient batches.

CHAPTER 3

BW-TREES

We have seen the impact of Zero Copy on network transmission and how that compares to the more traditional way of sending data over the network. The difference in throughput is explicitly seen for smaller 128 B records. Since databases have started to store all data in memory at all times, database designers have started to normalize aggressively and as a result, record sizes are trending smaller and 128 B records show a more realistic depiction of record sizes in the futures.

At first, the throughput results from comparing Zero Copy and Copy Out for 128 B records might tempt us to think that it's always better to use a traditional copy. This might be more alluring since we have established that the absolute savings on CPU cycles using Zero Copy may not be significant.

One might overlook the bigger problem that lies behind the use of traditional copy, that the memory pressure while doing Copy Out occupies a lion share of available memory bandwidth. While transmitting at around 7 GB/s(near line rate for the Connect-X3®), we believe the memory pressure will be closer to 21GB/s which might take up a third of all available memory bandwidth in a modern server.

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 analyse structures that can uniquely take advantage of the Zero Copy paradigm.

Microsoft®Research came up with a modern B-tree structure called the Bw-Tree [26], which is a high performance ordered index employing lock and latch free concurrency and effective utilisation of caches in modern multi core processors. We discuss it's lock freedom and it's NIC implications that make it well suited for Zero Copy in the following

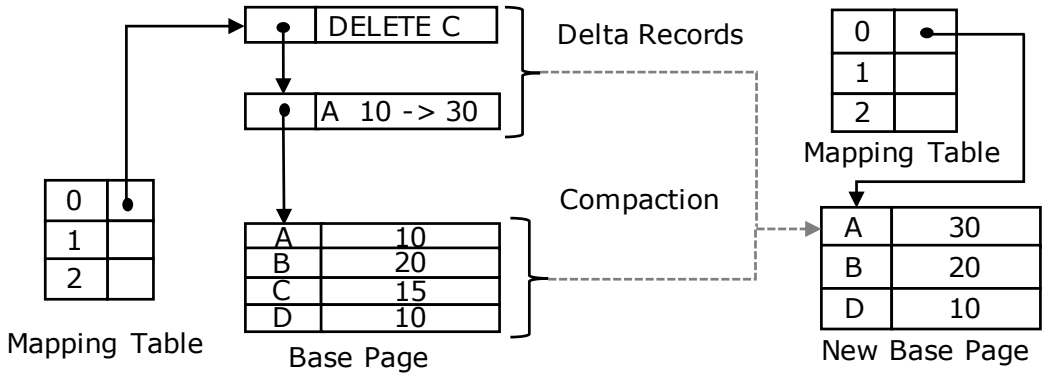

Figure 3.1: The structure of a Bw-Tree.

sections. We then present our evaluation to show the gains in throughput by using delta records in our benchmarks.

3.1 Lock-free Indexing

The Bw-Tree [26] is an atomic record store designed for extreme concurrency. It is an ordered index that supports basic record create, update, and delete (CRUD) operations in addition to search and range scans. It is fully lock-free and non-blocking, and is optimized for modern multicore workloads. It can store to flash, but is also intended for in-memory workloads; it serves as the ordered secondary index structure for in-memory SQL Server Hekaton [9] engine.

In many ways, the Bw-Tree is a conventional paged B-link tree, but it also has unique characteristics that interact with network-layer design choices. Its lock-freedom, its elimination of update-in-place, and its lazily consolidation of 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. A 16KB base page is transmitted here along with a varying number of delta records, results are shown for both 128 B and 1024 B records. In place update 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.

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

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

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

epoch and prevent reclamation of memory that has been unlinked from the data structure.

3.2 NIC Implications for Bw-Tree

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

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

Overall, we seek to answer these key questions:

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

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

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

3.3 Evaluation

3.3.1 Extending the Delta Format to Clients

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

Figure 3.2 shows the benefits of this approach. In this experiment, each transmission consists of a single 16 KB base page while the number and size of the delta records attached to each transmission is varied. The NIC benefits from the large base page, and it manages to keep the network saturated. CPU overhead ranges from less than 1% when there are a few delta records up to about 2% when there are more. Compared to zero-copy of scattered small records this approach also yields a 1.7$\times$ throughput advantage; Figure 2.4 shows that throughput peaks at 3.4 GB/s when records are scattered, while the delta format achieves 5.7 GB/s. The main cost to this approach is the cost of consolidating delta records into new base pages when chains grow long; we consider this overhead in more detail in the next section.

3.3.2 Tuning Page Consolidation

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

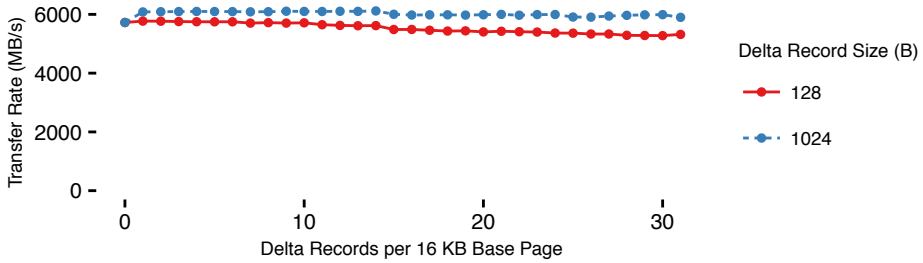

Figure 3.2: Zero-copy transmit performance when transmitting using Bw-Tree like delta records

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

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

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

3.3.3 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 **??**. Normally, the Bw-Tree copies-out returned records. While an operation and its copy-out are ongoing, memory that contains records that have been unlinked or superseded cannot be reclaimed. Zero-copy effectively defers the copy until the actual time of transmission. As a result, transmissions completions must notify the higher level database of transmission completion to allow reclamation. This delay results in more latent garbage and hurts the effective memory utilization of the system.

In practice, this effect will be small for two reasons. First, zero-copy adds a transmission, which completes within a few microseconds; however, it also saves a `memcpy/` which accounts for 1 to 2 μs for a 16 KB transmission. Second, the amount of resulting data held solely for transmission should generally be more than compensated for by eliminating the need for explicit transmit buffers. With copy-out, the size of the transmit buffer pool is necessarily larger than the amount of data under active transmission, and internal fragmentation in the transmit buffers makes this worse.

CHAPTER 4

RELATED WORK

In-memory databases and low latency networks are very heavily researched areas these days. Our work borrows concepts from numerous efforts both *à*- la-mode and prior from the databases, networking and systems research communities. RAMCloud [37] itself was a research product built on top of other work on log-structured file systems, striping filesystems and distributed crash recovery [36, 42]

4.1 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 didn't diverge from the tried and tested model for locality of reference. Conventional "secondary storage" devices such as disks were the final resting place of evergrowing 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 was dragging performance at scale. Memcached [13] is one such phenomenal system that stood the test of time and it's further development enabled it to operate at a scale of billions of operations per second on trillions of items [34].

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 by means of distributed recovery [36]. A more 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 [38] became more commonplace in data center networks. These developments have led to a

deluge of storage systems research [23] , exploring multiple verticals. Among the particularly notable ones are RAMCloud [37], FaRM [10] and H-Store [20] (Scale out memory stores) and MICA [29], HyPer [41] and HERD [18] (Focused on single node performance). While each of these systems have been driven by different dimensions of research, they all share the same concerns of In-memory storage systems operating on high speed networks.

Main memory database systems of today offers a plethora of features that are at par with traditional database systems including data storage and indexing, concurrency control, durability and recovery techniques, query processing and compilation, support for high availability, and ability to support hybrid transactional and analytics workloads [23]. FaRM [10] is a phenomenal In-memory, distributed computing platform that leverage lock free reads over RDMA [19], collocating objects and function shipping to provide performance characteristics of the highest order for an In-memory database. They could even scale distributed transactions to hundreds of millions of operations with recovery times of the order of milliseconds. There are other systems such as Silo [44] which effectively utilises multi core CPUs.

4.1.1 Lock freedom

Lock freedom is important for Zero Copy and RDMA since these complicates concurrency since you have less control over NIC read and writes. Lock and latch freedom are design philosophies perfected on previous work on fine grained concurrency primitives [8] and lock free hash tables [32]. Lock freedom is commonly used in databases these days [31] since they gained practicality by common use of epochs [14] based reclamation. More recent iterations of SQL engines such as Hekaton [9] are examples for In-memory optimised database engines. Bw-Tree [26], a data structure that is lock and latch free and follows a no-update-in-place philosophy to indexing. Others have worked on exploiting Hardware Transactional Memory [28] for In-memory transaction processing [45] as well as lock free indexing [31].

4.2 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. From the early arguments [39] calling for the need for offloading CPUs to research aiming for a billion key value operations on key value servers [27], RDMA has proven to be the future of data transfer for large scale databases. Key value stores such as HERD [18] and MICA [29] and systems that provide more complex data models such as RAMCloud [37] use RDMA effectively and there have been published guidelines to maximise performance for dispatch heavy workloads [19]

4.3 Data Transfer in Fast Networks

As pointed out by various researchers in the past, the days of slow networks are over [6] and the 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 [10, 11, 37, 41, 45] Our findings should also complement other efforts such as transactional key value stores [25] and shared-data databases [30]. Small, fixed-size request/response cycles have been optimized in existing research [10, 18, 19, 29, 37], 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 [19] 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. Squall [12] employs reactive pulls and pre partitioning of data for fast migrations but operates on platforms with throughput and latencies orders of magnitude less than what is possible with today's hardware.

Our work is built on the premises of these research ideas and it's 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 presence of modern networks has not been well understood from the server side.

CHAPTER 5

PROPOSED WORK

This is a forward looking section listing out a few experiments that will strengthen the thesis and pave way for a fast and efficient migration protocol. Table 5.1 shows the expected amount of work that needs to be done in the coming months to conclusively gather evidences that support the thesis. This will also serve as the schedule guideline for the rest of my program.

5.1 DDIO

Modern NICs can directly transmit or receive data from CPU's last level cache. We havent yet fully evaluated this effect, and it could have significant impact on the design of efficient migration. For example, the overhead of `memcpy`and CPU cache misses might at the source of migration may be offset by reduced transmission costs. We plan to repeat our ibv-bench experiments from the IMDM paper to assess whether we can exploit this effect for migration.

Analyse effects of Direct Caching on memory bandwidth: We need to find conclusive evidence about the impact of synchronization and advanced caching mechanism on our experiments. This would involve fully investigating the impact of DDIO [1] on memory bandwidth and figuring out a way to toggle the effect to compare before and after effects of Direct Caching.

Get uncore and offcore performance measurements from the microbenchmark: We need to re-run the ibv-benchmark to come sup with uncore and off-core performance measurements to pinpoint the areas of contention while transmitting huge blocks of data. So far, we have analysed the impacts on memory bandwidth by the traditional copy out approach without considering the changes in access patterns and the effects of Direct Caching [16]. We should investiaate the last level cache characteristics and other perfor-

mance events as prescribed in Intel®'s software development manual in order to gather accurate performance measurements using the perf tool.

Evaluating the effect of thread local storage: The synchronisation primitives that are used in the microbenchmarks should not have a driving effect to the conclusions we make from them. We need to get new numbers for our measurements with thread local buffers to see the impact.

Recalibrating measurements for Connect-IB®: Once we get the measurements accounting for the effects of Direct Caching and thread local buffers, we need to re-evaluate Connect-IB®(newer) cards and compare them with the Connect-X 3®cards to evaluate the difference in throughput that can be achieved by Zero Copy by these devices in the light of new findings.

5.2 Data Migration

We are currently profiling RAMClouds existing migration mechanism, which seems to max out at around 100MB/s, which is about two orders of magnitude away from the NIC's full line rate. The key deliverable is a piece-wise assessment of the bottlenecks of the existing migration protocol, obtained by progressively removing bottlenecks and measuring gains. Since indexes and data tables in RAMCloud are handled differently for migration, we need to assess them both separately.

Analysing bottlenecks in RAMCloud migration protocol: We need to motivate the need for a fast and efficient migration protocol by virtue of uncovering bottlenecks in the current state of the art. This would entail setting up the experiments showing data migration of RAMCloud using the latest available protocol and analysing and finding pain points that hamper performance. After the analysis, this needs to translate to performance benefits in the absense of disovered bottlenecks. We will analyse the upperbound of each measurements with the help of previous benchmarks showing the maximum transmission rate achieveable under a cluster of modern hardware. This will show how far we are away from operating at line rate and provide concrete evidence of limitations present in the current approach. Also, there needs to be another experiment which measures the impact of indexes in migration for change in load and skew on indexed reads in a system such as RAMCloud. We are hopeful that these will provide interesting insights on data locality.

Come up with guidelines for new protocol: Based on all our measurements, we need to come up with guidelines for a new migration protocol that will be efficient and jitter free while keeping up with the tight SLAs of RAMCloud. This new protocol will be at least a 1000x faster than the state of the art migration mechanism for in-memory stores.

Task 1	Develop a microbenchmark to measure performance of infiniband verbs	Feb 2016	Done
Task 2	Analyze performance of RDMA verbs over ibverbs library	Mar 2016	Done
Task 3	Measure Transmit performance and CPU cycles for Zero Copy and Copy Out	Apr 2016	Done
Task 4	Compare Peformance of Delta Records	May 2016	Done
Task 5	Write up and submit results to IMDM-2016	Jul 2016	Done
Task 6	Follow up measurements on Connect-IB®and hugepages	Aug 2016	Done
Task 7	Researching state of the art protocols for in-memory migration	Oct 2016	Done
Task 8	Write up Abstract	Oct 2016	Done
Task 9	Finish Writing Chapter on Introduction	Nov 2016	Done
Task 10	Analyse effects of DDIO® on memory bandwidth	Nov 2016	In Progress
Task 11	Finish the chapter on modern NICs	Nov 2016	Done
Task 12	Defend the thesis proposal	Nov 2016	In Progress
Task 13	Get uncore and offcore performance measurements from the microbenchmark	Dec 2016	TBD
Task 14	Analysing bottlenecks in RAMCloud migration protocol	Dec 2016	TBD
Task 15	Finish writing migration chapter	Jan 2017	TBD
Task 16	Come up with guidelines for new protocol	Jan 2017	TBD
Task 17	Finish writing Conclusion	Jan 2017	TBD
Task 18	Final thesis defense	Feb 2017	TBD

Table 5.1: Thesis Timeline. Pending tasks boldfaced.

REFERENCES

[1] *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.

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

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

[4] 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.

[5] 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 SIGMETRIC-S/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS '12, New York, NY, USA, 2012, ACM.

[6] 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.

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

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

[9] 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.

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

[11] 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.

[12] 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.

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

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

[15] 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.

[16] R. Huggahalli, R. Iyer, and S. Tetrick, *Direct cache access for high bandwidth network i/o*, in ACM SIGARCH Computer Architecture News, vol. 33, IEEE Computer Society, 2005, pp. 50–59.

[17] V. Jacobson, C. Leres, and S. McCanne, *pcap-packet capture library*, UNIX man page, (2001).

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

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

[26] 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.

[27] 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.

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

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

[34] 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.

[35] 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.

[36] 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.

[37] 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.

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

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

[40] 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).

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

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

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

[44] 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.

[45] 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.