Spark-uDAPL: Cost-Saving Big Data Analytics on Microsoft Azure Cloud with RDMA Networks*

Xiaoyi Lu, Dipti Shankar, Haiyang Shi, and Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering, The Ohio State University
{lu.932, shankar.50, shi.876, panda.2}@osu.edu

***Abstract*—Efficient Big Data analytics on Cloud Computing systems is still full of challenges. One of the biggest hurdles is the unsatisfactory performance offered by underlying virtualized I/O devices such as networks. To address this issue, the modern cloud resource providers (e.g., Microsoft Azure) have deployed high-performance networks, such as Remote Direct Memory Access (RDMA) capable networks in their clouds. However, in this paper, we find that by far, the RDMA networks on Microsoft Azure cannot support either IPoIB or native standard Verbs-based RDMA protocols. Instead, applications need to use the uDAPL (i.e., user Direct Access Programming Library) interface to enable RDMA communication on Azure Cloud, which makes impossible for modern Big Data stacks to leverage these high-performance networks as none of them can support the uDAPL interface yet. To address this issue, we first design an efficient uDAPL-based communication library with the best combinations of uDAPL communication operations. Then, we adapt the designed uDAPL library into the Hadoop RPC ping-pong message passing engine and the Spark Shuffle engine for bulk data transferring. Through our designs, we can improve the performance of Big Data analytics workloads with Hadoop RPC and Spark on RDMA-enabled Azure VMs by up to 90% and 82%, respectively, and save users' cloud resource renting cost by 4.24x. To the best of our knowledge, this is the first work to design a uDAPL-based RDMA communication engine for Big Data analytics stacks (e.g., Spark).**

I. Introduction

Cloud Computing with IaaS (Infrastructure as a Service) offers attractive flexibility to deliver resources by providing a platform for consolidating complex IT resources in a scalable manner. Microsoft Azure is one of the most popular IaaS Cloud Computing platforms in the world. The 'pay-as-you-go' model of clouds makes cloud virtual machines be fully controlled by the users as long as they pay the associated costs of how long the users' applications have been using them. In this model, how fast the users' applications can be fully completed on the clouds will consequently influence how much the users should pay. Thus, application efficiency matters a lot for the Cloud Computing model.

However, efficiently running Big Data applications on Cloud Computing systems is still full of challenges. One of the biggest hurdles in building efficient clouds is the unsatisfactory performance offered by underlying virtualized I/O devices such as networks. To address this issue, the modern cloud resource providers (e.g., Microsoft Azure, Alibaba Cloud) have deployed high-performance networks, such as Remote Direct Memory Access (RDMA) capable networks in their clouds. This motivates us to explore what kind of performance characteristics, programmability, and cost model do these RDMA networks capable clouds provide.

In this paper, we take Microsoft Azure as an example cloud platform with RDMA network support. We first investigate what kind of communication protocols and programming interface it supports. We first follow Azure's official guides [3] to perform experiments on RDMA-enabled VMs. Based on [4], we can run Intel MPI based parallel programs on RDMA-enabled VM instances, such as A8 and A9. Figure 1 shows the results we have taken on two A8 VM instances on Microsoft Azure Cloud.

Through these experiments, we find that by far, RDMA-enabled virtual machines on Azure cannot support either IP-over-IB (i.e., IPoIB) protocol or the full set of standard verbs [13] interface, which is the typical programming interface for using RDMA devices. To make the RDMA device work on Azure, we need to use a programming interface called uDAPL (i.e., user Direct Access Programming Library) [9], which is a user-space direct access framework to enable direct memory access over RDMA-capable networks. Intel MPI supports uDAPL channel which makes it achieve desired performance in terms of latency and bandwidth on the Azure RDMA network. In the community, another alternative high-performance MPI library, called MVAPICH2, also supports the uDAPL interface.

In our study of latency performance, Figure 1 demonstrates that uDAPL-based MPI libraries (e.g., Intel MPI and MVAPICH2) improve the latency performance up to 98.5%, 98.3%, and 98.1% for small messages (Figure 1a), medium messages (Figure 1b), and large messages (Figure 1c), respectively, compared with using the traditional Ethernet interface. In the meantime, our experiments of bandwidth performance in Figure 1d show that uDAPL significantly improves the bandwidth performance by up to 53.7x compared to the default Ethernet card in the A8 instance. This is because the default Ethernet card on A8 VM is just a virtualized 1Gig Ethernet and the virtualized RDMA network card is about 32Gbps (i.e., QDR).

These numbers clearly show that there are good opportunities to gain performance benefits for Big Data analytics workloads, which typically have huge data movement requirements, by running them over the fast RDMA-based network channel on Azure. Unfortunately, so far, the default Big Data analytics stacks such as Apache Hadoop and Spark can only run on top of RDMA networks with IPoIB protocol. The community distributions of RDMA-Hadoop [8, 10, 15] and RDMA-Spark [11, 12] can only support native Verbs-based

*This research is supported in part by National Science Foundation grants #IIS-1636846 and #CCF-1822987. The Microsoft Azure Cloud resources used in this paper are supported by Microsoft Azure Sponsorship.

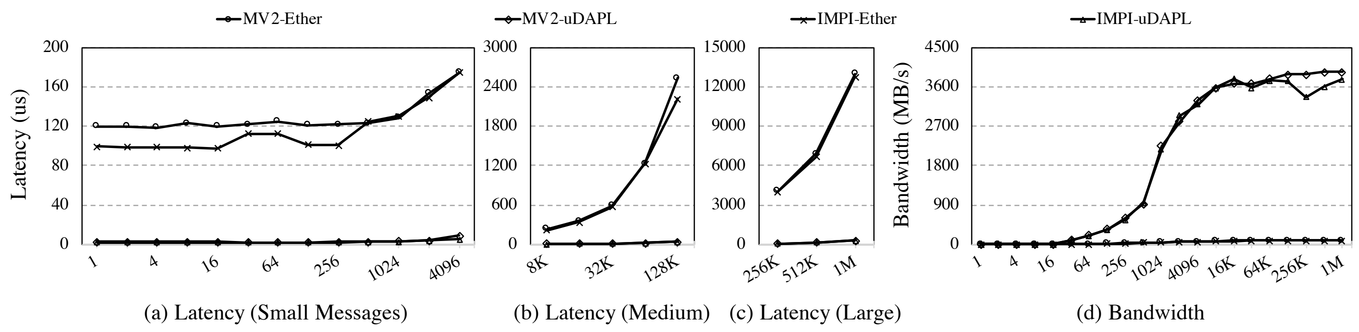


Fig. 1: Performance Gains by Using uDAPL-based RDMA Network on Microsoft Azure Cloud. Tested with Intel MPI and MVAPICH2 over A8 VM Instances on Azure. Note that the RDMA devices on Azure cannot support IPoIB protocol so far. This is the reason why there is no IPoIB numbers in the graphs.

interfaces to utilize RDMA functions. Neither of them can be supported efficiently on current generation Azure cloud.

In addition, to the best of our knowledge, none of these Big Data analytics stacks (such as Hadoop and Spark) can support uDAPL interface yet, which means there is no way for a broad range of Big Data analytics applications to run efficiently over these RDMA-enabled VM instances on Azure. There is a clear need to design an efficient communication substrate to enable high-performance RDMA protocols for Big Data analytics applications running on Azure.

To address this need, in this paper, we first design an efficient uDAPL-based communication library with the best combinations of uDAPL communication operations, such as DAPL-based two-sided (Send/Recv) and one-sided RDMA operations (Read/Write). Then, we adapt our designed uDAPL library into the Hadoop RPC ping-pong message passing engine and the Spark Shuffle engine for bulk data transferring.

Performance evaluations of Hadoop RPC show that, compared with Ethernet, our proposed design can improve the latency performance by up to 71.9%, 90.1%, and 89.9% for small, medium, and large messages, respectively, and significantly accelerate the throughput performance of small and large messages by up to 4.5x and 10.6x, respectively. Performance evaluations of Spark show that, our proposed design can achieve 47-81% and 58-82% performance improvement for GroupBy and SortBy, respectively, compared with running Spark over Ethernet on 9-VM based virtual clusters on Microsoft Azure.

We further calculate the costs of running Big Data analytics jobs on different Microsoft Azure VM instance types. We find that with our proposed designs, we are able to significantly reduce the users' cost up to 4.24x by achieving much shorter job execution times on the cloud.

To the best our knowledge, this is the first design for accelerating Spark-like Big Data analytics stacks with native uDAPL interface over RDMA networks. Our designs can accelerate Big Data processing over bare-metal and virtualized RDMA networks with uDAPL support in clouds.

II. Background & Related Work

In this section, we present a brief overview of the necessary background information and some related work.

A. uDAPL Overview

Remote Direct Memory Access (RDMA) capable networks, such as InfiniBand (IB) [7], Virtual Interface Architecture (VIA) [6], iWarp [16], do not commonly provide a common set of APIs. Towards overcoming this issue, DAT (Direct Access Transport) Collaborative [5] defined the DAPL interface, providing a common interface for different RDMA-capable interconnects. Efforts have been made to standardize APIs for both the kernel-space (kDAPL) and the user-space (uDAPL), as shown in Figure 2.

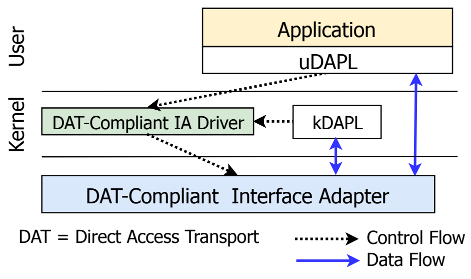


Fig. 2: DAPL User Model

Specifically, uDAPL (user Direct Access Programming Library) [9] defines a set of standard and lightweight set of APIs that can be used to design transport- and platform-independent RDMA-aware portable user-applications across different RDMA interconnects. To support multiple vendors, uDAPL creates a registry between the consumer and the vendor-specific software. It supports reliable connections (RC) using a client/server-like connection model via endpoints (EPs). It also enables both two-sided (Send/Receive) and one-sided (Read/Write) RDMA data movement operations. They can be issued by placing the appropriate operations onto send/receive queues for processing, and either poll the completion queue or wait for an event for signaling the operation completion. The applicability of uDAPL-based designs for scientific middleware, such as Message Passing Interface (MPI) [17, 18] has been studied. However, not much research has been carried out to see how uDAPL can benefit Big Data middleware (e.g., Hadoop, Spark, etc.) over modern HPC clouds.

Features \ VM type	A8	A9	A10	A11	H16r	H16mr
Node Config. (Cores / RAM)	8 cores 56 GB	16 cores 112 GB	8 cores 56 GB	16 cores 112 GB	16 cores (Haswell) 112 GB	16 cores (Haswell) 224 GB
Network Type	IB QDR (uDAPL)	IB QDR (uDAPL)	1 GbE	1 GbE	IB FDR (uDAPL)	IB FDR (uDAPL)
Est. Price ($/hour)	1.466	2.933	1.173	2.346	3.859	5.169

TABLE I: Microsoft Azure VM instances (QDR = 32 Gbps, FDR = 56 GBps); based on [2] for East U.S.

B. Microsoft Azure Cloud with RDMA Networks

Microsoft Azure [3] provides VM instances, such as, (1) A8, A9, (2) H16r, H16mr, and (3) NC24r, which have RDMA connectivity enabled via InfiniBand networks. The A-series instances are general purpose and H-series are provisioned specifically for HPC applications. These RDMA capabilities are introduced to particularly boost the scalability and performance of certain MPI applications. Details for some of these instances are furnished in Table I; the costs are based on East U.S. region (in US dollars) [2]. It is vital to note that TCP/IP protocol over IB (IPoIB) [1] is not supported on Microsoft Azure. Native verbs library [13] is also not directly accessible. Only RDMA over IB is supported via the uDAPL library. This makes a majority of the TCP/IP-over-IB and RDMA-based native-IB Big Data middleware (e.g., Apache Spark [12, 19]) incompatible with these RDMA-aware VM instances on Azure, especially in terms of maximizing cost-performance.

C. RDMA-aware Designs for Big Data Middleware

With the increased need to design Big Data middleware capable that can deliver high-performance on modern HPC clusters, several studies have been devoted to exploring the advantages of re-designing their components to be RDMA-aware. RDMA-aware designs for Apache Spark [12, 14], which is the basis for this paper, have shown tremendous performance improvement and scalability. However, they only support native IB via the verbs interface.

With the increasing usage of Big Data frameworks in the cloud, there is a vital need for supporting these RDMA-based middleware over uDAPL. In this paper, we address this missing link by making the attempt to integrate RDMA-aware Spark and Hadoop RPC with uDAPL to maximize cost-efficient performance in the Azure cloud environment.

III. Proposed Design

In this section, we present our proposed uDAPL-based designs.

A. uDAPL-based RDMA Protocol Design and Study

To co-design RDMA-aware middleware with uDAPL, we first perform detailed analysis of the performance characteristics of different RDMA-based communication protocols with uDAPL operations. Based on supported primitives in uDAPL, we can design the following three basic communication protocols as shown in Figure 3. Note that all these three protocols are generic for all message sizes, while they can be used differently in different communication scenarios.

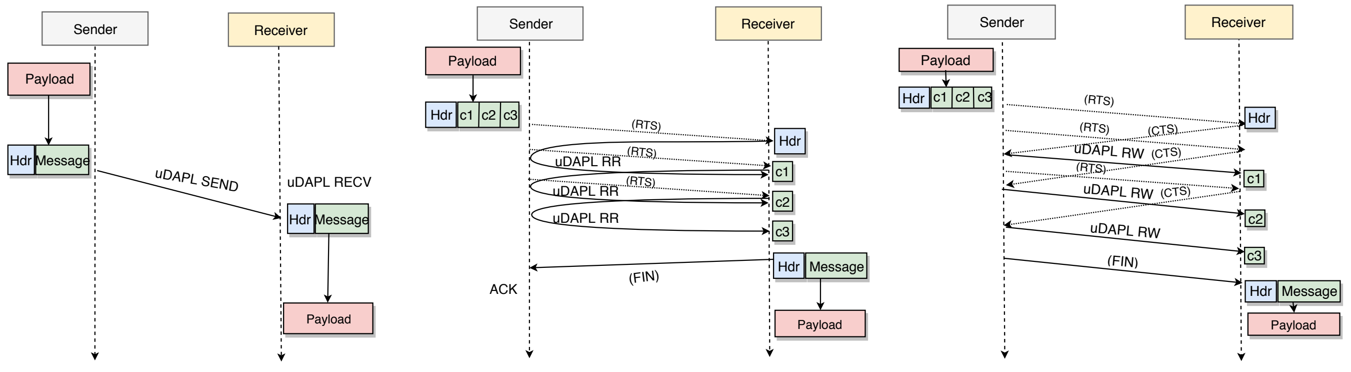
(1) **uDAPL Send/Receive protocol (*uDAPL-SR*)**: In this protocol, as shown in Figure 3(a), the sender will initial the send request by copying user data into the header body (in an eager manner) and the receiver needs to post receive operation, which means this is a two-sided communication protocol and both sides need to be involved during the communication process. This protocol can be used to perform small message transfers since it can avoid handshaking overhead. The corresponding primitives for implementing this protocol with uDAPL include `dat_ep_post_send` and `dat_ep_post_recv`.

(2) **uDAPL RDMA Read Protocol (*uDAPL-RR*)**: This protocol employs one-sided RDMA Reads for data transfers as shown in Figure 3(b). To avoid pre-registering many buffers for RDMA operations on both sides, we need to first send a 'Request-To-Send (RTS)' message to the receiver side to notify where the data can be read. Then, the receiver will perform a follow-up RDMA Read operation to fetch the data. For a high-performance design, we typically need to support message chunking and overlapping in the protocol design. The corresponding primitive for implementing this protocol with uDAPL is `dat_ep_post_rdma_read`.

(3) **uDAPL RDMA Write Protocol (*uDAPL-RW*)**: As shown in Figure 3(c), this protocol employs one-sided RDMA Writes for data transfers. Different than the RR protocol, the RDMA Write operation is initiated by the sender side with the received target address in the 'Clear-To-Send (CTS)' message. This protocol can also be designed to support chunking and overlapping. The corresponding primitive for implementing this protocol with uDAPL is `dat_ep_post_rdma_write`. Both uDAPL-RR and uDAPL-RW protocols are good for transferring large messages since it can achieve zero-copy data transfers.

Figure 4 presents the point-to-point latency measured with the three uDAPL protocols identified above. One important insight of our study is that uDAPL-SR outperforms other two protocols for relatively small messages (messages less than or equal to $32KB$ on $A8$). More specifically, we see only around 4–5us latency for the uDAPL-SR protocol to send less than 2KB payloads. Note that 2KB is the default MTU (Maximum Transmission Unit) size on A8 VMs. However, both uDAPL-RR and uDAPL-RW protocols need around twice of the time to perform the short message transfers, which is mainly because of the handshaking overhead.

While for the medium and large message transfers, the uDAPL-RW and uDAPL-RR protocols can outperform uDAPL-SR. This is mainly due to the message copying overhead in the uDAPL-SR protocol. The copy overhead will be significantly increased when the message becomes larger.



(a) Send/Receive Protocol (uDAPL-SR) (b) RDMA-Read Protocol (uDAPL-RR) (c) RDMA-Write Protocol (uDAPL-RW)

Fig. 3: uDAPL-aware Data Movement Protocols for Spark/RPC

Motivated by this, we design an optimized protocol as the fourth protocol:

(4) **uDAPL-OPT**, that uses a hybrid uDAPL-SR + uDAPL-RW/RR approach for message sizes varying from small to large. Based on this, from Figure 4, we further demonstrate that by combining the benefits of uDAPL-SR and uDAPL-RW/RR, uDAPL-OPT can always conduct communication in the most efficient manner.

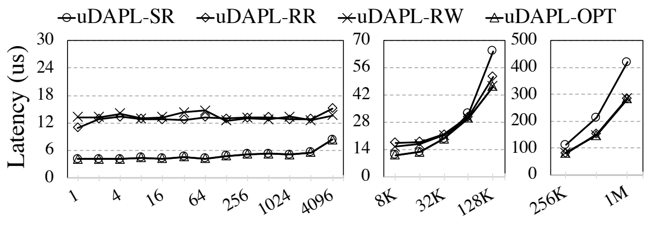


Fig. 4: Performance with different uDAPL RDMA Protocols. X-axis represents payload size (Unit: Byte).

B. *Adapting uDAPL into Hadoop RPC and Spark*

Figure 5 presents an architecture overview of RDMA-based Apache Spark running over Hadoop in a cloud environment with InfiniBand or Ethernet networks. By default, Spark fetches remote blocks as needed and the operations to which are initiated by the `ShuffleManager` layer.

As shown in Figure 5, the Spark `ShuffleManager` and Hadoop RPC engine over Azure VM instances, such as A10/A11, use the 'Default TCP/IP path' for performing data communication. On the other hand, for RDMA-capable instances, such as A8/A9, Spark and Hadoop can utilize the 'RDMA path' by employing the native uDAPL-based RDMA communication engine that we propose in this paper. We design an interface for Spark and RPC engines that can be used to offload data transfers in the Scala/Java layer to the native uDAPL engine that is written in native C, via the Java Native Interface (JNI).

We choose Spark and Hadoop RPC as example systems because Spark mostly involves bulk data transfer patterns with large messages, while Hadoop RPC follows ping-pong message transfer patterns with small or medium messages. As RPC workloads are more latency-oriented due to their ping-pong nature, we use the uDAPL-SR + uDAPL-RW hybrid protocol as our optimized uDAPL-OPT protocol.

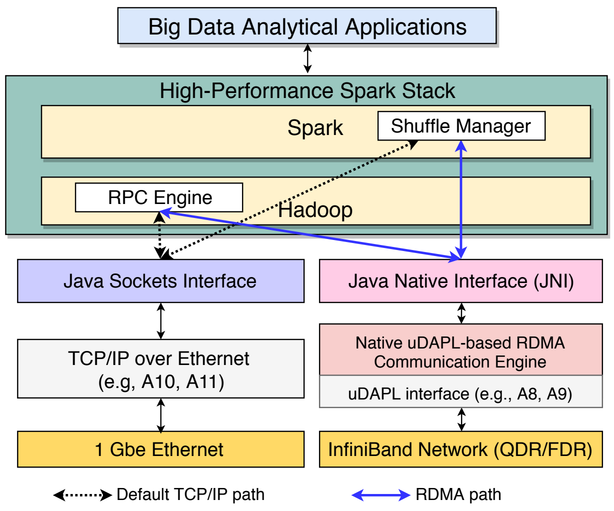

Big Data Analytical Applications
High-Performance Spark Stack
Spark
Shuffle Manager
RPC Engine
Hadoop
Java Sockets Interface
Java Native Interface (JNI)
TCP/IP over Ethernet
(e.g, A10, A11)
Native uDAPL-based RDMA Communication Engine
uDAPL interface (e.g., A8, A9)
1 Gbe Ethernet
InfiniBand Network (QDR/FDR)
Default TCP/IP path
RDMA path

Fig. 5: Architecture of uDAPL-based RDMA-enhanced Apache Spark and Hadoop RPC

On the other hand, for large block transfers in Spark, we can not use the eager-based transfer since the copy overhead of large block will be high. Thus, RDMA-based zero-copy data transfer is chosen for large block data transfers. As Spark shuffle needs to be co-ordinated among multiple Spark instances across multiple VMs, it is inclined to be more throughput-oriented. As a result, we can enable better performance at the shuffle server instance by offloading more communication tasks at the shuffle client. By leveraging uDAPL-RR, the server can be near bypassed for the actual block transfers; thus facilitating a high-throughput approach. Hence, we use the uDAPL-SR + uDAPL-RR hybrid protocol as our optimized uDAPL-OPT protocol for Spark.

These optimizations while used as default, are flexible enough to be employed in any hybrid combination to cater to the user application's performance and data movement patterns. We enable chunking-based uDAPL-RR and uDAPL-RW, as presented in Figures 3(b) and 3(c), respectively. The chunk size can be tuned to optimal based on the underlying network interconnect (e.g., QDR, FDR, EDR, etc.).

IV. Performance Evaluation

In this section, we present the detailed performance evaluations.

A. Experimental Setup

Table II shows the specifications of A8 and A10 VM instances used in this paper. We choose A8 and A10 to compare performance since their hardware configurations are very similar and the main difference between them is that A10 does not have the RDMA network device equipped. We have used up to nine VMs (i.e., one RPC server VM and eight client VMs for Hadoop experiments; One master VM and eight worker VMs for Spark experiments) in the following experiments.

	Specification
Processor	Intel(R) Xeon(R) CPU E5-2670 0 @ 2.60GHz
RAM (DDR)	56 GB
Interconnect	IB QDR 32Gbps on A8, 1 GbE on A10
Storage	HDD 500GB
OS	CentOS 7.3
Hypervisor	Microsoft Hyper-V
DAPL	Linux Integration Services (LIS) Driver 4.2.4 + dapl-2.1.10
JDK	OpenJDK 1.8.0
Hadoop	RDMA-Apache-Hadoop-2.x 1.3.5
Spark	RDMA-Spark 0.9.5

TABLE II: Testbed Specifications

B. Performance Evaluation of Apache Hadoop RPC

In this section, all evaluations of Apache Hadoop are conducted with six handler threads within RPC engine. We choose six threads based on tuning. On an A8 VM, we only have eight cores, which results in six handlers giving the best performance for RPC engine.

In our latency performance evaluation on top of Hadoop RPC, as depicted in Figure 6, when compared with A10-Ether, A8-uDAPL reduces the latencies of transferring small, medium, and large messages by up to 71.9%, 90.1%, and 89.9%, respectively.

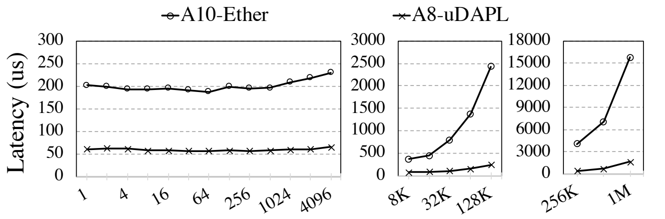


Fig. 6: Latency Performance Evaluation of Hadoop RPC. X-axis represents payload size (Unit: Byte).

In evaluating the throughput performance of Hadoop RPC over A10-Ether and A8-uDAPL, we conduct two representative experiments to illustrate how well A8-uDAPL works. In the first experiment, we choose a payload size of $512B$ to represent small messages. Figure 7a shows that A8-uDAPL accelerates small message passing by 2.7x to 4.5x with variant scalability (one to 64 clients as shown on the x-axis). While for evaluating the impact on large messages, a payload size of $32KB$ is chosen as a representative. Since in RPC framework, a payload with size of $32KB$ becomes a message with size larger than $32KB$ because of serialization. As demonstrated in Figure 7b, A8-uDAPL improves throughput by 5.8x to 10.6x compared to A10-Ether with different scales.

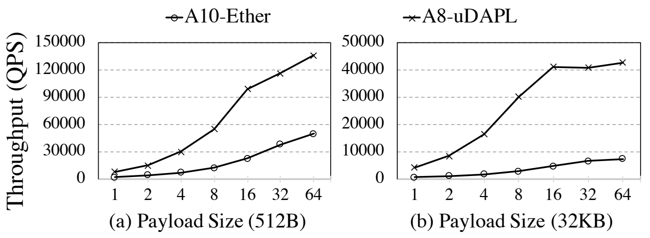


Fig. 7: Throughput Performance Evaluation of Hadoop RPC. X-axis represents number of clients.

These numbers show that our designed uDAPL communication protocols and engine can perfectly fit with ping-pong type of message passing pattern in RPC frameworks.

C. Performance Evaluation of Apache Spark

The performance evaluation of Apache Spark is conducted with the following settings: nine VMs, $48GB$ executor memory, $2GB$ daemon memory, and 8 worker cores per Spark executor.

Figure 8 shows that our proposed design (*Spark-uDAPL-A8*) reduces the execution time of Spark GroupBy significantly compared with running Spark over Ethernet on $A10$ (*Spark-Def-1GigE-A10*). For the experiment with two tasks per VM instance, as shown in Figure 8a, the improvements we achieve are from 60% to 81% with different workloads (8GB to 32GB as shown on the x-axis). Figure 8b depicts that execution time is reduced by 47% to 74% if four tasks are employed on each VM instance. Figure 8a and Figure 8b show that employing more tasks on the same VM is beneficial to performance, if we compare jobs with the same workload but a different number of tasks on each VM. As aforementioned, there are eight cores on both $A8$ and $A10$, thus utilizing more CPU cores appropriately definitely performs better.

For evaluation of SortBy, Spark-uDAPL-A8 reduces the execution time by 59% to 82% compared with Spark-Def-1GigE-A10 when running two tasks per instance (shown in Figure 9a). Figure 9b illustrates that utilizing more CPU cores is also helpful to improve the performance of Spark SortBy. With four tasks per VM, the improvements of Spark-uDAPL-A8 are from 58% to 81% with varied workloads (16 GB to 64 GB as shown on the x-axis).

These numbers demonstrate that our designed uDAPL communication protocols and engine can clearly accelerate Spark shuffle tasks with the bulk data transferring communication pattern.

D. Cost-Efficiency with uDAPL-based RDMA-aware designs

RDMA networks can deliver better application performance than the traditional low-speed networks on clouds. This has been proved by the results in this paper. However, since RDMA network based VM instances cost more money than the traditional network based VM instances, cloud users may doubt that "should we use the RDMA-enabled VM instances or regular ones?" This leads to an interesting research problem that renting which type of VMs can lead to more cost-efficient plans.

To further demonstrate the gains that can be obtained via uDAPL-based RDMA-aware designs with Big Data analytical

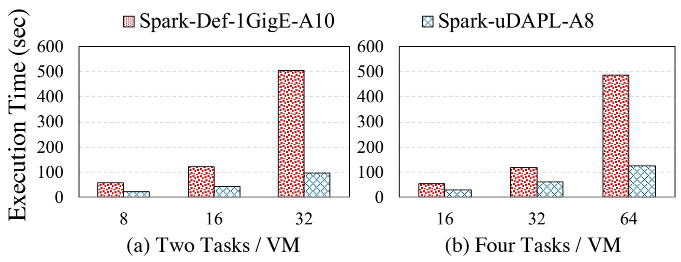


Fig. 8: Performance Evaluation of Spark GroupBy. X-axis represents workload size (Unit: GB).

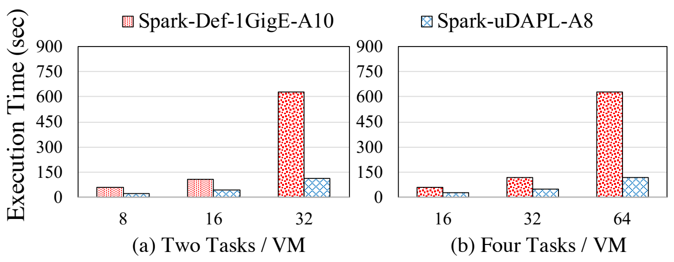


Fig. 9: Performance Evaluation of Spark SortBy. X-axis represents workload size (Unit: GB).

frameworks, such as Spark, we study the cost of running 16-64 GB shuffle-intensive GroupBy and SortBy workloads on the nine Azure-VMs (i.e., one master VM and eight worker VMs) based Spark cluster. We contrast the cost-performance characteristics of the non-uDAPL-RDMA instances based on Azure's A10 VMs that run over 1GigE with the uDAL-RDMA-optimized Azure A8 instances running over IB QDR (32 Gbps).

We calculate the average cost-per-Spark-job run based on the $-per-hour furnished in Table I. From Table III, we observe that we can achieve cost-efficiency of about 1.51x to 4.24x with our high-performance uDAPL-RDMA designs for Spark, and cloud customers can save more money if they process and analyze more data with our proposed Spark-uDAPL on Microsoft Azure.

		GroupBy			SortBy		
Shuffle Size (GB)		16	32	64	16	32	64
Cost	A10-Ether	0.161	0.349	1.422	0.182	0.354	1.835
($)	A8-uDAPL	0.106	0.223	0.454	0.095	0.175	0.432
Cost Savings		1.52X	1.56X	3.13X	1.91X	2.02X	4.24X

TABLE III: Cost-efficiency of non-uDAPL-RDMA (A10) vs. uDAPL-RDMA (A8) with Spark GroupBy and SortBy

Thus, we demonstrate that we can enable both high-performance and cost-effective designs for Big Data analytical workloads on the Azure cloud with RDMA-capable VM instances over uDAPL.

V. Conclusion

In this paper, we present a detailed performance analysis on uDAPL-based communication operations and protocols. We propose designs to incorporate high-performance uDAPL-based RDMA substrate into Apache Spark stack as well as Hadoop RPC engine, which can enable Big Data analytics workloads efficiently running over uDPAL interface based RDMA networks. Our evaluation results show that compared to the default Spark running over virtualized Ethernet network on Azure cloud, our proposed design can achieve up to 82% performance improvement for Spark benchmarks (e.g., GroupBy, SortBy), which can lead to 4.24x cost efficiency for end users.

In the future, we plan to investigate how can we easily support more Big Data stacks by using our proposed uDAPL-based RDMA communication library and protocols.

References

[1] "IP over InfiniBand Working Group," `http://www.ietf.org/html.charters/ipoib-charter.html`.

[2] "Microsoft Azure: Cloud Services pricing," https://azure.microsoft.com/en-us/pricing/details/cloud-services/.

[3] "Microsoft Azure: High performance compute VM sizes," https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-hpc.

[4] "Microsoft Azure: Set up a Linux RDMA cluster to run MPI applications," https://docs.microsoft.com/en-us/azure/virtual-machines/linux/classic/rdma-cluster.

[5] D. Collaborative, "uDAPL: User Direct Access Programming Library Version 1.2," 2004.

[6] D. Dunning, G. Regnier, G. McAlpine, D. Cameron, B. Shubert, F. Berry, A. M. Merritt, E. Gronke, and C. Dodd, "The Virtual Interface Architecture," *IEEE Micro*, no. 2, pp. 66–76, 1998.

[7] InfiniBand Trade Association, "Infiniband," http://www.infinibandta.org/.

[8] N. S. Islam, X. Lu, M. W. Rahman, D. Shankar, and D. K. Panda, "Triple-H: A Hybrid Approach to Accelerate HDFS on HPC Clusters with Heterogeneous Storage Architecture," in *15th IEEE/ACM Intl. Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2015.

[9] J. Lentini, V. Pham, S. Sears, and R. Smith, "Implementation and Analysis of the User Direct Access Programming Library."

[10] X. Lu, N. S. Islam, M. W. Rahman, J. Jose, H. Subramoni, H. Wang, and D. K. Panda, "High-Performance Design of Hadoop RPC with RDMA over InfiniBand," in *The Proceedings of IEEE 42nd International Conference on Parallel Processing (ICPP)*, France, October 2013.

[11] X. Lu, M. Rahman, N. Islam, D. Shankar, and D. Panda, "Accelerating Spark with RDMA for Big Data Processing: Early Experiences," in *2014 IEEE 22nd Annual Symposium on High-Performance Interconnects (HOTI)*, 2014.

[12] X. Lu, D. Shankar, S. Gugnani, and D. K. D. Panda, "High-performance Design of Apache Spark with RDMA and its Benefits on Various Workloads," in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 253–262.

[13] OpenFabrics Alliance, https://www.openfabrics.org/.

[14] OSU NowLab, "The High-Performance Big Data Project," http://hibd.cse.ohio-state.edu.

[15] M. W. Rahman, X. Lu, N. S. Islam, and D. K. Panda, "HOMR: A Hybrid Approach to Exploit Maximum Overlapping in MapReduce over High Performance Interconnects," in *International Conference on Supercomputing (ICS)*, Munich, Germany, 2014.

[16] RDMA Consortium, "Architectural Specifications for RDMA over TCP/IP," http://www.rdmaconsortium.org/.

[17] J. Singh and Y. Sonawane, "Multiplexing Endpoints of HCA for Scaling MPI Applications: Design and Performance Evaluation with uDAPL," in *2010 IEEE International Conference on Cluster Computing*. IEEE, 2010, pp. 136–145.

[18] A. Vishnu, P. Gupta, A. R. Mamidala, and D. K. Panda, "A Software based Approach for Providing Network Fault Tolerance in Clusters with uDAPL Interface: MPI Level Design and Performance Evaluation," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. ACM, 2006, p. 85.

[19] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*, Boston, MA, 2010.