Databases on Modern Networks: A Decade of Research That Now Comes into Practice

Alberto Lerner
University of Fribourg, Switzerland
alberto.lerner@unifr.ch

Carsten Binnig
TU Darmstadt & Google
carsten.binnig@cs.tu-darmstadt.de

Philippe Cudré-Mauroux
University of Fribourg, Switzerland
philippe.cudre-mauroux@unifr.ch

Rana Hussein
University of Fribourg, Switzerland
rana.hussein@unifr.ch

Matthias Jasny
TU Darmstadt
matthias.jasny@cs.tu-darmstadt.de

Theo Jepsen
Intel
theo.jepsen@intel.com

Dan R. K. Ports
Microsoft Research
Dan.Ports@microsoft.com

Lasse Thostrup
TU Darmstadt
lasse.thostrup@cs.tu-darmstadt.de

Tobias Ziegler
TU Darmstadt
tobias.ziegler@cs.tu-darmstadt.de

ABSTRACT

Modern cloud networks are a fundamental pillar of data-intensive applications. They provide high-speed transaction (packet) rates and low overhead, enabling, for instance, truly scalable database designs. These networks, however, are fundamentally different from conventional ones. Arguably, the two key discerning technologies are RDMA and programmable network devices. Today, these technologies are not niche technologies anymore and are widely deployed across all major cloud vendors. The question is thus not if but how a new breed of data-intensive applications can benefit from modern networks, given the perceived difficulty in using and programming them. This tutorial addresses these challenges by exposing how the underlying principles changed as the network evolved and by presenting the new system design opportunities they opened. In the process, we also discuss several hard-earned lessons accumulated by making the transition first-hand.

PVLDB Reference Format:
Alberto Lerner, Carsten Binnig, Philippe Cudré-Mauroux, Rana Hussein, Matthias Jasny, Theo Jepsen, Dan R. K. Ports, Lasse Thostrup, and Tobias Ziegler. Databases on Modern Networks: A Decade of Research That Now Comes into Practice. PVLDB, 16(12): 3894 - 3897, 2023.
doi:10.14778/3611540.3611579

1 BACKGROUND AND MOTIVATION

The old networks. The discipline of programming old networks involved simple concepts. Messages were exchanged through sockets, which invariably forced applications to use a sequence of simple send-receive patterns to communicate [22]. Message forwarding was completely opaque to the application. Once a message was passed on to an initiating machine's `send` call, it would simply resurface on the desired target machine's `recv` call. How the network was structured between the initiating and target machines did not matter to the application.

Modern networks are very different. In the last decade, much has changed in networking to enable data-centric systems and applications at the cloud scale [2]. Modern networks are larger, faster, more efficient, and offer more services. Unsurprisingly, they bring many changes in how they are programmed. These changes and how to harvest their benefits when building data-intensive applications are the topic of this tutorial.

What are the changes, and why invest in them now? The two technologies that best exemplify these fundamental abstraction shifts are RDMA and programmable network devices. We will discuss them shortly, but for now, it suffices to say that for someone that works in the cloud provider industry, the question of why to use these technologies is moot. This industry has been using one or both of them for a few years because, quite simply, cloud providers cannot afford to forgo technologies that are efficient and deliver high performance. For a cloud customer, the story used to be different. In the past, it was difficult to access these technologies, and the learning curve was discouraging. Currently, these technologies are off-the-shelf, and some are already offered online. The risk for researchers and practitioners building innovative systems without these technologies is to find themselves behind systems that do.

What do those techniques provide? RDMA stands for Remote Direct Memory Access, and, as the name implies, blurs the boundaries among servers allowing, for instance, for a process to read the memory of a remote machine. Programmable network devices allow applications to customize the way the network hardware behaves. They allow, for instance, semantics-based routing, e.g., routing a request to a server that is available rather than to a fixed destination address.

What can you learn from this tutorial? First, we make the case that it is the right time for system builders to learn about these changes and talk about the underlying hardware supporting them in great detail. Second, we explain the key fundamentals of how to use the technologies, in particular of how to adapt them for database systems use cases. With this material, system builders will be able to create very efficient, very fast data movement primitives. Lastly, the authors have collectively amassed a large body of knowledge adopting the new technologies. The tutorial distills these experiences into recommendations, pitfalls to avoid, and best practices the tutorial attendants can incorporate immediately.

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611579

The tutorial is structured as follows:

- **Part I: Modern Networking Infrastructure (§ 2)** presents the basic infrastructure of data-center-sized/cloud networks, focusing on their hardware and topology. We use cloud networks here because they are increasingly opening RDMA and accelerator technologies to customers (besides using them internally) and because on-premise installations nowadays can use the same technologies albeit at a smaller scale.
- **Part II: RDMA and Derivatives (§ 3)** introduces RDMA as a means for high-performance database systems to efficiently interact with the network. We provide a primer on RDMA—and its derivatives in the cloud—and discuss lessons learned over a decade on how to leverage RDMA efficiently for designing scalable systems.
- **Part III: Programmable Devices (§ 4)** discusses how to leverage the network's programmable devices to accelerate or offload application logic. We introduce the most common computing models and show how to use them to express data structures and computations that NICs and switches can execute. Like above, we comment on pitfalls to avoid.
- **Part IV: Open Problems and Next Steps (§ 5)** argues that we are seeing the dawn of a next generation of cloud data-intensive systems that fully embraces modern networks. It discusses the design space for these new systems and pinpoints promising open research areas.

2 MODERN NETWORKING INFRASTRUCTURE

Data centers rely on fast networks to support customer-applications traffic and provider-operated services. They are engineered to be fast, bringing 40 Gb to 100 Gb Ethernet links to each server, with 400 Gb on the horizon. They are also larger. A single network can connect tens to hundreds of thousands of servers with an unprecedented *bisection bandwidth*[1]. To understand the scale of these networks, consider that Google reported that one of its datacenters had 1.2 Petabits of bisection bandwidth in 2012 [20]. If all the servers on the network were dedicated to data-intensive systems, and the average database row was 100 bytes, such a network could transport 1.5 *trillion* tuples at any point in time—in a single datacenter. In today's numbers, this could be orders of magnitude larger!

The need for new server stacks. A network rate of 100 Gb means that a packet may arrive at a server roughly every 6.7 nanoseconds. At 400 Gb, the rate goes down proportionally. Traditional hardware and software stacks cannot keep this pace. This time scale is orders of magnitude below what even a single system call would take, which is what the old interface and network hardware used to receive/transmit a single message.

Modern server stacks are designed to address these issues. These stacks embrace two underlying principles to achieve efficiency and high performance. First, they allow the application to communicate directly with the network card in what is called *OS bypass.* In other words, data transmission does not involve system calls anymore. Second, an application needs only to point to the data it wishes to transmit. The card reads data directly from userspace in what is deemed *zero-copy.* Not surprisingly, RDMA and its variations adopt these principles. We will discuss how in Section 3, but for now, we note that this combination is so powerful that Microsoft reported that RDMA protocols currently carry 70% of its storage traffic [1].

The need for new switches. The fundamental changes with switches came from the need to manage a larger, potentially heterogeneous fleet of them. Software-Defined Networks (SDN) was the first attempt to make commercial switches adopt a common control interface. This interface allowed for managing large networks centrally, which was welcome by hyperscalers. However, SDN was still not flexible enough. As virtualization technologies evolve, they brought a constant stream of new crucial networking protocols that needed to be implemented very fast. Waiting until a new hardware switch reached the market that supported a new set of much-needed protocols was just unsustainable. In the mid-2010s, a new class of hardware switches became commercial that solved this issue via programmability. These switches could execute networking protocols that were encoded as software programs. We will discuss this in more detail in Section 4.

Lessons Learned. Understanding the characteristics of modern networking hardware—i.e., NICs, switches, and accelerators—helps to explain several mechanisms and design decisions that the modern networking programming abstractions adopted. This is the foundation to understand modern networks.

3 RDMA AND DERIVATIVES

RDMA is gaining traction in data center networks for all major cloud providers, enabling efficient routing of massive application traffic at scale with low latency. Simultaneously, the DBMS market pivoted from on-premise to cloud-based solutions in recent years. Consequently, we believe it is an opportune moment for system builders to adopt RDMA for constructing scalable database systems. In this part of the tutorial, we briefly introduce RDMA and its cloud-deployed derivatives, followed by an aggregated perspective on the insights gained over the past decade on effectively utilizing RDMA for designing scalable database systems.

An RDMA Primer. As part of the RDMA primer, we first provide an overview of the basics of the original RDMA technologies developed for high-performance computing (HPC) and then discuss the major differences between RDMA derivatives in the cloud.

RDMA enables direct remote memory access in a cluster without engaging the remote system's CPU. This technique also avoids unnecessary data copies on the sender's side through zero-copy transfer, reducing CPU usage and latency. Notable network architectures supporting RDMA include InfiniBand, used primarily in HPC, and RDMA over Converged Ethernet (RoCE).

While Microsoft stands out as the only major cloud provider that deploys RDMA using networking technologies developed for HPC (RoCE and InfiniBand), this does not imply that RDMA is not widely utilized. In fact, all other leading cloud providers, such as Amazon and Google, have developed their networking technologies (e.g., EFA or 1RMA) to enable RDMA-based data transfers. These stacks share similarities with original RDMA technologies, such as a common low-level API between EFA and RDMA, but also possess notable differences [21, 30]. For example, EFA only supports two-sided primitives, unlike original RDMA technologies that offer

[1] Bisection bandwidth is a measure of how much data the entire network can carry if all servers were sending and transmitting data at once.

one- and two-sided verbs. Other subtleties, like message ordering guarantees, can impact application design. In the RDMA primer, we will provide an overview of the commonalities and differences and known performance characteristics of cloud-based RDMA stacks.

Leveraging RDMA. The database community started to redesign database systems to use RDMA optimally a while ago. We will offer an overview of the initial directions proposed in research and then discuss lessons learned from the last decade of research.

The initial efforts to redesign database systems for RDMA can be divided into two primary directions. First, enhancing existing database components' efficiency using RDMA verbs without modifying the underlying architecture [15, 18]. The second direction involves exploring the idea that RDMA could enable novel database architectures with better scalability than traditional ones [2]. Early papers investigated the use of RDMA in facilitating disaggregated database architectures, which have now become prevalent in cloud environments. This line of work revealed that an architectural shift could result in scalable database systems capable of handling various workloads that were once considered unscalable (e.g., distributed OLTP [25] or graph processing [6]).

Lessons Learned. We summarize the lessons from the first decade of RDMA-based database research along three axes, as follows: (1) DBMS architectures must evolve to optimally leverage RDMA. Although early work indicated the need for architectural changes, discussions about RDMA-optimal architectures continue, with recent proposals suggesting disaggregated structures with coherent caching layers for cloud database systems [29]. (2) A second lesson that we learned "the hard way" is that understanding RDMA and implementing efficient, accurate RDMA-based systems is far from being trivial, as it requires a deep understanding of RDMA's interaction with the local DMA subsystem (e.g., DMA using PCIe). Specifically, designing correct protocols for concurrent RDMA write operations—inherent to numerous database workloads, such as OLTP—poses significant challenges, and many early implementations have proven incorrect. (3) Lastly, improved RDMA abstractions are needed, offering database-centric primitives that simplify RDMA's complexities without compromising performance [4, 23].

4 PROGRAMMABLE DEVICES

As discussed above, modern NICs and switches have become programmable, i.e., some modern devices can accommodate and perform application tasks. There are, however, at least two limitations on what can be offloaded. First, NICs and switches are not general-purpose computers. Most adopt peculiar computing models, which may or may not be able to express the computations worth unloading. Second, this type of equipment has tremendous I/O power but is limited regarding memory size and the length of programs they can support. Even with these restrictions, several database system areas can benefit. Here are some concrete examples:

Semantics-Based Routing. Replicated databases, may allow clients to send queries to secondary servers instead of the primary. This feature reduces the read workload on the latter, but clients risk reading stale data if a secondary server lags on replication. To address this issue, a network switch connecting clients and servers can keep track of database updates—which transactions (packets) were sent to the primary server and which reached the secondaries. The network can, therefore, detect up-to-date secondary servers. A read transaction sent to the primary can be safely redirected to a caught-up secondary. This *semantic packet routing* can alleviate the primary server and scale well with the number of secondaries. Some of us have implemented this logic in a commercial, off-the-shelf programmable switch [27]. In a separate work, we showed that a switch could batch and reorder the transactions sent to servers [10]. The batching greatly amortizes network overhead. The reordering causes transactions with high *affinity* to execute concurrently, bolstering cache hit ratios on the server. These semantic routing techniques have other applications, but let us look into other areas that benefit network support.

Caching and early OLTP execution. What if instead of just routing or manipulating transaction requests, the network would also act on them? For instance, consider a typical OLTP workload. It often contains small transactions that target hot data regions. With enough such transactions the hot data regions become contended, which slows down transaction execution. One alternative is to take advantage of the switch's speed to manage the contended areas. We developed a scenario in which, instead of forwarding a transaction that targets a hot area to the server, the switch would pull that hot area and process the transaction locally instead [8].

Relational and Graph Analytic Acceleration. The benefits discussed above extend to analytical workloads as well. For instance, assume a rack-sized data warehouse where servers process OLAP queries in parallel. The network switch connecting the servers is normally just their data conduit. Instead, we programmed a switch to perform joins and aggregations on behalf of the servers as it receives the data that would be reshuffled in a normal algorithm [12]. By using the switch, we swapped what would have been a sophisticated distributed computation (aggregation on the servers) with a central, simpler one (aggregation on the switch). In the same spirit but in a different work, we taught a switch to process graph data rather than forward it, in the context of a Graph Pattern Mining system [6]. We obtained similar benefits. We say, in such cases, that the network is accelerating the workloads.

Machine Learning Acceleration. This workload category has gained enormous importance and can, unsurprisingly, benefit from network programmability as well. For instance, parameter aggregation plays a crucial role in the training phases of Machine Learning workloads. During such time, the servers must communicate to collectively update the parameters (coefficients) they are calculating in parallel. The communication pattern in question would normally be all-to-all message exchanges, whose quadratic factor creates severe scalability problems. Some of us proposed and implemented this aggregation to use the network instead, implementing these so-called *parameter servers for ML training* on a switch [19].

Lessons Learned. The common factor throughout all these examples is that deploying database system logic in the network cannot be done by simply recompiling software. The computing model in networking hardware is different enough to requires a redesign of the algorithms and data structures involved. Another important aspect we learned is that networking devices are very imbalanced: they are built for massive IO but support only small computations. The topic of when and what can be offloaded in these devices sparked a study of itself that some of us published [16].

5 OPEN PROBLEMS & NEXT STEPS

Despite all the advancements that we described about RDMA and programmable devices, these technologies are still experiencing much progress. Regarding RDMA, some fundamental needs, such as proper guides on using advanced features, are still very much needed and are just starting to appear [31]. The opportunities here go beyond establishing best practices; there has been enough application experience that we are starting to see new forms of communication being suggested. These may appear as higher-level communication primitives [7] or behavior customization motivated by database system use cases [17].

Regarding programmable network devices, we see a similar forward motion. In particular, the difficulties identified in the past about maintaining state on the switch [5] are slowly being addressed. Sometimes, these advancements come in the form of suggested hardware changes [24]. They can also come as new low-level data structures [26, 28] or high-level data services [9]. These advancements, too, are motivated by specific database systems needs such as transaction manipulation [14, 27], database benchmarking [11, 13], and parallel analytics execution [3], just to cite a few.

Lessons Learned. The discipline of modern networking is still in its initial stages, and this means that great systems opportunities lie ahead, e.g., new disaggregated architectures.

6 PRESENTERS' BIOS

The tutorial is offered by seasoned researchers in the area. We can divide them into three groups depending on their origin, starting with those that work in the Industry. Dan Ports works at Microsoft Research and has shown how to use programmable switches to offload tasks such as semantic routing for database replication. His paper on in-network parameter servers is a reference on the field [19]. Theo Jepsen works at Intel and has used programmable switches to offload string search or perform transaction triaging onto the network [9, 10].

The reamaining tutorial co-authors come from Academia. Tobias Ziegler is a Postdoctoral fellow, and Matthias Jasny, and Lasse Thostrup are Ph.D. students at TU Darmstadt, under the leadership of Prof. Carsten Binnig. Tobias Ziegler has extensive experience in RDMA techniques applied to distributed database systems [30]. Matthias Jasny has worked on different approaches to accelerate database subsystems using switches [8]. Lasse Thostrup has experience with high-level abstraction for network programming [23]—the Best Paper Award in SIGMOD'21. Carsten Binnig has pioneered many techniques we discuss in this tutorial, including a seminal paper on modern networks [2].

Rana Hussein and Alberto Lerner are a Ph.D. student and a Senior Researcher, respectively, at the University of Fribourg in Switzerland under the leadership of Prof. Philippe Cudré-Mauroux. Rana Hussein has pioneered techniques to use programmable switches for graph database analytics [6]. Alberto Lerner has worked on several methods to accelerate query processing and database benchmarking with programmable devices [5, 13]. Philippe Cudré-Mauroux is a veteran with many significant contributions to the database field, including the first papers showing that query execution is possible in programmable switches [12].

ACKNOWLEDGMENTS

This work was supported by the the Swiss State Secretariat for Education (SERI) in the context of the SmartEdge EU project (grant agreement No. 101092908), the German Research Foundation priority program 2037 (DFG) under the grant BI2011/2, and the DFG Collaborative Research Center 1053 (MAKI).

REFERENCES

[1] Wei Bai, Shanim Sainul Abdeen, Ankit Agrawal, Krishan Kumar Attre, Paramvir Bahl, et al. 2023. Empowering Azure Storage with RDMA. In *NSDI*.

[2] C. Binnig, A. Crotty, A. Galakatos, T. Kraska, and E. Zamanian. 2016. The End of Slow Networks: It's Time for a Redesign. In *PVLDB*.

[3] M. Blöcher, T. Ziegler, C. Binnig, and P. Eugster. 2018. Boosting Scalable Data Analytics with Modern Programmable Networks. In *DaMoN*.

[4] M. Burke, S. Dharanipragada, S. Joyner, A. Szekeres, J. Nelson, I. Zhang, and D. R. K. Ports. 2021. PRISM: Rethinking the RDMA Interface for Distributed Systems. In *SOSP*.

[5] N. Gebara, A. Lerner, M. Yang, M. Yu, P. Costa, and M. Ghobadi. 2020. Challenging the Stateless Quo of Programmable Switches. In *HotNets*.

[6] R. Hussein, A. Lerner, A. Ryser, L. Bürgi, A. Blarer, and P. Cudré-Mauroux. 2023. GraphINC: Graph Pattern Mining at Network Speed. In *SIGMOD*.

[7] Matthias Jasny, Lasse Thostrup, and Carsten Binnig. 2023. Zero-Sided RDMA: Network-Driven Data Shuffling. In *DaMoN*.

[8] M. Jasny, L. Thostrup, T. Ziegler, and C. Binnig. 2022. P4DB - The Case for In-Network OLTP. In *SIGMOD*.

[9] T. Jepsen, D. Alvarez, N. Foster, C. Kim, J. Lee, M. Moshref, and R. Soulé. 2019. Fast String Searching on PISA. In *SOSR*.

[10] T. Jepsen, A. Lerner, F. Pedone, R. Soulé, and P. Cudré-Mauroux. 2021. In-network Support for Transaction Triaging. In *PVLDB*.

[11] Theo Jepsen, Masoud Moshref, Antonio Carzaniga, Nate Foster, and Robert Soulé. 2018. Life in the Fast Lane: A Line-Rate Linear Road. In *SOSR*.

[12] A. Lerner, R. Hussein, and P. Cudré-Mauroux. 2019. The Case For Network Accelerated Query Processing. In *CIDR*.

[13] A. Lerner, M. Jasny, T. Jepsen, C. Binnig, and P. Cudré-Mauroux. 2022. DBMS annihilator: a high-performance database workload generator in action. In *PVLDB*.

[14] J. Li, E. Michael, N. Kr Sharma, A. Szekeres, and D.R.K. Ports. 2016. Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering.. In *OSDI*.

[15] F. Liu, L. Yin, and S. Blanas. 2017. Design and Evaluation of an RDMA-aware Data Shuffling Operator for Parallel Database Systems. In *EuroSys*.

[16] D. R. K. Ports and J. Nelson. 2019. When Should The Network Be The Computer?. In *HotOS*.

[17] A. Ryser, A. Lerner, A. Forencich, and P. Cudré-Mauroux. 2022. D-RDMA: Bringing Zero-Copy RDMA to Database Systems. In *CIDR*.

[18] W. Rödiger, S. Idicula, A. Kemper, and T. Neumann. 2016. Flow-Join: Adaptive skew handling for distributed joins over high-speed networks. In *ICDE*.

[19] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. R. K. Ports, and P. Richtarik. 2021. Scaling Distributed Machine Learning with In-Network Aggregation. In *NSDI*.

[20] A. Singh et al. 2015. Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. In *SIGCOMM*.

[21] A. Singhvi et al. 2020. 1RMA: Re-envisioning Remote Memory Access for Multi-tenant Datacenters. In *SIGCOMM*.

[22] W Richard Stevens and Thomas Narten. 1990. UNIX network programming. *ACM SIGCOMM Computer Communication Review* 20, 2 (1990), 8–9.

[23] L. Thostrup, J. Skrzypczak, M. Jasny, T. Ziegler, and C. Binnig. 2021. DFI: The Data Flow Interface for High-Speed Networks. In *SIGMOD*.

[24] Y. Yuan et al. 2022. Unlocking the Power of Inline Floating-Point Operations on Programmable Switches. In *NSDI*.

[25] E. Zamanian, C. Binnig, T. Kraska, and T. Harris. 2017. The End of a Myth: Distributed Transaction Can Scale. In *PVLDB*.

[26] L. Zeno et al. 2022. SwiSh: Distributed Shared State Abstractions for Programmable Switches. In *NSDI*.

[27] H. Zhu, Z. Bai, J. Li, E. Michael, D. R. K. Ports, I. Stoica, and X. Jin. 2019. Harmonia: Near-Linear Scalability for Replicated Storage with in-Network Conflict Detection. In *PVLDB*.

[28] H. Zhu, T. Wang, Y. Hong, D. R. K. Ports, A. Sivaraman, and X. Jin. 2022. NetVRM: Virtual Register Memory for Programmable Networks. In *NSDI*.

[29] T. Ziegler, P. Bernstein, V. Leis, and C. Binnig. 2023. Is Scalable OLTP in the Cloud a Solved Problem?. In *CIDR*.

[30] T. Ziegler, D. Mohan, V. Leis, and C. Binnig. 2022. EFA: A Viable Alternative to RDMA over InfiniBand for DBMSs?. In *DaMoN*.

[31] Tobias Ziegler, Jacob Nelson-Slivon, Viktor Leis, and Carsten Binnig. 2023. Design Guidelines for Correct, Efficient, and Scalable Synchronization Using One-Sided RDMA. In *SIGMOD*.