Minimizing the Hidden Cost of RDMA

Philip W. Frey
Systems Department
IBM Research GmbH
8803 Rueschlikon, Switzerland
phf@zurich.ibm.com

Gustavo Alonso
Systems Group
Department of Computer Science, ETH Zurich
8092 Zurich, Switzerland
alonso@inf.ethz.ch

Abstract

Remote Direct Memory Access (RDMA) is a mechanism whereby data is moved directly between the application memory of the local and remote computer. In bypassing the operating system, RDMA significantly reduces the CPU cost of large data transfers and eliminates intermediate copying across buffers, thereby making it very attractive for implementing distributed applications. With the advent of hardware implementations of RDMA over Ethernet (iWARP), its advantages have become even more obvious. In this paper we analyze the applicability of RDMA and identify hidden costs in the setup of its interactions that, if not handled carefully, remove any performance advantage, especially in hardware implementations. From an application point of view, the major difference to TCP/IP based communication is that the buffer management has to be done explicitly by the application. Without the proper optimizations, RDMA loses all its advantages. We discuss the problem in detail, analyze what applications can profit from RDMA, present a number of optimization strategies, and show through extensive performance experiments that these optimizations make a substantial difference in the overall performance of RDMA based applications.

1. Introduction

Remote Direct Memory Access (RDMA) is a mechanism whereby data is placed directly in the application memory of a remote computer [1]. In bypassing the operating system and eliminating intermediate copying across buffers (zero copy), RDMA significantly reduces the CPU cost of large data transfers as well as the end-to-end latency, thereby making it very attractive for implementing distributed applications [2], [3], [4]. Having the CPU available for computation while receiving and sending data at a very high rate is important for various applications such as a distributed real-time analysis of large scientific experiments or high-definition video streaming to a substantial number of clients. Although the ideas behind RDMA are not new [5], [6], [7], [8], [9], [10], it has only been with the constant increase in network bandwidth that they have become a necessity not only for proprietary high-bandwidth fabrics such as Infiniband [11] but also for Ethernet based TCP/IP [12]. Early attempts to reduce the CPU load caused by high-bandwidth TCP/IP communication offloaded the entire TCP/IP stack onto the network interface card (NIC). This proved not to be enough [13]. Today, the approach most favored is a TCP/IP offload engine (TOE) plus direct memory access (DMA) from the NIC to application memory [14], [15]. In a nutshell, this is what RDMA over Ethernet (also known as *iWARP*) provides.

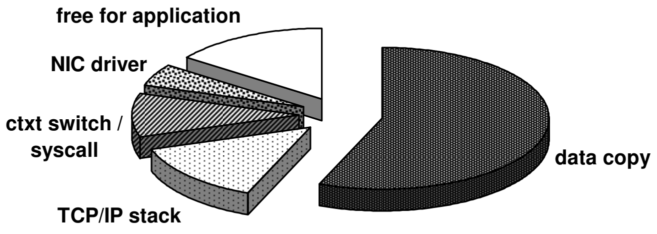

Figure 1. Typical example of TCP/IP network stack CPU load distribution.

The advantages of RDMA over plain TCP/IP offloading can be demonstrated with a simple experiment [16]. Transferring bulk data as fast as possible over a standard TCP/IP connection reveals a CPU load distribution like the one shown in Figure 1. Most CPU cycles are spent on data copying within the local host. Although offloading the TCP/IP stack onto the NIC shows some improvement, only the TOE in combination with RDMA reduces the CPU load significantly by eliminating the data copying. This is the reason why RDMA is being looked at with increasing interest as a key design element of future distributed systems [17], [18], [19]. Nevertheless, it is also facing a lot of criticism [20], [21]. To our knowledge, RDMA is not widely applied today (besides for MPI and HPC applications). We have added RDMA support to a distributed compiler and to Java RMI, and were disappointed to find that the performance benefit was negligible. In Section 5 we explain why. Those informal experiments motivated the investigation presented in this paper, which more rigorously characterizes the circumstances under which the benefit of RDMA becomes significant.

1.1. Problem Statement

One of the key differences between RDMA and operating system driven TCP/IP communication is that the application has to explicitly manage the memory segment(s) that will be used as communication buffer(s). The application has to preregister certain parts of its memory with the RDMA subsystem as source and/or destination buffers for the data transfers. During registration, the memory pages get pinned by the OS making sure they stay resident and cannot be swapped out to disk. The pinned pages are then registered with the RDMA enabled NIC (RNIC) so that it can access them using DMA operations which eliminates the need for OS callbacks and intermediate buffering during transfers. The RDMA syntax refers to these registered memory segments as *Memory Regions* (MR).

MR registration happens through the resource management path which requires kernel activity and therefore induces a delay as well as a nonnegligible CPU load. Even though the expensive data copy operations are avoided with RDMA, the explicit buffer management renders RDMA useless for applications that are not able to reuse their buffers. In this paper we show that the management of these user space communication buffers is crucial to implementing efficient RDMA communication.

We address the open question to what extent the hidden memory management costs affect the performance of RDMA. Our experiments show that, even for data transfers of moderate size, these hidden costs can completely eliminate the performance advantage of RDMA. We further present the critical parameters such as the increased connection setup time or the more expensive and complex RDMA object management that need to be considered when assessing the value of RDMA for any application.

1.2. Contributions

In this paper we describe the hidden costs of RDMA and show in detail how to design applications so that they take full advantage of RDMA. We also use our results to characterize which applications are likely to benefit from RDMA.

The contributions of this paper are three-fold. First, we provide extensive performance experiments that show the hidden costs of RDMA and compare them with the potential advantages. Second, we describe cost-effective memory management strategies for RDMA and demonstrate their feasibility and performance with experiments on the Chelsio RNIC over 10 Gigabit Ethernet. Third, we present a list of the critical parameters based on which the added value of RDMA can be assessed.

2. RDMA Background

RDMA is described in full detail in the RDMA Verbs [1] document. In here, we focus on the aspects that are relevant for our purposes in this paper.

2.1. Asynchronous Communication Interface

In contrast to the classical TCP/IP semantics, all RDMA operations are executed asynchronously. They are described by the application in terms of *Work Requests* (WR) which are posted to a *Work Queue* for asynchronous processing by the RNIC. Since posting a WR is nonblocking and since the actual data transfer described in the WR is handled by the RNIC without CPU involvement, the application can overlap communication with computation. RDMA might be of limited use for an application that cannot to profit from this.

2.2. RDMA Data Transfer Operations

The data transfer operations offered by RDMA are *Send*, *Receive*, *RDMA Read* and *RDMA Write*.

The Send and Receive operations are called *two-sided* because the applications on both sides are actively involved in the data transfer. The sending application specifies the buffer from which the data to be sent must be taken by posting a Send WR and the receiving application at the other side decides where to place the inbound data by posting a Receive WR beforehand containing a descriptor of the destination buffer. The other two operations are called *one-sided*, meaning that only the application layer of the host issuing the operation is actively involved in the data transfer. At the other host, the operation is handled entirely by the RNIC without notifying the application. The RDMA Write operation copies data from a local MR into a remote one whereas the RDMA Read does the opposite. Asynchronous notification about completion of the local work request can be demanded for all operations.

The following list presents peculiarities of the above RDMA operations which limit their applicability in certain cases:

1) they are executable only on explicitly preregistered buffers
2) the receiver needs to know the size of the inbound data in order to have an appropriate target buffer ready
3) one-sided operations require knowledge of the remote buffer which necessitates a prior advertisement
4) reregistration of a buffer requires a readvertisement inducing protocol delay
5) the remote side of a one-sided operation cannot implicitly be notified of the completion of an operation

2.3. Explicit Buffer Management

Applications based on TCP/IP assume implicit communication buffers provided by the OS. The flexibility of that approach comes with the major drawback of requiring intermediate buffer copies, which induce a significant CPU and memory bus overhead as shown in Figure 1. The RDMA model on the other hand requires the application developer to allocate his communication buffers (or Memory Regions, MRs) explicitly and to register them with the RNIC for hardware accelerated direct data placement using DMA. Once registered, an MR has a fixed size which can only be changed by deregistering it and thereafter registering the buffer as a new MR. Since the registered MRs block the underlying memory for other applications, they should be deregistered when they are no longer needed. As we will see later in this paper, MR (de-)registration induces a significant overhead. Applications that cannot reuse their communication buffer(s) therefore lose a significant performance advantage.

3. RDMA Cost Analysis

When looking at a network data transfer from a high-level perspective, the data path can be divided into two parts: Firstly, the data is copied locally between the application buffer and the wire (part A in Figure 2). Secondly, it is transferred across the network (part B) to the remote host.

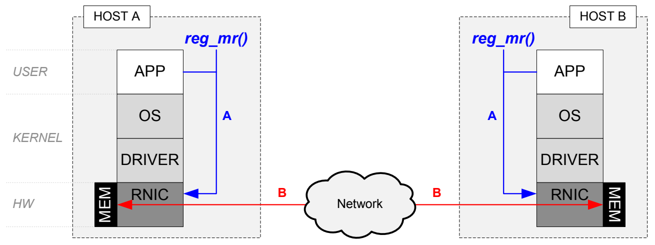

Figure 2. Network data transfer.

The performance advantages of RDMA appear only after the path (A-B-A of Figure 2) is established and does not need to be changed anymore. Hence, for an application to profit from RDMA, it has to reuse its buffers during operation. Such reuse is only possible if the application can be designed to output all its data directly to that fixed user virtual memory address interval where the MR is situated. Furthermore the data set must always be of the same size or else memory is wasted. If this is not possible, the application must either copy the data locally into an existing MR (Application X in Figure 3) or register a new MR on the data (Application Y). Our experiments in this section indicate that only a combined approach is able to keep the overhead low.

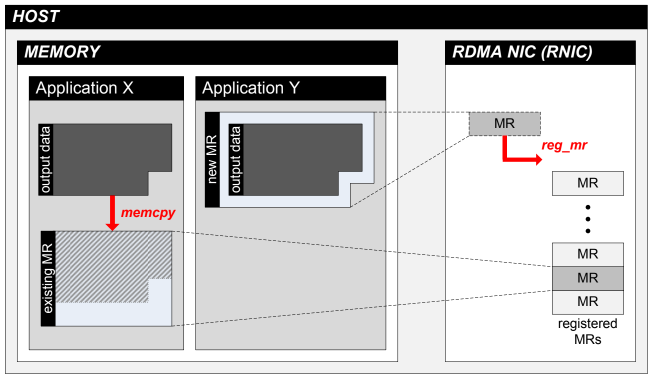

Figure 3. Prepare output data for RDMA.

3.1. Test Environment

Our testbed consists of an IBM BladeCenter containing HS21 BladeServers. Each of them is equipped with a quad core Intel Xeon CPU running at 2.33 GHz, 32 KB L1 data cache and 32 KB L1 instruction cache, 4 MB unified L2 cache and 8 GB of main memory (see Figure 4).

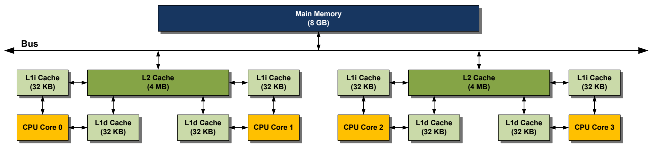

Figure 4. Intel Xeon cache layout.

RDMA hardware support is provided by Chelsio T3 RNICs (S320EM-BCH) which offer full TCP/IP offloading (TOE) and iWARP RDMA support. The RNICs are interconnected through a Nortel 10 Gb Ethernet Switch Module. The BladeServers are running Fedora Core 9 with a 2.6.24 vanilla kernel. The OpenFabrics Enterprise Distribution (OFED v1.3.1) software stack [22] serves as OS interface to the RDMA subsystem.

3.2. RDMA Setup

Before remote DMA access is even possible, an elaborate connection setup followed by the creation of a number of RDMA objects such as the *Protection Domain*(PD), *Queue Pair*(QP), *Completion Queue*(CQ) and others is necessary. In order to assess the overhead compared to a simple TCP handshake, we have measured the time-to-first-byte — the time it takes for a client to connect to a server and send a byte of payload. It includes the connection setup, the creation of the required resources, and the transfer of the single byte. On our link (round-trip time of 0.025 ms) we have found a time-to-first-byte of 202 ms for RDMA and a mere 0.1 ms for TCP/IP. A factor of a thousand. This clearly shows that RDMA is a bad fit for any application using many short-lived connections (e.g., a webserver) due to the huge connection setup cost.

3.3. Memory Region (De-)Registration

In our next experiment, we have identified the cost for registering and deregistering RDMA MRs of different sizes.

```
for size in {min ... max} do
    buf = mmap(size);
    clock_gettime(CLOCK_REALTIME, &t_start);
    ibv_reg_mr(pd, buf, size, access);
    clock_gettime(CLOCK_REALTIME, &t_stop);
end for
```

Our benchmark core is sketched as pseudo code in the above listing. Reading the clock induces an overhead of 0.14 μs which is negligible unless stated otherwise. The Memory Regions are registered on line 5 in the above listing. In our RDMA subsystem, they are managed by the user space library of OpenFabrics in a balanced search tree (red-black tree). When calling `ibv_reg_mr()`, a new MR reference is created and added to the tree. After that, the code traps into privileged mode where the user pages are mapped into the kernel virtual address space. In the next step, the underlying physical pages are allocated and pinned in main memory as necessary. Last, the page addresses are translated into bus addresses which are then registered with the RNIC for DMA.

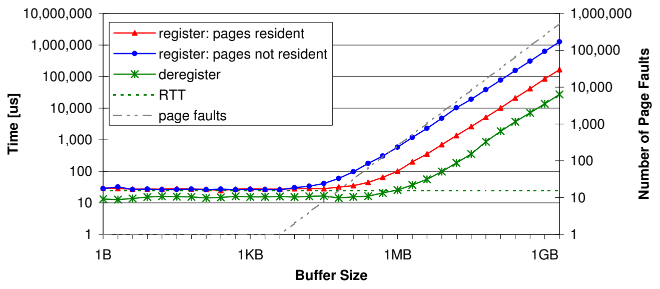

Figure 5. MR (de-)registration costs.

Figure 5 shows the time required for registering Memory Regions of sizes between 1 B and 2 GB on a doubly logarithmic scale. As can be seen, the MR registration cost is constant up to and including 4 KB (page size) and increases linearly with the size of the MR (number of pinned pages) after that. The constant overhead for small buffers results from the rather long code path described above. It is a coincidence of our test system that the constant cost is almost equal to the round-trip time — MR registration does not involve network communication. After 4 KB, the dominating costs are page table lookups, walking and updating the involved data structures, translating the addresses and the like.

Since all the pages of the new MR must be pinned, they first need to be resident in physical memory. In the worst case, this means that each page of the MR causes a page fault (marked with circles in Figure 5). To find out how much weight these page faults carry in the context of MR registration, we have conducted a second test series with MRs whose underlying pages were already resident in physical memory before registration. Beyond the size of a page, registering Memory Regions on pages which are not resident becomes significantly more expensive than registering an MR on resident pages. At a size of about 2 MB, the difference reaches almost an order of magnitude (we are not considering the noncompetitive case where the pages have been swapped out to disk).

Huge pages can be used to reduce the total number of pages required to back a memory region of a given size. Since the overhead of registering a memory region increases with the number of pages involved, we expect to see a cost reduction when using pages of size 2 MB rather than the standard 4 KB. Our experiments have shown a reduction of the registration cost of up to 40% as compared to the standard page size. This is only true for memory regions which are larger than the size of a huge page, of course.

Deregistration is a slightly simpler, inverse version of the registration code path. Since the pages of registered MRs are always pinned, page residency is not an issue here. As expected, Figure 5 shows that deregistration is significantly faster than registration (for all sizes). The explanation for this is that there is no need for address translations and that unpinning the pages is cheaper than pinning them. Up to and including 128 KB, the deregistration time is constant at ~15 μs. For larger MRs, the time increases linearly with the buffer size as well. As pointed out before, deregistration of MRs which are no longer used is vital for system resource management because the underlying physical memory is pinned and not available for other processes.

We conclude that (de-)registering MRs induces a nonnegligible hidden cost in terms of CPU load as well as increased delay.

3.4. Memory Copying

Having an application that cannot steer its output into an existing MR (e.g., coming from a mapped file) forces us to either register a new MR on the data (see previous section for the cost analysis) or copy the data into an existing MR if we want to use RDMA. We now consider the second option (copying). For that, we measure the pure copy performance of `memcpy()` on the same buffers as before, once with the pages resident in main memory and once causing page faults. Figure 6 displays the results of our experiments where the buffer size indicates the amount of data being copied.

Since `memcpy()` is a highly optimized function with a short code path, it has a very low overhead for buffers smaller than a page — the delay measured is actually dominated by the timer here. In our setup, up to 2 KB can be

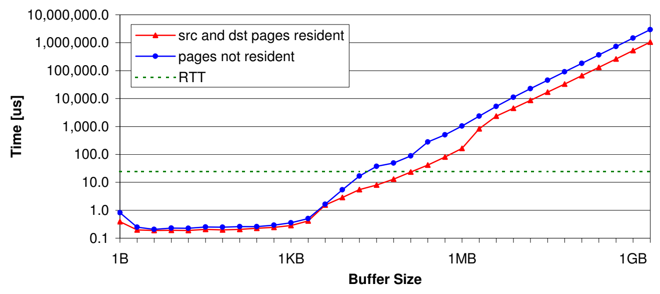

Figure 6. Memory copy costs.

copied in under 1 μs which is about two orders of magnitude faster than MR registration. For buffers larger than 2 KB we see a picture which is similar to MR registration: The time increases linearly to the amount of bytes copied. As we will explore in Section 4, copying very large buffers is significantly slower than registering them as new MRs. Page residency is also important for `memcpy()` but the performance improvement on large buffers is not as dramatic as in the case of MR registration.

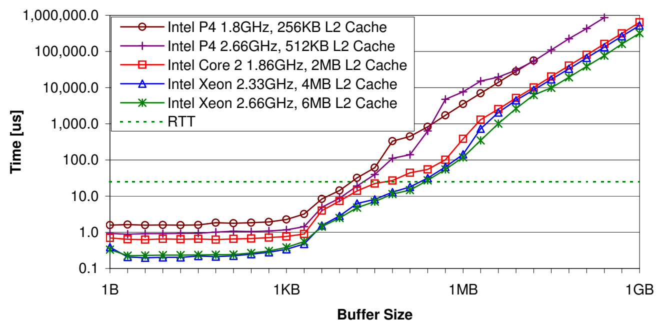

Figure 7. Memory copy on various systems.

Since `memcpy()` does not involve the RNIC, we expect its performance to be strongly dependent on the CPU as well as on the the page size and cache sizes. Figure 7 confirms that for various systems. On an older Intel P4 1.8GHz, the copy delay is much larger than on a more recent Intel Xeon 2.66GHz for all buffer sizes.

4. Optimization Strategies

Based on our findings we now present several optimization strategies that reduce the overhead of RDMA communication.

4.1. Respect the Critical Buffer Size

In the case where we cannot steer our application output into an existing MR, we must either register the application output buffer as a new MR by means of `ibv_reg_mr()` or copy its content into an existing MR of appropriate size using `memcpy()` (see Section 3). Both approaches were used out of the box (i.e., no on-machine tuning was performed).

Figure 8 compares the delay of these two options and shows that MR registration has a significant overhead for small Memory Regions (<256 KB) as compared to `memcpy()` (up to several orders of magnitude!). After 256 KB however, it is much more efficient to register a new MR than it is to copy the application output. At about 4 MB (size of L2 cache), the delay difference reaches almost an order of magnitude in favor of the registration. Furthermore,

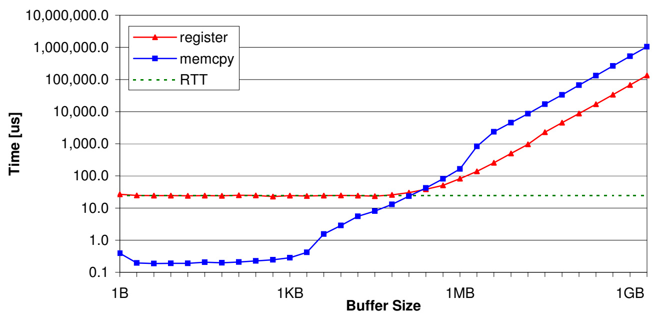

Figure 8. MR registration VS. memcpy with resident pages.

copying always induces 100% CPU load, whereas a reregistration (deregistration followed by registration) induces between 60% and 75% due to the additional hardware I/O.

This leads to our first optimization: If buffer reuse is not possible but RDMA is still desired, buffers smaller than the critical size ought to be copied into existing MRs and larger buffers must be reregistered. Copying a large (>2 MB) buffer is not only a lot slower than registering a new MR on it but it also consumes all the available CPU cycles and induces a much higher load on the memory bus. In terms of cache pollution, registration is also preferred because it does not touch the data. As we will discuss later, this optimization matches typical communication buffer usage.

The important question now is what determines this critical size where registration outperforms copying. The answer is two-fold. The first aspect is the CPU performance: Since `memcpy()` is more CPU intensive than `ibv_reg_mr()`, running the above experiments on a slower CPU decreases the copy performance compared to registration, which results in a shift of the critical size towards smaller buffers. The second aspect which is even more serious is the page residency. Figure 9 shows that the registration of nonresident pages outperforms copying already for buffers larger than 32 KB which is a shift in favor of `ibv_reg_mr()` by a factor of 8. Using huge pages does not have a significant influence on the critical size as the cost reduction for `memcpy()` is about the same as for `ibv_reg_mr()`.

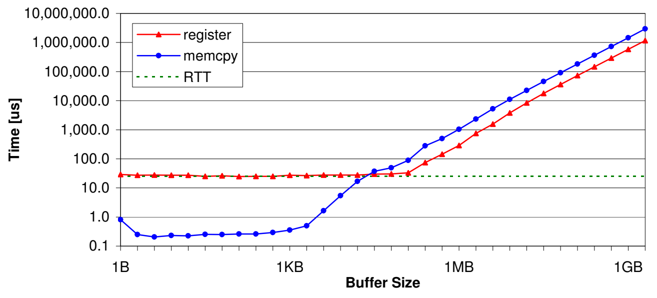

Figure 9. MR registration VS. memcpy with nonresident pages.

4.2. Overlap Buffer Management with Communication

The second optimization is to amortize the MR (de-)registration cost by overlapping it with waiting for a message from the remote host. The dotted horizontal line in the previous figures marks the round-trip time (RTT) between two RNICs through our 10 GbE fabric ($\sim$25 μs). We can register a buffer on resident pages of up to 64 KB (or 8 KB in case of nonresident pages) within the RTT without inducing an overall protocol delay. MRs of up to 1 MB can be deregistered during RTT. The 64 KB for registration and 1 MB for deregistration are pessimistic lower bounds since our RTT does not take into account any application processing delay which typical real world protocols have.

4.3. Register Buffer on Resident Pages

Our third optimization is to design RDMA-based applications such that they register their MRs shortly after having touched or created the data — especially if they encompass more than one page — which reduces the expensive page fault processing during registration and therefore reduces the registration time by up to an order of magnitude. This also allows larger MRs to be registered during RTT.

4.4. Parallel Buffer Registration and Applicability

Today, most machines are equipped with several CPUs and/or several cores. The question arises whether MR registration can benefit from that. Figure 10 depicts the latency for registering 4 MRs in parallel (one on each core). Again, we look at the case where the pages are resident (marked with triangles) and compare it with the one causing page faults. The dotted lines on top serve as reference showing the delay for sequential registration of the MRs on a single core and the solid lines at the bottom reflect the cost for registering a single MR. When comparing sequential with parallel MR registration, it is evident that the parallel registration can only profit from multiple cores when the pages are resident. Otherwise it is almost as expensive to register four MRs in parallel as it is to register them one after the other. We have found that the reason for this is the limited overall page fault rate of the system. Another motivation for the application to make sure the pages are resident before registration.

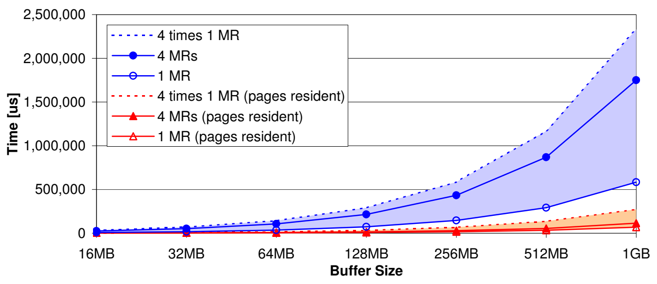

Figure 10. Parallel MR registration.

4.5. Suitability of the Optimizations

We now show why the proposed buffer management optimizations fit very well with regard to real world protocols.

Large buffers typically contain the actual application data and are highly variable in size. Registering these large buffers as MRs, renders an equally large amount of the physically available main memory unusable for other processes due to the pinning requirement of RDMA buffers. Hence they should be deregistered when they are no longer needed. This matches our finding of reregistration being cheaper for large MRs than copying.

Small buffers on the other hand are typically used for exchanging control messages of constant size. By keeping them registered as MRs and refilling them with `memcpy()`, we induce a significantly lower delay and do not waste much memory even if they are not used at the moment. Since communicating hosts are often waiting for some kind of control messages from their peers before they can proceed with the protocol, it is vital that the delay for shipping these messages is low in order to get an efficient overall data exchange. An alternative approach is to transport these small control messages using plain TCP/IP but that results in the loss of packet ordering guarantees because of the extra socket.

Our experimental evaluation demonstrates clearly that a straight-forward explicit Memory Region management can degrade the overall application performance dramatically not only in terms of latency but also in terms of induced CPU load, cache pollution etc. For a good buffer management strategy — whenever buffer reuse is not an option — it is vital to respect the critical size where reregistration becomes more efficient. As we have shown, reregistering or copying the data according to the critical size results in a performance gain of up to several orders of magnitude in both directions.

Critical Parameter	Example App.	RDMA Suitable
CPU intensive	HPC [18]	yes
large data transfers	DB recovery	yes
long lasting connections	DB logging	yes
data in main memory	HPC	yes
small data size variation	DB logging	yes
reuse buffers	streaming	yes
use async interface	HPC	yes
make use of scatter-gather list	HPC	yes
use one-sided operations	streaming	yes
time-to-first-byte	DNS	no
short-lived connections	webserver [17]	no
unpredictable msg size	RPC [23]	no
indirect buffer	Java apps	no

Table 1. RDMA Suitability Assessment

5. When is RDMA beneficial?

RDMA over Ethernet offers a lower latency as well as a higher throughput than TCP/IP and even complements that with a close to idle CPU that TCP/IP cannot provide. Figure 11 represents an ideal scenario of an application that can reuse its registered buffers extensively and therefore does not have to face the aforementioned costs. For this experiment, we transmit 1 GB of payload using consecutive RDMA Write operations. Figure 11 shows that RDMA performs efficiently when transmitting messages larger than 4 KB where the communication link is saturated and the CPU is close to idle.

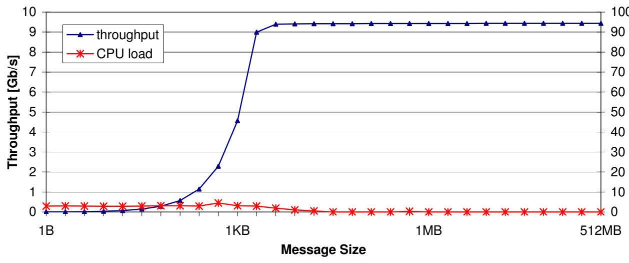

Figure 11. RDMA Write Throughput.

Nevertheless there are applications that perform better when using plain TCP/IP. In Section 3 we have shown that RDMA has hidden costs such as the connection setup or the explicit buffer management. In Table 1, we address the open question as to when RDMA offers a benefit over traditional TCP/IP by listing the critical parameters and giving an example application for each parameter.

We conclude that an application profiting from full RDMA performance has the following characteristics: It lives in an environment where there is little or no churn (negligible connection setup costs), it can reuse its buffers (MRs) extensively and transfers a lot of data. Furthermore, it is able to overlap communication with computation (asynchronous interface) and can make use of one-sided operations (remote computer does not need to be notified; few synchronization points in the protocol).

On the other hand, an application that faces a lot of churn, operates on unpredictable, highly varying buffer sizes and depends on a short time-to-first-byte (e.g., RPC) will perform much better using plain TCP/IP.

6. Related Work

The fact that RDMA requires an explicit communication buffer management has been identified as a drawback before. A detailed analysis of the MR registration cost in the Mellanox Infiniband software stack can be found in [24]. Even though it is not iWARP and not based on the OpenFabrics RDMA stack, the issues are similar. Unlike in this this paper, however, the only optimization put forward in [24] is to use large pages which results in a cost reduction of 15%. Arbitrarily increasing the page size is not suitable in many contexts and has nonnegligible side effects. Our proposed solution is less intrusive and over all more effective.

The idea to deregister MRs lazily is proposed in [25]. Upon receiving a deregistration request from the application, the subsystem does not actually remove the registered segment but keeps it in a cache to reduce future MR registration costs. The memory is unpinned only when the cache exceeds a certain limit. Most popular RDMA subsystems as well as the Linux kernel do not support this yet. Even though it sounds like a good idea, kernel support is required to keep the cache consistent since the user has access to a variety of operating system calls to alter the memory layout and thereby potentially destroying earlier cached registrations. A detailed description of the issues can be found in [26]. Our proposed optimizations do not require a modified kernel and are therefore more generally applicable.

The performance advantage of RDMA is marginal when issuing lots of small I/Os as the per-I/O cost prevails over the per-byte cost (see Figure 11). In the context of RPC over RDMA, an optimistic RDMA is proposed in [23] where data is sent assuming that there is a target buffer ready at the receiver and an exception is returned if it is not the case. While this optimization shows an improved performance for small RPC I/Os due to the reduced response time, it breaks compliance with the RDMA Verbs on which the industry has agreed.

7. Conclusions

Even though RDMA offers zero copy and kernel bypassing for very efficient data transfers between remote hosts in terms of CPU load and memory bus bandwidth, it has hidden costs. Therefore RDMA is not equally well suited for all applications. We have argued why a low-overhead communication buffer management is key to the efficient use of RDMA and have presented a number of optimization strategies. For the case where the buffers cannot be reused, we have shown how the communication delay is reduced by up to a couple orders of magnitude if the critical buffer size is respected. Large buffers must be reregistered and small buffers refilled. The result is a significantly lower latency, less CPU load and reduced waste of memory. In that context, we have pointed out the importance of registering MRs on pages which are resident in main memory. Finally we have specified the application parameters which must be considered when assessing the use of RDMA for a concrete application.

Acknowledgment

We would like to thank Jon Howell for his thorough feedback on this work.

References

[1] J. Hilland, P. Culley, J. Pinkerton, and R. Recio, "RDMA Protocol Verbs Specification, Version 1.0." [Online]. Available: http://www.rdmaconsortium.org/home/draft-hilland-iwarp-verbs-v1.0-RDMAC.pdf

[2] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda, "High performance RDMA-based MPI implementation over InfiniBand," in *Proceedings of the 17th Annual International Conference on Supercomputing*, 2003, pp. 295–304.

[3] R. Noronha, L. Chai, T. Talpey, and D. K. Panda, "Designing NFS with RDMA for security, performance and scalability," in *Proceedings of the 2007 International Conference on Parallel Processing*, 2007, p. 49.

[4] M. Ko, M. Chadalapaka, J. Hufferd, U. Elzur, H. Shah, and P. Thaler, "Internet Small Computer System Interface (iSCSI) Extensions for Remote Direct Memory Access (RDMA)," 2007.

[5] B. S. Davie, "A host-network interface architecture for ATM," *ACM SIGCOMM Computer Communication Review*, vol. 21, pp. 307–315, 1991.

[6] C. Dalton, G. Watson, D. Banks, C. Calamvokis, A. Edwards, and J. Lumley, "Afterburner," *IEEE Network*, vol. 7, pp. 36–43, 1993.

[7] J. M. Smith, C. Brendan, and S. Traw, "Giving applications access to Gb/s networking," *IEEE Network*, vol. 7, pp. 44–52, 1993.

[8] P. Druschel and L. L. Peterson, "Fbufs: A high-bandwidth cross-domain transfer facility," in *Proceedings of the 14th ACM symposium on Operating Systems Principles*, 1993, pp. 189–202.

[9] Compaq Computer Corp., Intel Corp., Microsoft Corp., "Virtual interface architecture specification." [Online]. Available: http://www.intel.com/intelpress/chapter-via.pdf

[10] J. Chase, A. Gallatin, and K. Yocum, "End system optimizations for high-speed TCP," *IEEE Communications Magazine*, vol. 39, pp. 68–74, 2001.

[11] InfiniBand Trade Association, "InfiniBand architecture specification." [Online]. Available: http://www.infinibandta.org

[12] D. D. Clark, V. Jacobson, J. Romkey, and H. Salwen, "An analysis of TCP processing overhead," *IEEE Communications Magazine*, vol. 27, pp. 23–29, 1989.

[13] J. C. Mogul, "TCP offload is a dumb idea whose time has come," in *Proceedings of the 9th conference on Hot Topics in Operating Systems*, 2003, pp. 5–5.

[14] A. Romanow, J. Mogul, T. Talpey, and S. Bailey, "Remote Direct Memory Access (RDMA) over IP Problem Statement," 2005.

[15] P. Buonadonna and D. Culler, "Queue pair IP: A hybrid architecture for system area networks," *ACM SIGARCH Computer Architecture News*, vol. 30, pp. 247–256, 2002.

[16] F. Neeser and B. Metzler, "High-speed network I/O based on RDMA." [Online]. Available: http://www.zurich.ibm.com/sys/servers/rdma_hispeed.html

[17] D. Dalessandro and P. Wyckoff, "Accelerating web protocols using RDMA," in *Proceedings of the 6th IEEE International Symposium on Network Computingand Applications*, 2007.

[18] S. Narravula, H. Subramoni, P. Lai, R. Noronha, and D. Panda, "Performance of HPC middleware over InfiniBand WAN," in *Proceedings of the 37th International Conference on Parallel Processing*, 2008, pp. 304–311.

[19] S. Pakin, "Receiver-initiated message passing over RDMA networks," in *Proceddings of the IEEE International Symposium on Parallel and Distributed Processing*, 2008, pp. 1–12.

[20] P. Geoffray, "A critique of RDMA." [Online]. Available: http://www.hpcwire.com/features/17886984.html

[21] K. Magoutis, M. Seltzer, and E. Gabber, "The case against user-level networking," in *Third Workshop on Novel Uses of System Area Networks*, 2004.

[22] OpenFabrics Alliance, "OpenFabrics Enterprise Distribution." [Online]. Available: http://www.openfabrics.org/

[23] K. Magoutis, S. Addetia, R. Fedorova, and M. I. Seltzer, "Making the most out of direct-access network attached storage," in *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, 2003.

[24] F. Mietke, R. Rex, R. Baumgartl, T. Mehlan, T. Hoefler, and W. Rehm, "Analysis of the memory registration process in the mellanox InfiniBand software stack," in *Proceedings of the 12th International Euro-Par Conference*, 2006, pp. 124–133.

[25] H. Tezuka, F. O'Carroll, and A. Hori, "Pin-down cache: A virtual memory management technique for zero-copy communication," in *Proceedings of the 12th International Parallel Processing Symposium*, 1998.

[26] P. Wyckoff and J. Wu, "Memory registration caching correctness," in *Proceedings of the Fifth IEEE International Symposium on Cluster Computing and the Grid*, 2005.