A Case for RDMA in Clouds: Turning Supercomputer Networking into Commodity

Animesh Trivedi
IBM Research
Saeumerstrasse 4
Rueschlikon, Switzerland
atr@zurich.ibm.com

Bernard Metzler
IBM Research
Saeumerstrasse 4
Rueschlikon, Switzerland
bmt@zurich.ibm.com

Patrick Stuedi
IBM Research
Saeumerstrasse 4
Rueschlikon, Switzerland
stu@zurich.ibm.com

ABSTRACT

Modern cloud computing infrastructures are steadily pushing the performance of their network stacks. At the hardware-level, already some cloud providers have upgraded parts of their network to 10GbE. At the same time there is a continuous effort within the cloud community to improve the network performance inside the virtualization layers. The low-latency/high-throughput properties of those network interfaces are not only opening the cloud for HPC applications, they will also be well received by traditional large scale web applications or data processing frameworks. However, as commodity networks get faster the burden on the end hosts increases. Inefficient memory copying in socket-based networking takes up a significant fraction of the end-to-end latency and also creates serious CPU load on the host machine. Years ago, the supercomputing community has developed RDMA network stacks like Infiniband that offer both low end-to-end latency as well as a low CPU footprint. While adopting RDMA to the commodity cloud environment is difficult (costly, requires special hardware) we argue in this paper that most of the benefits of RDMA can in fact be provided in software. To demonstrate our findings we have implemented and evaluated a prototype of a software-based RDMA stack. Our results, when compared to a socket/TCP approach (with TCP receive copy offload) show significant reduction in end-to-end latencies for messages greater than modest 64kB and reduction of CPU load (w/o TCP receive copy offload) for better efficiency while saturating the 10Gbit/s link.

Categories and Subject Descriptors

C.2.5 [**Local and Wide-Area Networks**]: Ethernet; D.4.4 [**Communications Management**]: Network communication—*RDMA, Efficient data transfer, performance measurement*

APSys '11, July 11-12 2011, Shanghai, China

General Terms

Design, Measurement, Performance

Keywords

RDMA, Commodity Clouds, Ethernet Networking

1. INTRODUCTION

Applications running in today's cloud computing infrastructures are increasingly making high demands on the network layer. To support current data-intensive applications, terabytes worth of data is moved everyday among the servers inside a data center requiring high network throughput. For large scale web applications like Facebook low data access latencies are key to provide a responsive interactive experience to users [4]. And recently there have been cases running HPC workload inside scalable commodity cloud strengthening the need for a high performance network stack further [1]. To accommodate those requirements modern cloud infrastructures are continuously pushing the performance of their network stack. Most of the work focuses on improving the network performance inside the virtualization layers [14, 21]. In parallel some cloud providers have upgraded parts of the networking hardware to 10GbE which recently has moved closer to the price range acceptable for commodity clouds.

While high network performance is greatly desired, it also increases the burden for end hosts. This is especially true for applications relying on standard TCP sockets as their communication abstraction. With raw link speeds of 10Gbit/s, protocol processing inside the TCP stack can consume a significant amount of CPU cycles and also increases the end-to-end latency perceived by the application. A large part of the inefficiency stems from unnecessary copying of data inside the OS kernel [3]. The increased CPU load is particularly problematic as parallelizing the network stack remains challenging [25].

High performance networks are state of the art in supercomputers since a long time. For instance, the Remote Direct Memory Access (RDMA) network stack provides low-latency/high-throughput with a small CPU footprint. This is achieved in RDMA mainly by omitting intermediate data copies and offloading the protocol processing to hardware [16]. Adopting RDMA by commodity clouds is, however, difficult due to cost and special hardware requirements. In this paper we argue that the basic concept and the semantics of RDMA can be beneficial for modern cloud infrastructures

[1] http://aws.amazon.com/hpc-applications/

as the raw network speed increases, and – most importantly – that RDMA network support can efficiently be provided in pure software, without a need for special hardware. To demonstrate our findings we have developed SoftiWARP, a software-based RDMA stack. We have evaluated our stack against a socket-based approach and show that SoftiWARP provides high-performance network I/O with reduced CPU load and significantly lower end-to-end-latencies for reasonably large message sizes.

2. REVISITING SUPERCOMPUTER NETWORKING

Supercomputers are designed to run specialized workloads and are built using customized hardware and interconnects. The type of workload running on supercomputers, e.g. MPI-based data processing, requires super-fast and efficient network I/O to shuffle data across the interconnects. At a first glance, it seems apparent that the requirements faced by supercomputers in the past resemble the ones of future commodity data centers. Thus, a natural next step would be to look at the network stack operated by supercomputers and see if a similar approach could be beneficial for data centers as well. RDMA is one networking technology used by supercomputers. RDMA enables applications to have high bandwidth and low latency access to data by efficiently moving data between application buffers. It is built upon the *user-level networking* concept and separates data from the control path, as only the latter requires host OS involvement. An important aspect of RDMA when compared to socket-based networking is that RDMA has rich asynchronous communication semantics, that helps in overlapping communication with computation for better resource utilization.

2.1 Commodity Clouds and RDMA

Clouds run on data centers built from commodity hardware and perform on-demand, flexible and scalable resource allocation for the applications. Resource utilization becomes an important factor to cloud efficiency because hardware resources (e.g. CPU, memory, and networks) are being multiplexed among many virtual machines. Due to the economy of scale, inefficiencies in resource usage can potentially eclipse large gains from the clouds. In the following we look at CPU usage, latency and energy consumption in data centers and discuss how RDMA can improve the performance in those cases.

CPU Usage: Network I/O can cost a lot of CPU cycles [3]. Applications in clouds require better support from the OS to efficiently transfer large quantities of data. RDMA-enabled NICs (RNICs) have sufficient information to completely bypass the host OS and directly move data between application buffers and the network without any intermediate copies. This requires considerably lower CPU involvement and frees up CPU cycles for productive application processing.

Latency: Low data access latencies are key for large-scale web applications like Twitter or Facebook in order to provide sufficient responsiveness to user interactions [20]. Low data access latencies also play an important role in determining which consistency model a cloud storage layer can possibly implement [23]. And finally, absence of timely access to the data may render certain applications inefficient and cause a loss of productivity. RDMA helps in reducing end-to-end application latencies as it does not require local or remote applications to be scheduled during network transfers. As a consequence RDMA also is minimizing the penalty for processor state pollution such as cache flushes.

Energy: The energy consumption of commodity data centers have been alarming [2]. Future cloud infrastructures need to squeeze more performance per Watt. Because of an efficient data movement pattern on the memory bus (zero-copy), less application involvement, and low OS overhead, RDMA data transfers are more energy efficient than traditional socket based transfers [12].

2.2 Challenges for RDMA in Clouds

Although promising, RDMA has neither extensive end-user experience and expertise nor widespread deployment outside the HPC world. We point out three major challenges in deploying RDMA hardware in commodity cloud infrastructure.

First, to efficiently move data between NIC and application buffers, current RDMA technology offloads the transport stack. The issues related to integration of stateful offload NICs in mature operating systems are well documented [16]. These problems include: consistently maintaining shared protocol resources such as port space, routing tables or protocol statistics, scalability, security and bug fixing issues etc.

Second, clouds run on commodity data centers. Deploying RDMA in such an environment might be expensive. RDMA requires special adapters with offload capabilities like RNICs. The usability of those adapters, however, is limited to a couple of years at best since processor and software advancements are catching up rapidly.

Third, RDMA technology has often been criticized for its complex and inflexible host resource management [10]. RDMA applications are required to be completely aware of their resource needs. System resources such as memory, queue pairs etc. are then statically committed on connection initialization. This goes against the cloud philosophy of flexible and on-demand resource allocation.

3. RDMA IN SOFTWARE

Despite challenges we argue that a major subset of RDMA benefits still can be provided to commodity data center environment using only software-based stacks. In this section we present a case for the software-based RDMA stack, discuss its advantages, and why it is a good match for the clouds.

First, there are currently several networks stacks supporting RDMA communications semantics. iWARP [11] (Internet Wide Area RDMA Protocol) being one of them, enables RDMA transport on IP networks. In contrast to other RDMA stacks like Infiniband, iWARP uses TCP/IP or SCTP/IP as end-to-end transport. This enables RDMA on the ubiquitous Ethernet infrastructure typical to commodity data centers and opens the door to inexpensive, Internet-wide and pure software-based implementations.

Second, we believe that the availability of sufficient application specific knowledge (passed using the RDMA API) during network protocol processing can single-handedly boost efficiency [7], obsoleting offloading in many cases. The rich RDMA API supports one-sided pull/push operations, semantic aggregation of operations, grouping of work postings, and completion notifications etc. Using this knowledge with modest support from the OS, goals of the RDMA stack such as zero copy transmission, direct data placement into appli-

cation's buffer, and one-sided operations are possible even in a software-based solution. This results in better end-to-end latencies, and low CPU footprint in the commodity data centers.

Third, with its flexible resource management policies, a software-based solution is matching closely with the requirements of the cloud. It also helps in materializing *on-demand RDMA networks* between hosts. While dynamically managing resources can result in some performance loss, applications can amortize this cost up-to an extend by reusing I/O critical resources such as memory regions. Please note that raw performance is not the primary concern for such a software-based stack. It can not outperform a hardware based solution but still can provide a large subset of RDMA benefits in pure software.

3.1 SoftiWARP

We have implemented the iWARP protocol in a software stack for Linux called *SoftiWARP* [15]. SoftiWARP enables commodity Ethernet NICs to handle RDMA traffic in software. It aims to provide the benefits of rich RDMA semantics, while avoiding most of the problems with TOEs or RNIC deployment. By integrating transparently within industry standard OFED stack [18], SoftiWARP appears to the application as a regular RNIC device. It also interoperates with available RNIC hardware such as the Chelsio T3 adapters.

3.1.1 Comparing SoftiWARP with plain TCP

Historically, TCP implementations have been closely associated with the BSD socket interface. The socket API abstracts all protocol details and exports a simple send and receive interface for data transfer. From an applications perspective, all additional communication semantics must be a part of the application itself. With this most of the data path optimizations RDMA is focusing on are not consistently achievable, despite some attempts to augment the OS networking stack with shortcuts depending on implicit state information(e.g. [6]). In this section we highlight the advantages RDMA brings to applications and compare it to socket based TCP network I/O operations:

Application Data Copy Avoidance: To preserve system call semantics, the socket interface must copy application data between user space and kernel. The RDMA API helps to avoid copy operations on the transmit path by explicitly passing buffer ownership when posting a work request. Any sending or retransmission of data can be done directly from the user buffer. However using kernel TCP sockets, the current SoftiWARP implementation does not explicitly know when data is finally delivered to the peer and restricts zero copy to process a non-signaled work. A simple check of TCP send state information would be sufficient to appropriately delay work completion generation.

Less Application Scheduling: With the necessary data placement information known, SoftiWARP places data directly into user's buffers from the TCP receive softirq processing context. Also, while processing one-sided operations, it does not schedule the application for network I/O. These characteristics are particularly attractive for applications such as media streaming, CDNs, distributed data storage and processing etc. Furthermore, work completion processing is minimized to the application needs – completing non-signaled work will not trigger application scheduling.

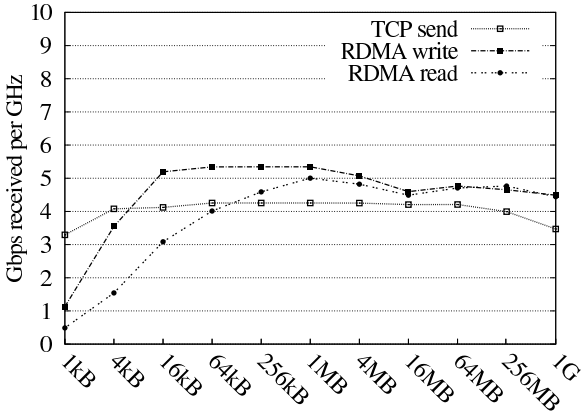

Figure 1: Receive efficiency of TCP w/o receive copy offloading and SoftiWARP

Non-blocking Operations: The asynchronous application interface frees applications from blocking and waits on data send or receive operations to complete. If semantically needed, the application may still do a blocking wait for completion.

Flexible Memory Management: User buffer pinning in memory is inevitable for hardware RDMA implementations [10]. However a software based in-kernel RDMA implementation has flexibility of performing on-demand buffer pinning, once source or sink address and data length is known. This puts memory resource economics on par with kernel TCP stack, though it may result in some performance loss.

3.1.2 SoftiWARP and RNICs:

Looking beyond the context of this paper, SoftiWARP will likely not replace RNICs on given installations. It will also not obsolete their applicability on high-end server systems or if very low delay requirements are stringent. Rather, it enables RDMA communication on any commodity system. Heterogeneous setups, deploying RNICs on the server side and running a software RDMA stack on the typically less loaded client, are thus made possible.

4. PERFORMANCE

In this section we will give some early performance results for SoftiWARP. The numbers reported are average of 3 runs, each lasting 60 seconds. The experiments were run on identical IBM[1] HS22 blades containing dual Quadcore 2.5 GHz Intel Xeon CPUs (E554), 8GB RAM and connected using Mellanox ConnectX 10GbE adapters. All experiments were done on Linux (`2.6.36`) using netperf [17] with our extensions for RDMA tests and oprofile [19]. We plan to open source these extensions which consist of uni-directional data transfer tests. To obtain CPU usage, we divided the total number of CPU samples obtained during a benchmark run by the number of samples obtained when we ran a busy loop on a single CPU. Hence they include the overhead of buffer allocation and page pinning for the RDMA. We report two variants of TCP performance, with and without TCP receive copy offload using Intel QuickData [1]technology in Linux(`CONFIG_NET_DMA`). Although helpful with the CPU usage for small size messages, in our experiments we have found TCP receive copy offloading to be major source of performance degradation for the large message sizes.

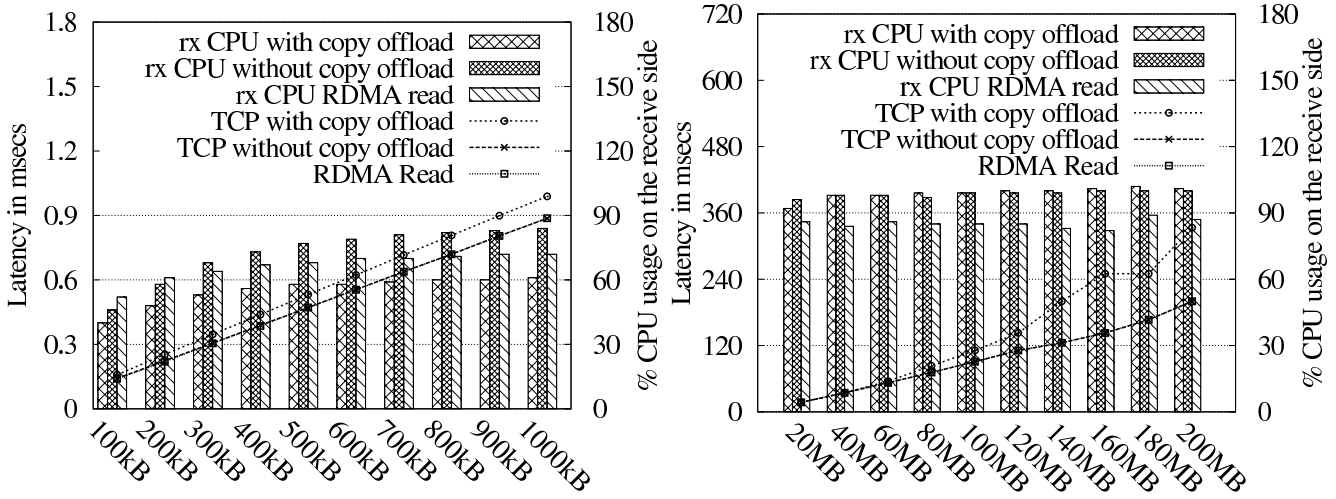

Figure 2: Round trip latencies for TCP request response and RDMA read a) 100-1000kB, b) 20-200MB data blocks

Bandwidth: SoftiWARP does not impose any performance penalty for large buffer sizes. Compared to TCP's buffer size of 2kB, SoftiWARP achieves full line speed with buffer sizes of 16kB and 32kB for RDMA Write and Read tests respectively (not shown in the paper). If aggregate buffer posting using multiple work elements (100) is performed, it saturates the link with message sizes of 8kB and 16kB for RDMA Write and Read tests respectively. We are working on optimizing SoftiWARP's performance for transmitting small buffer sizes.

CPU Efficiency: SoftiWARP is very economical with the CPU cycles. By using zero copy mechanism on the transmit side, SoftiWARP's CPU usage for non-signaled workloads is approximately 40%-45% (not shown), which is less than a send call and at par with TCP sendpage mechanism. However for, small buffer size (less than 64kB) send is more efficient. On the receiving side SoftiWARP consumes 8%-22% less CPU than TCP while receiving at the full link speed. Figure 1 shows the receive side efficiency of SoftiWARP and TCP (w/o TCP receive copy offload) in terms of Gbits received per GHz of CPU use. Beyond a modest 16kB (64kB for reads), SoftiWARP's receive is more efficient than the TCP receive.

Latency: The latency experiment consists of a request (4 bytes) and response (variable, asked size) system. Figure 2 shows end-to-end delay of a request with the CPU usage on the receiving side. For small requests (in kBs), TCP copy offloading saves significant CPU cycles (13%-27% compared to the non-offloaded version) but it results in a non-negligible performance penalty for the end-to-end latencies. SoftiWARP offers 11%-15% (for 100kB-1000kB range) and a significant 5%-67% (20MB-200MB range) reduction in end-to-end application latencies over TCP (copy offloaded version). However without TCP receive copy offloading, the performance gap closes at a cost of higher CPU usage (5%-18%). On the response transmitting side by using in-kernel sendpage mechanism for already pinned pages, SoftiWARP improves the CPU usage by 34%-50% (not shown) against TCP.

The CPU and performance gains are primarily achieved by avoiding application involvement and exploiting zero copy mechanism for the transmission. We will provide an in depth-analysis of the system in future.

5. RELATED WORK

iWARP is built upon the Virtual Interface Architecture (VIA) concept, which has roots in systems such as U-Net [24], Hamlyn [5], ADC [9], Typhoon [22] etc. Although these systems helped in developing the key principles behind iWARP and RDMA, but in the absence of any standardization work lead to multiple ad-hoc implementations of application and network interfaces. SoftiWARP is an effort to find a golden middle ground between trivial read/write system calls on sockets and exotic user-accessible network interface exports.

Unlike true user-level networking systems, SoftiWARP does not run the network stack in user space. Networking remains to be a service of the OS but user-tailored. SoftiWARP uses application specific knowledge, which is available through rich RDMA API, to make efficient use of resources. Also, giving resource control to the kernel instantiates its status as a resource arbitrator that helps in imposing global policies [13]. The Ohio Supercomputer Center has presented another kernel based iWARP implementation [8]. Their system, however, is not compatible with the OFED framework.

6. CONCLUSION

With the availability of 10Gbit/s Ethernet for commodity data centers and the ever increasing demand of distributed applications for efficient inter-node network I/O, network stacks are challenged to keep up with the pace. In this paper, we argue that traditional socket-based networking is not capable of satisfying those future demands, as it puts too big of a burden onto the host. Instead we propose the use of RDMA as a networking concept to overcome those issues. RDMA traditionally has been used as a high-performance networking technology in supercomputers, but they are costly and require special hardware which makes them difficult to be adopted by commodity data centers. However, we believe that the main advantages of RDMA can be provided in pure software inside commodity data cen-

ters. In the paper, we discuss an early implementation of a software-based RDMA solution and present experimental results. The results confirm the high-throughput performance of such approach while significantly reducing end-to-end application latencies than plain socket based TCP/IP.

In the future we plan to investigate potential performance benefits of SoftiWARP in the presence of OS and application level virtualization. SoftiWARP integrates seamlessly with asynchronous communication APIs found in, e.g., Java NIO, and may allow to efficiently bypass virtual layers down to the hypervisor.

SoftiWARP is an open source project and the code is available at `www.gitorious.org/softiwarp`.

7. REFERENCES

[1] Intel Corporation. Intel QuickData Technology Software Guide for Linux* at `http://www.intel.com/technology/quickdata/whitepapers/sw_guide_linux.pdf`, 2008.

[2] D. G. Andersen et al. FAWN: a fast array of wimpy nodes. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 1–14, New York, NY, USA, 2009. ACM.

[3] P. Balaji. Sockets vs RDMA Interface over 10-Gigabit Networks: An In-depth analysis of the Memory Traffic Bottleneck. In *In RAIT workshop*, 2004.

[4] D. Beaver et al. Finding a needle in Haystack: Facebook's photo storage. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–8, Berkeley, CA, USA, 2010. USENIX Association.

[5] G. Buzzard et al. An implementation of the Hamlyn sender-managed interface architecture. In *Proceedings of the second USENIX symposium on Operating systems design and implementation*, OSDI '96, pages 245–259, New York, NY, USA, 1996. ACM.

[6] H.-k. J. Chu. Zero-copy TCP in Solaris. In *Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*, pages 21–21, Berkeley, CA, USA, 1996. USENIX Association.

[7] D. D. Clark et al. Architectural considerations for a new generation of protocols. In *Proceedings of the ACM symposium on Communications architectures & protocols*, SIGCOMM '90, pages 200–208, New York, NY, USA, 1990. ACM.

[8] D. Dalessandro et al. iWARP protocol kernel space software implementation. In *Proceedings of the 20th international conference on Parallel and distributed processing*, IPDPS'06, pages 274–274, Washington, DC, USA, 2006. IEEE Computer Society.

[9] P. Druschel et al. Experiences with a high-speed network adaptor: a software perspective. In *Proceedings of the conference on Communications architectures, protocols and applications*, SIGCOMM '94, pages 2–13, New York, NY, USA, 1994. ACM.

[10] P. W. Frey et al. Minimizing the Hidden Cost of RDMA. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, pages 553–560, Washington, DC, USA, 2009. IEEE Computer Society.

[11] IETF. Remote direct data placement working group. `http://datatracker.ietf.org/wg/rddp/charter/`.

[12] J. Liu et al. Evaluating high performance communication: a power perspective. In *Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 326–337, New York, NY, USA, 2009. ACM.

[13] K. Magoutis. The Case Against User-level Networking. In *In Third Workshop on Novel Uses of System Area Networks (SAN-3)*, 2004.

[14] A. Menon et al. Optimizing network virtualization in Xen. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 2–2, Berkeley, CA, USA, 2006. USENIX Association.

[15] B. Metzler, P. Frey, and A. Trivedi. SoftiWARP - Project Update, 2010. Available online at `http://www.openfabrics.org/OFA-Events-sonoma2010.html`.

[16] 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*, pages 5–5, Berkeley, CA, USA, 2003. USENIX Association.

[17] Netperf, 2.4.5. `http://www.netperf.org/netperf/`, 2011.

[18] OpenFabric Alliance. OpenFabrics Enterprise Distribution (OFED) Stack, 2010. Available online at `www.openfabrics.org`.

[19] Oprofile, 0.9.6. `http://oprofile.sourceforge.net`, 2011.

[20] J. Ousterhout et al. The case for RAMCloud. *Commun. ACM*, 54:121–130, July 2011.

[21] K. K. Ram et al. Achieving 10 Gb/s using safe and transparent network interface virtualization. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '09, pages 61–70, New York, NY, USA, 2009. ACM.

[22] S. K. Reinhardt et al. Tempest and Typhoon: user-level shared memory. In *Proceedings of the 21st annual international symposium on Computer architecture*, ISCA '94, pages 325–336, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[23] B. Tiwana et al. Location, location, location!: modeling data proximity in the cloud. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets '10, pages 15:1–15:6, New York, NY, USA, 2010. ACM.

[24] T. von Eicken et al. U-Net: a user-level network interface for parallel and distributed computing. In *Proceedings of the fifteenth ACM symposium on Operating systems principles*, SOSP '95, pages 40–53, New York, NY, USA, 1995. ACM.

[25] P. Willmann et al. An evaluation of network stack parallelization strategies in modern operating systems. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 8–8, Berkeley, CA, USA, 2006. USENIX Association.

Notes

[1]IBM is a trademark of International Business Machines Corporation, registered in many jurisdictions worldwide. Intel is a registered trademark of Intel Corporation or its subsidiaries in the United States and other countries. Other product and service names might be trademarks of IBM or other companies.