On Limitations of Network Acceleration

Animesh Trivedi
IBM Research, Zurich
atr@zurich.ibm.com

Bernard Metzler
IBM Research, Zurich
bmt@zurich.ibm.com

Patrick Stuedi
IBM Research, Zurich
stu@zurich.ibm.com

Thomas R. Gross
ETH Zurich, Switzerland
trg@inf.ethz.ch

Abstract

The performance of large-scale data-intensive applications running on thousands of machines depends considerably on the performance of the network. To deliver better application performance on rapidly evolving high-bandwidth, low-latency interconnects, researchers have proposed the use of network accelerator devices. However, despite the initial enthusiasm, translating network accelerator's capabilities into high application performance remains a challenging issue.

In this paper, we describe our experience and discuss issues that we uncover with network acceleration using Remote Direct Memory Access (RDMA) capable network controllers (RNICs). RNICs offload the complete packet processing into network controllers, and provide direct user-space access to the networking hardware. Our analysis shows that multiple (un)related factors significantly influence the performance gains for the end-application. We identify factors that span the whole stack, ranging from low-level architectural issues (cache and DMA interaction, hardware pre-fetching) to the high-level application parameters (buffer size, access pattern). We discuss implications of our findings upon application performance and the future of integration of network acceleration technology within the systems.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Performance attributes; C.2.4 [**Distributed Systems**]: Client/server

Keywords

Network Acceleration; Cache Coherence; Performance;

1. INTRODUCTION

Big Data applications generate, store, and analyze large volumes of data everyday. Examples of these applications are instant business analytics, real-time distributed graph processing, data-intensive scientific computing (e.g., the LHC experiment at CERN) and associated supporting application frameworks [2, 10, 13]. These data-intensive applications run in parallel on thousands of machines inside a large, heavily networked environment such as data centers. Consequently, the performance of these applications depends considerably on the performance of the underlying network.

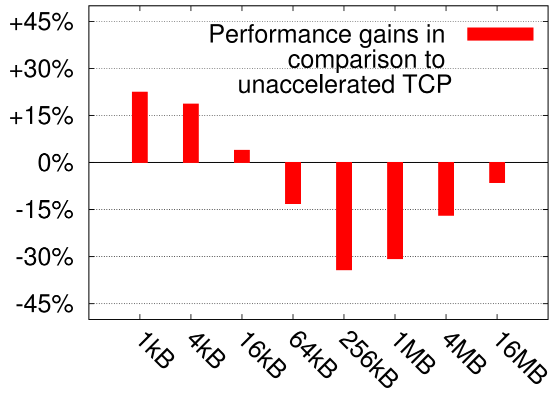

Figure 1: Performance gains of using network acceleration in comparison to the unaccelerated TCP stack. Positive numbers on the y-axis represent application performance gains. Use of a network accelerator device does not always translate into a better application performance. See Section 2.1 for details.

To improve the performance of these applications on rapidly evolving high-bandwidth (10, 40, 100 Gbps) low-latency (5-10 μsecs) interconnects, researchers have proposed using network accelerator devices [24, 14]. Remote Direct Memory Access (RDMA) is a network acceleration technology that offloads the packet processing into the network controller and provides safe userspace access to networking resources. Data is transmitted and received directly from userspace application buffers. Together, these properties provide high-bandwidth low-latency data transfer between systems with a very low CPU footprint. Furthermore, RDMA is also shown to be power efficient [12].

However, translating the advantages of RDMA based network acceleration into application-level performance is challenging. In this paper we report on our experiences when using RDMA acceleration for data transfers. We report on a simple request-response experiment in a server-client configuration. For every client request, the server *prepares* a

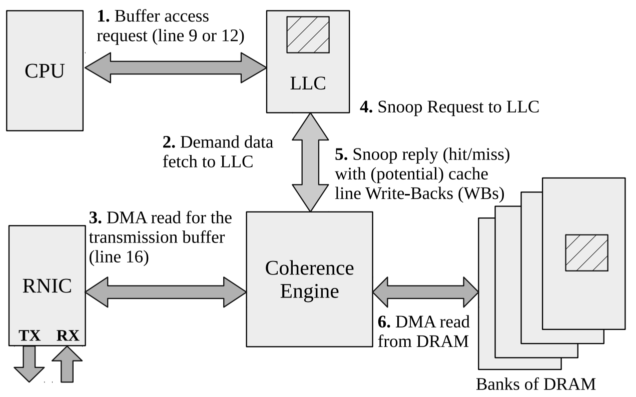
CPU
1. Buffer access request (line 9 or 12)
LLC
4. Snoop Request to LLC
2. Demand data fetch to LLC
5. Snoop reply (hit/miss) with (potential) cache line Write-Backs (WBs)
3. DMA read for the transmission buffer (line 16)
RNIC
TX
RX
Coherence Engine
6. DMA read from DRAM
Banks of DRAM

Figure 2: Interaction sequence among the CPU core, LLC, DMA access, and Coherence Engine. Line numbers refer to the code listing in Figure 3.

```
1. char dummy_buff[BUF_SZ], tx_buff[BUF_SZ];
2. /* Until timeout, keep receiving requests */
3. while(!time_out){
4.   /* Receive the request from the client */
5.   recv_request();
6.   for(i=0; i<BUF_SZ; i+=CACHE_LINE_SZ){
7.#if SCAN_MODE == TOUCH
8.     /* scan the transmission buffer */
9.     scan(tx_buff[i]);
10.#elif SCAN_MODE == NO_TOUCH
11.    /* else, scan the dummy buffer */
12.    scan(dummy_buff[i]);
13.#endif
14.  }
15.  /* always send the transmission buffer */
16.  send_buffer(tx_buff, BUF_SZ);
17. }
```

Figure 3: Server-side execution logic.

response buffer and sends it out on the network back to the client. We use RDMA and the unaccelerated TCP stack in Linux for data transfers and compare their performances. Figure 1 shows the performance gains (seen by the client as improved request completion time) on the y-axis for different sizes of the response buffer (on the x-axis). Positive numbers on the y-axis represent performance gains for the end-application using RDMA. Our experiment suggests:

(a) Networked applications can even lose performance when using network accelerators in particular circumstances. The performance implications of complex interactions among sophisticated CPU cores, last-level caches, and low-latency network controllers are highly machine specific and are hard to predict. For the same application running on different generations of CPUs and NICs one can observe a wide-range of performance fluctuations, including performance loss.

(b) Modern network latencies are getting closer and comparable to architectural overheads. The overhead of coherence maintenance, cache-misses, DRAM access, and CPU stalls significantly influence the performance of an end-application operating in a low-latency network environment. Although the exact overhead is workload specific it is affected by a number of characteristics such as buffer size, access pattern etc.

In this paper we identify performance factors that span the whole stack, ranging from low-level architectural issues (cache and DMA interaction, hardware pre-fetching) to the high-level application parameters (buffer size, access pattern) and attribute costs to them on our systems. Although our findings are RDMA and CPU specific, they are illustrative of a growing confusion about performance when using network accelerator devices. This paper is meant to open a dialogue about emerging high-performance interconnects, network accelerators, and their impact on performance.

2. EXPERIMENTS WITH RDMA

Our analysis reveals that application-level latencies seen by the client are dominated by the buffer preparation step at the server. Hence, we further investigate the interaction among various entities involved in the buffer preparation and transmission steps, namely CPU, last level cache (LLC), and DMA access to DRAM. Figure 2 illustrates the sequence of interaction among the entities on an I/O coherent architecture such as `x86`.

We start by designing a controlled request-response experiment between the server and the client (as outlined in Section 1). The client constantly sends a request to the server in a tight loop without any pipelining. Upon receiving the request, the server prepares a buffer and transmits data in the buffer back to the client. The size of the buffer is variable. In our controlled setup, the *preparation* is a simple scan operation on the buffer. In a real-world application, the preparation step can involve reading data from a persistent storage and then copying it into the buffer. Figure 3 shows the code which we implement within the `netperf` benchmark framework [16].

We now explain the preparation step in greater detail. Different buffer preparation configurations give us flexibility to analyze cache and snoop protocols in a controlled environment while keeping a uniform CPU load. On the server side, the *preparation* step has two modes: **Touch** and **NoTouch**. In the Touch mode, data in a transmission buffer is scanned using a `for` loop. In the NoTouch mode, a similar scan is done on a dummy buffer. The two buffers, transmission and dummy, are identical but only the transmission buffer is transmitted on the network (see lines 15-16 in Figure 3).

Furthermore, the scan can be of two types: **Read Scan** or **Write Scan**. A Write Scan emulates a reader-writer sharing scenario, where the CPU writes and the network controller reads the buffer. A Read Scan represents a read-

	NoTouch	**Touch**
Write Scan	Modified cache lines (M) from the dummy buffer.	Modified cache lines (M) from the transmission buffer.
Read Scan	Exclusive cache lines (E) from the dummy buffer.	Exclusive cache lines (E) from the transmission buffer.

Table 1: Content of last-level cache depending on the mode and the scan type. Modified(M) and Exclusive(E) cache line status represent the MESIF protocol states.

	Intel Xeon E7520
CPU cores	4×1.8GHz
QPI speed	4.8 GT/sec
L1 cache	64kB, 2.1nsec, 8-ways associativity
L2 cache	256kB, 5.3nsec, 8-ways associativity
LLC	18MB, 22.7nsec, 24-ways associativity
LLC type	Inclusive of L1 and L2 caches
Cache line size	64 Bytes
DRAM latency	131nsec
Prefetching	Next-line Prefetcher, enabled

Table 2: Architectural properties and configuration of Intel Nehalem-EX Xeon E7520 CPU.

read sharing of the buffer. The scan access on the buffers (either transmission or dummy) brings the associated cache lines into the LLC. To maintain the I/O coherence, transmission of the transmission buffer generates snoop requests for LLC (see Figure 2). Table 1 summarizes the LLC content for different combinations of the modes and the scan types.

2.1 Experiment Methodology and Hardware

We measure the single request completion time (the time between issuing a request and receiving the complete response buffer) at the client as the key performance metric. We use two network transport implementations for the buffer transmission - unaccelerated Linux in-kernel TCP/IP and an accelerated RDMA stack. The Linux stack runs on the host CPU together with the benchmark application. We calculate performance gains by comparing the request serving time between the two stacks. TCP performance is measured under a similar setup by using a modified `TCP_RR` test from the `netperf` test suit. Previously shown Figure 1 compares the performance of the RDMA accelerated stack and the in-kernel TCP/IP stack under the Touch/Write Scan configuration for different response buffer sizes.

We perform our experiments on two identical IBM system x3690 X5 machines containing the Intel X58 chipset with Intel Xeon Nehalem-EX E7520 CPUs. Table 2 summarizes architectural parameters for the CPU. Chelsio Terminator4 (T4) RDMA-capable Network Interface Controllers (RNIC) are used for network acceleration on the 10Gbps Ethernet. However, we repeated our experiments with Intel NetEffect network accelerator adapters and found no significant deviations in our findings. We use Linux `perf` [1] measurement framework to measure the global coherence events as documented in the Intel manual [8]. Linux kernel version `3.7.0` is used in all experiments.

All experiments last 60 seconds, and are repeated three times. We omit reporting variance because reported performance numbers have less than 5% standard deviation between the three runs. To avoid any multi-core coherence interference, all cores except Core0 are switched off. Core0 and RNIC are the only two entities in the system sharing the access to the DRAM. Pre-fetching is enabled for all experiments except for Section 3.3.

3. ANALYSIS AND RESULTS

In this section we present our results regarding various architectural overheads, and attribute costs to them on our systems. All data transfers in this section use RDMA.

	NoTouch	Touch	Loss
Write	300 μseconds	470 μseconds	56.6%
Read	275 μseconds	315 μseconds	14.5%

Table 3: Latency numbers for a complete request-response loop, measured at the client side for the different modes and scans on the server. The response buffer size is 256kB.

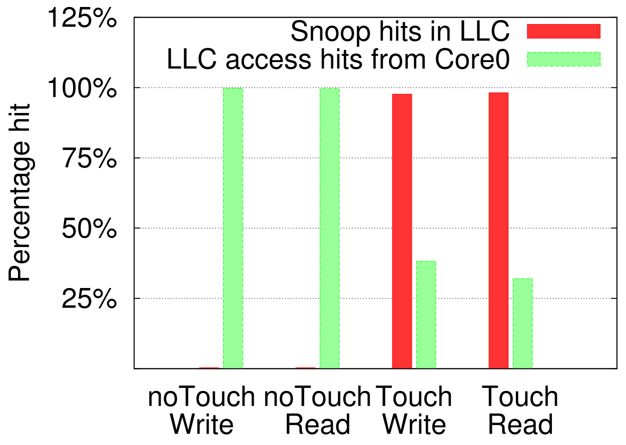

Figure 4: The snoop and the LLC hit rates.

3.1 DMA Access and Cache Coherence

We start by analyzing the large performance penalties for touching the transmission buffer (as any real-world application would do). As shown in Table 3, the Touch mode access results in a 56% and 14% drop in performance for Write and Read Scans, respectively. In our experiment, the server always transmits the transmission buffer. To maintain coherence, snoop requests for the transmission buffer are generated when the DMA engine on the RNIC accesses the DRAM (see steps 3 and 4 in Figure 2). There are two possible outcomes of a snoop request (a) a snoop miss, when the LLC does not contain snooped addresses, (b) a snoop hit, when the LLC contains snooped addresses. As the snoop requests are always generated for the transmission buffer, the Touch access has a high snoop hit rate. In the case of a snoop hit the coherence engine must take appropriate actions to ensure I/O coherence. Modified cache lines are evicted and written back (WB) to the DRAM to ensure that DMA access reads the latest content. In the case of clean Exclusive cache lines nothing should be done. However, as we illustrate, the exact actions are implementation specific.

The performance drop for the Write Scan can be attributed to cache lines eviction and costly WBs to DRAM. However, unexpectedly the Read Scan on the transmission buffer also results in a performance drop. This behavior leads to a further investigation about snoop and coherence interaction. We measure the snoop hit rate by counting the snoop requests that hit the LLC. Similarly, we measure the LLC hit rate for accesses by the Core0 in the subsequent scan steps. Figure 4 shows our results. As expected, NoTouch access results in an LLC hit rate of almost 100%, with a negligible snoop hit rate. The Touch access results in a snoop hit rate of almost 100%. The high snoop hit rate evicts the cache

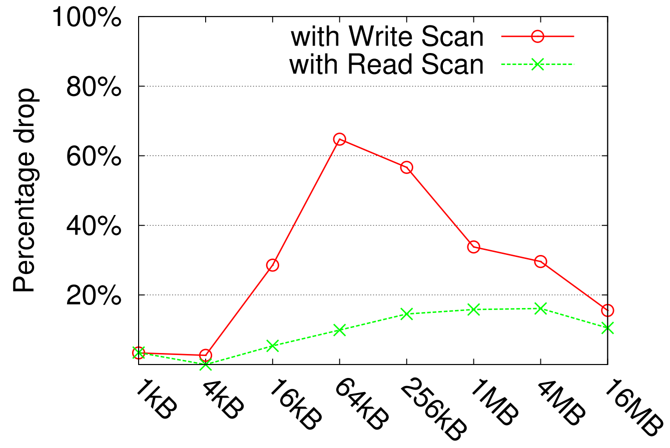

Figure 5: Performance degradation due to LLC misses and coherence overhead. The percentage performance drop is calculated by comparing the performances of the Touch access to the NoTouch accesses.

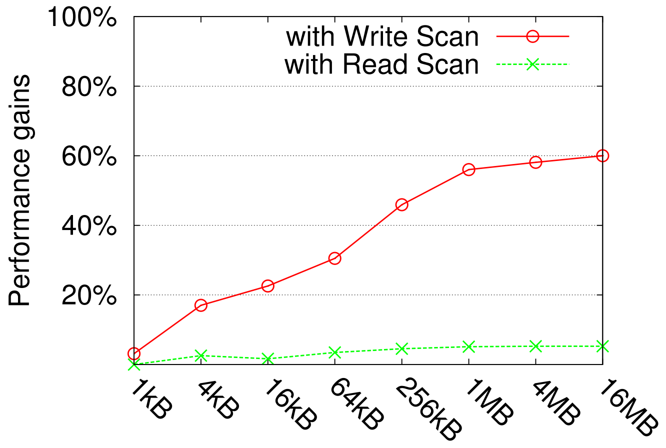

Figure 6: Performance gains due to Next-line hardware pre-fetching.

lines and consequently, further access to the transmission buffer by the Core0 misses the LLC. The LLC misses are not capacity or conflict misses, as only cache lines in the Invalid state are filled. Further analysis reveals that the snoop requests are of type `REMOTE_RFO` (remote Request For cache line Ownership). This ownership request moves Exclusive cache lines to the Invalid state and discards them. This state transition resulted in mandatory cache line misses for the Read Scans.

Summary: Write back of Modified cache lines is costly on Xeon E7520 due to high memory access latencies. However, more interestingly E7520's coherence engine interprets a DMA read request during the data transmission as a `REMOTE_RFO`. This request for ownership forces the coherence engine to evict even clean cache lines. The mismatch between DMA access intentions and coherence implementation results in a performance loss for the application where read-read sharing is expected.

3.2 LLC Misses and Coherence Overhead

In this section we investigate the effect of the buffer size on the coherence overhead. The buffer size is directly related to the number of cache lines that need work for coherence maintenance. Large buffer sizes result in a large number of cache lines, and consequently add coherence overhead. For mandatory cache misses (in the case of Write Scan), accessing a large buffer from DRAM with a cold cache is also costly. Figure 5 shows the effect of collective penalties of coherence overhead and cache misses. The y-axis shows performance degradation when Touch access is compared to NoTouch access. As shown in the previous section, the NoTouch access does not have any snoop-hits and maintains a high cache-hit rate. The Touch access results in snoop-hits and additional coherence maintenance work. For small buffer sizes, due to the small number of cache lines, the cost of coherence maintenance is small and relatively low compared to the overall network latency. As we increase the buffer size, this cost increases and becomes the dominant part of the overhead (4-256kB range). Further down the x-axis, with large buffer sizes the transmission cost on the 10GbE link becomes dominant and shadows the coherence overhead.

Summary: The shared access of the transmission buffer between the CPU core and network accelerator brings the accelerator into the memory coherence domain. Unlike the well studied (and optimized) behavior of memory sharing among many CPU cores, the performance implications of this shared access is not well understood. Different natures (inclusive or exclusive of L1 and L2) and implementations (topology, on- or off-chip) of last-level caches make reasoning about the performance a difficult problem. The architectural overheads stemming from cache misses, CPU stalls etc., have now become comparable to the network latencies. As illustrated, the (potential) performance gains in a shared access environment can be easily eclipsed by high architectural overheads.

3.3 Pre-Fetching and Buffer Access Pattern

Write back and (forced) eviction of cache lines result in mandatory cache misses. Because our benchmark is doing a sequential access (in a `for` loop) to the transmission buffer, the Next-line hardware pre-fetcher can fetch subsequent cache lines to avoid the high cache-miss penalty. However, real-world applications have complex data structure layouts in the transmission buffer, where parts of the buffers can be transmitted and received. Further, data can be accessed based on freshness, or urgent interest, e.g., only accessing the keys in a key-value pair. These types of accesses are strictly non-sequential and do not activate hardware pre-fetching. To understand the benefit of sequential access, we explicitly enabled and disabled pre-fetching in the BIOS. Figure 6 shows our findings. Hardware pre-fetchers can help to accelerate the end-application performance when accessing the cold transmission buffer but only under restricted access patterns. The gains from the sequential access (due to hardware fetching) can be as high as 60%.

Summary: Non-sequential access patterns that do not match any available pre-fetchers (adjacent-line, DCU streamer etc.) will not get any performance boost. Unaccelerated network stacks get benefits from software pre-fetching hints (using `prefetch` family instructions) passed during protocol processing and data copying in the kernel. In contrast, with RDMA, where data is directly transmitted and received from userspace, side-effects of DMA access (e.g., cold cache) are completely visible to the application and cannot be avoided. Hence, various cache optimization techniques and large cache sizes are of little help, and factors such as DRAM access latencies start to dominate performance of end-applications.

4. DISCUSSION

The distributed execution of application and network code on network accelerators is a radical departure from the traditional model, where everything is optimized to be accessed from a centralized host CPU. Hence, the interaction among off-chip non-CPU components becomes an important performance factor. These off-chip non-CPU components include shared caches, different coherency engine implementations, transport links (e.g., Intel QPI) to cores and DRAMs etc.

We realize that our investigation is processor and architecture specific, and therefore it would be wrong to make any final conclusions regarding the network acceleration technology. However, our findings do highlight (potential) architectural pitfalls, which are usually hidden from high-level applications while deploying network accelerators in large-scale environments.

Earlier works discuss network acceleration specifically in the context of transparent TCP offloading that maintains the socket interface to applications [15, 6]. Our analysis, however, is not limited to TCP/Socket interface and has wider implications. As I/O accelerator devices are becoming part of mainstream computing, I/O latencies are rapidly becoming closer to architectural overheads. Our findings have further implications as RDMA I/O interface and semantics are now being investigated even for GPUs [18] and storage [26].

4.1 Impact on Networked Applications

Various high-performance NoSQL data stores [5, 11, 20] have been proposed to serve multiple clients. Efforts have been made to transfer data by leveraging capabilities of RDMA network acceleration [24, 14]. Such applications that reportedly enjoy performance gains with RDMA can also experience performance loss if used with a particular CPU or chipset in a low-latency environment. Overheads reported in Section 3.1 affect servers, in Section 3.3 affect clients, and in Section 3.2 both. Other in-memory data stores [17, 27] are also susceptible to suffer performance losses. Also, in a shared environment cluster where storage and compute nodes are co-located, network serving of data by the storage application can potentially purge warm caches of the compute applications.

However, certain classes of applications are also less likely to be affected by reported overheads. Applications which have limited CPU-NIC interaction such as media streaming (e.g., YouTube), where the CPU brings video data into the memory once, and network accelerators such as RDMA, which can be used to serve content repeatedly to multiple clients [7], are not exposed to the overheads. Another class of applications are where the non-network part of the application, either computation or disk I/O, dominates the overall client latencies. These latencies are orders of magnitude higher (in msecs) than latencies reported in our setup.

4.2 Architectural Implications

In the previous sections, we have illustrated that architectural overheads can eclipse gains from network acceleration in high-performance environments. The exact overhead cost is sensitive to a particular implementation of coherence engine, cache-miss cost, write-back and DRAM access latencies etc. Hence, application developers must now be aware of costs associated with low-level architectural events. With ever increasing complex NICs and CPUs internals, there is a growing confusion about performance [22]. Different processor vendors implement different variations (or even a subset) of cache coherence protocols. Implementation and cost of architectural features can be different even between different models of the same processor generation [3].

With the high residual transistor count on the CPU chips, it should now be possible to implement high-performance NICs on CPU chips [4, 9]. This NIC-CPU integration makes network a first class citizen of CPUs, with access to all on-chip resources such as caches and memory controllers. This access enables a better interaction between caches and network I/O. Furthermore, network access to memories (DRAMs, caches or even NVRAMs) can be optimized (and reasoned about) using similar techniques for manycores CPUs. As there is no final word for high-performance network interfaces (hardware and software), it is a challenging task to design a single chip to meet all demands. However, demands for very high network performance (100Gbits/sec with less than 1μsecond latency [21]) necessitate this integration.

Another orthogonal issue is network accelerator integration in non I/O-coherent architectures, e.g., ARM. In such architectures, understanding the interaction among non-CPU components (caches, DMA and coherence) is even necessary for the sake of correctness. The current OFED RDMA subsystem on Linux is broken for non I/O-coherent architectures [19].

4.3 Experience

RDMA offers low (5-10μsecs) data access latencies together with very high data bandwidths (10-40Gbps) with a negligible CPU load. Due to its unique performance potential it has (again) started to draw a lot of attention from the systems building community [25, 24, 14, 23]. However, managing network resources in userspace does expose applications to low-level hardware details which are usually hidden in the operating system kernel. For example, in order to reduce cache pollution, Linux uses non-temporal copy instructions (e.g., `movnti`) to copy data from user buffers to SKBs. It also hides the cost of cache misses from applications by doing pre-fetching and data copy during the network processing.

Reasoning about RDMA performance requires a good understanding of CPU, NIC, and architecture internals. Complex off-CPU components, primitive performance monitoring facilities, ambiguous documentation of hardware, and a limited software support etc. make RDMA performance analysis a very challenging task.

5. CONCLUSION

In this paper, we documented our experience with network acceleration using RDMA. We have demonstrated that a number of (un)related parameters can significantly affect the gains of applications when using network accelerators.

Unfortunately, there is no silver bullet solution that guarantees performance improvements. As we move toward a heterogeneous computing environment, the use of network accelerator devices will become more common. This will change the decade old assumption that all processing and data access happens from the central CPUs. Thus, instead of focusing on a high CPU core performance, system architects must take a holistic system-wide approach toward network accelerator integration to achieve application performance boosts.

Acknowledgements

We thank the anonymous reviewers and our shepherd, Jon Crowcroft, for their helpful comments and suggestions. We would also like to thank Zoltan Majo for his feedback.

6. REFERENCES

[1] perf: Linux profiling with performance counters. http://perf.wiki.kernel.org/.

[2] Apache Hadoop. `http://hadoop.apache.org/`.

[3] D. Bachand, S. Bilgin, R. Greiner, P. Hammarlund, D. L. Hill, T. Huff, S. Kulick, and R. Safranek. The Uncore: A Modular Approach to Feeding The High-Performance Cores. In *Intel Technology Journal*, Volume 14, Issue 3, pages 30–49, 2010.

[4] J. Brown, S. Woodward, B. Bass, and C. Johnson. IBM Power Edge of Network Processor: A Wire-Speed System on a Chip. *Micro, IEEE*, 31(2):76–85, March-April.

[5] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of 21st SOSP 2007*, pages 205–220.

[6] D. Freimuth, E. Hu, J. LaVoie, R. Mraz, E. Nahum, P. Pradhan, and J. Tracey. Server Network Scalability and TCP Offload. In *Proceedings of the USENIX Annual Technical Conference*, ATC '05, pages 209–222, 2005.

[7] P. W. Frey, A. Hasler, B. Metzler, and G. Alonso. Server-efficient high-definition media dissemination. In *Proceedings of the 18th NOSSDAV '09*, pages 49–54.

[8] Intel. Intel Xeon Processor 7500 Series Uncore Programming Guide at `http://www.intel.com/Assets/en_US/PDF/designguide/323535.pdf`.

[9] Intel weaves strategy to put interconnect fabrics on chip. `http://www.hpcwire.com/hpcwire/2012-09-10/intel_weaves_strategy_to_put_interconnect_fabrics_on_chip.html`, 2012.

[10] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM EuroSys 2007*, pages 59–72.

[11] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, Apr. 2010.

[12] J. Liu, D. Poff, and B. Abali. Evaluating high performance communication: a power perspective. In *Proceedings of the 23rd ICS 2009*, pages 326–337.

[13] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD*, pages 135–146.

[14] C. Mitchell, Y. Geng, and J. Li. Using one-sided rdma reads to build a fast, cpu-efficient key-value store. In *Proceedings of the 2013 USENIX Annual Technical Conference*, USENIX ATC'13, pages 103–114, 2013.

[15] 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 - Volume 9*, HotOS'03, pages 5–5, 2003.

[16] Netperf, 2.4.5. http://www.netperf.org/netperf/.

[17] J. Ousterhout et al. The case for ramclouds: scalable high-performance storage entirely in dram. *SIGOPS Oper. Syst. Rev.*, 43(4):92–105, Jan. 2010.

[18] RDMA for GPUDirect, CUDA Toolkit Documentation. `http://docs.nvidia.com/cuda/gpudirect-rdma/index.html`, 2013.

[19] (R)DMA in userspace on Linux RDMA mailing list. `http://comments.gmane.org/gmane.linux.drivers.rdma/13635`, october, 2012.

[20] Redis in memory key-value store. http://redis.io.

[21] S. M. Rumble, D. Ongaro, R. Stutsman, M. Rosenblum, and J. K. Ousterhout. It's time for low latency. In *Proc. of the 13th HotOS*, pages 11–11, 2011.

[22] P. Shinde, A. Kaufmann, T. Roscoe, and S. Kaestle. We Need to Talk About NICs. In *Proceedings of the 14th USENIX workshop on Hot Topics in Operating Systems*, HotOS'13, pages 1–1, 2013.

[23] P. Stuedi, B. Metzler, and A. Trivedi. jVerbs: Ultra-low Latency for Data Center Applications. In *Proceedings of the 4th ACM Symposium on Cloud Computing*, SOCC'13, 2013.

[24] P. Stuedi, A. Trivedi, and B. Metzler. Wimpy nodes with 10GbE: leveraging one-sided operations in soft-RDMA to boost memcached. In *Proceedings of the USENIX ATC*, 2012.

[25] A. Trivedi, B. Metzler, and P. Stuedi. A case for RDMA in clouds: turning supercomputer networking into commodity. In *Proc. of the 2nd APSys*, pages 17:1–17:5, 2011.

[26] A. Trivedi, P. Stuedi, B. Metzler, R. Pletka, B. G. Fitch, and T. R. Gross. Unified High-Performance I/O: One Stack to Rule Them All. In *Proceedings of the 14th USENIX workshop on Hot Topics in Operating Systems*, HotOS'13, pages 4–4, 2013.

[27] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX NSDI*, pages 2–2.