Communicating Efficiently on Cluster-Based Remote Direct Memory Access (RDMA) over InfiniBand Protocol

applied sciences

MDPI

Article

Communicating Efficiently on Cluster-Based Remote Direct Memory Access (RDMA) over InfiniBand Protocol

Masoud Hemmatpour *, Bartolomeo Montrucchio and Maurizio Rebaudengo

Dipartimento di AUTOMATICA E INFORMATICA (DAUIN), Politecnico di Torino, 10129 Torino, Italy; bartolomeo.montrucchio@polito.it (B.M.); maurizio.rebaudengo@polito.it (M.R.)

* Correspondence: masoud.hemmatpour@polito.it

Received: 21 September 2018 ; Accepted: 17 October 2018; Published: 24 October 2018

Abstract: Distributed systems are commonly built under the assumption that the network is the primary bottleneck, however this assumption no longer holds by emerging high-performance RDMA enabled protocols in datacenters. Designing distributed applications over such protocols requires a fundamental rethinking in communication components in comparison with traditional protocols (i.e., TCP/IP). In this paper, communication paradigms in existing systems and new possible paradigms have been investigated. Advantages and drawbacks of each paradigm have been comprehensively analyzed and experimentally evaluated. The experimental results show that writing the requests to server and reading the response presents up to 10 times better performance comparing to other communication paradigms. To further expand the investigation, the proposed communication paradigm has been substituted in a real-world distributed application, and the performance has been enhanced up to seven times.

Keywords: high performance; parallel programming; distributed programming; datacenter

1. Introduction

The ever-increasing technological breakthroughs in hardware, software and networking led to the growth of distributed applications. Distributed applications require to communicate among components to access remote resources. The underlying challenges in distributed landscape have changed many times over time. Nowadays, an important aspect in designing a modern real-world distributed application is the communication model to connect its various components. This paper focuses on communication in distributed applications to highlight the advantages and drawbacks of each communication paradigm.

Communications over TCP/IP protocols were unfit for high speed data transfer because of significant CPU and memory usage. This constraint emerged a new category of network fabrics, using a technology known as *Remote Direct Memory Access* (*RDMA*). RDMA unwraps the boundary of each machine by creating a virtual distributed shared memory among connected nodes in a cluster. It allows direct memory access from the memory of one host to the memory of another one. Examples of such network fabrics are *internet Wide-Area RDMA Protocol* (*iWARP*) [1], *RDMA over Converged Ethernet* (*RoCE*) [2], and *InfiniBand* [2]. In contrast with conventional protocols, RDMA implements the entire transport logic in network interface card—commonly known as *Host Channel Adapter* (*HCA*)—to boost the performance.

Recently, modern clusters are holding sufficient amount of main memory to store data applications in a distributed manner, thus they are embracing RDMA technology in their network infrastructures to speed up the distributed memory access. RDMA is being exploited in different contexts such

as network file system [3], distributed deep learning [4–6], and in-memory data storage [7–11]. Exploiting RDMA requires revisiting the design of existing distributed systems. Basically, RDMA can be exploited to enhance mainly the communication component of an existing application or to design a novel system based on it [7–11].

This paper is motivated by the seeming contrast between the type of communications in RDMA-based distributed applications. It explains the fundamental differences in network traffic, type of connection, and synchrony in several communication paradigms. To address the generality issue, the core communications are evaluated.

This paper strives to answer important research questions for the design of distributed datacenter applications on modern clusters, presenting the following contributions:

- A comprehensive discussion on challenges, opportunities and suitability of RDMA protocols
- The proposal of some optimization techniques to improve the performance
- The investigation on an optimal communication design for exchanging requests/responses in RDMA systems
- The evaluation of the proposed communication paradigm in a real-world distributed application

In the proposed communications, RDMA is treated as a means for sending and receiving requests and responses. All communications are implemented and evaluated, and results show that writing the requests to server's memory by RDMA write and reading the response through RDMA read presents up to 10 times better performance comparing to other communication paradigms.

The remainder of this paper is organized as follows. A background on RDMA mechanism is described in Section 2. RDMA opportunities, challenges, and suitability is provided in Section 3. A comprehensive analysis of communication paradigms, and the key performance challenges of how to exploit RDMA are presented in Section 4. Experiments and results are discussed in Section 5. A real-world distributed application is evaluated in Section 6. Finally, conclusions are drawn in Section 7.

2. Background

Although iWARP, RoCE, and InfiniBand protocols provide a unique set of operations, however adopting an appropriate protocol requires the awareness of advantages and drawbacks of each protocol. The iWARP [1] is a complex protocol published by *Internet Engineering Task Force* (*IETF*) which only supports reliable connected transport (see Section 4.2) [12]. It was designed to convert TCP to RDMA semantics which limits the efficiency of iWARP products. On the contrary, RoCE is an Ethernet-based RDMA solution published by *InfiniBand Trade Association* (ITA) supporting reliable and unreliable transports. InfiniBand is an advanced network protocol with low latency and high bandwidth commonly used in commodity servers.

The *OpenFabrics Alliance* (*OFA*) publishes and maintains user-level *Application Programming Interface* (*API*) for the RDMA protocols (i.e., iWARP, RoCE, and InfiniBand) [13] along with useful utilities, called *OpenFabrics Enterprise Distribution* (*OFED*).

As Figure 1 shows, RDMA allows an application to queue up a series of requests to be executed by HCA. Queues are created in pairs, called *Queue Pair* (*QP*), to send and receive operations. An application submits a *Work Queue Element* (*WQE*) on the appropriate queue. Then, channel adapter executes WQEs in the FIFO order on the queue. Each WQE can work with multiple scatter/gather entries to read multiple memory buffers and to send them as one stream and write them to multiple memory buffers. When the channel adapter completes a WQE, a *Completion Queue Element* (*CQE*) is enqueued on a *Completion Queue* (*CQ*). RDMA provides various work requests consisting of different operations such as:

- `SEND/RECV` sends/receives a message to/from a remote node
- *Fetch-And-Add* (`FAA`) atomically returns and increments the value of a virtual memory location in the remote machine

- *Compare-and-Swap* (`CAS`) atomically compares the value of a virtual memory address with a specified value and if they are equal, a new value will be stored at the specified address
- `READ/WRITE` reads/writes a data from/to a remote node exploiting the *Direct memory Access* (*DMA*) engine of the remote machine (i.e., bypassing the CPU and kernel)

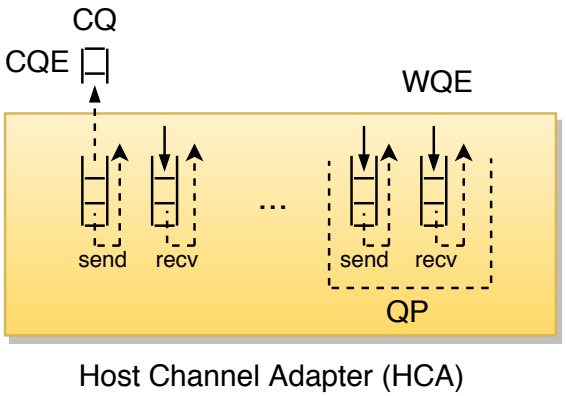

Figure 1. InfiniBand Architecture.

Figure 2 shows a possible RDMA communication between Machines A and B. Firstly, the user application issues a send request in the step 1 to communicate with the channel adapter through CPU in the step 3. Kernel space operation in the step 2 is used only for starting an RDMA connection, and there is not any operation or buffering when connection is set up. Furthermore, the existence of the fourth operation depends on the request type. If the data are inlined (see Section 4.3), the HCA does not need to perform an extra DMA operation to read the payload from the user-space memory. Afterwards, the request is enqueued in the send queue and is waiting for its turn to be processed by the *Network Processor* (*NP*). When the message is received by Machine B, it can be performed on a memory address (step 4) without any reply to Machine A or with a reply to Machine A (step 5).

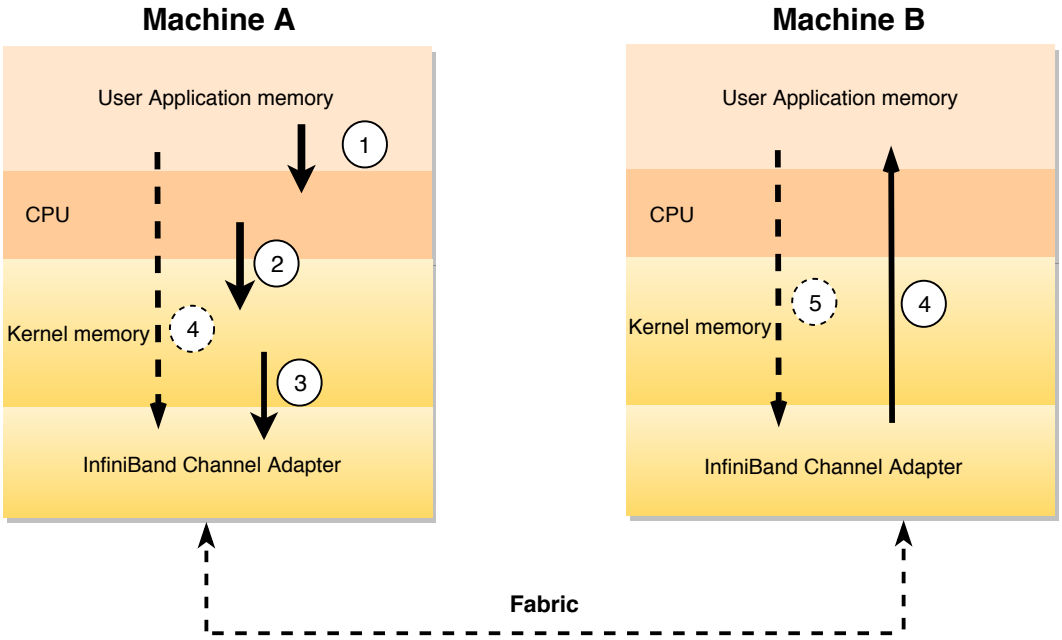

Figure 2. RDMA communication cutting view.

3. Opportunities, Challenges and Suitability

The next generation of high-performance networks exploiting RDMA are giving rise to extremely important opportunities and challenges for these new protocols. These issues as well as suitability of RDMA-enabled networks are described in the following.

3.1. Opportunities

RDMA-enabled protocols provide important features to accelerate data transfer without network software stack involvement (zero-copy), context switches (kernel bypass), and any intervention (e.g., cache pollution, CPU cycles) of the remote processor (no CPU involvement). These features allow RDMA applications to saturate the network using few cores, unlike traditional distributed systems. RDMA applications use CPU in an efficient way which can be beneficial in shared CPU environment (i.e., cloud solution). Furthermore, these features enrich RDMA-enabled application to achieve the highest communication throughput, in particular for small messages. RDMA provides a distributed shared memory over a reliable loss-less infrastructure through hardware-based re-transmission of lost packets [8].

Compatibility with legacy protocols is an important capability of RDMA-enabled protocols. For example, iWARP and RoCE are designed to be compatible with legacy Ethernet protocol, and InfiniBand supports legacy socket applications through *IP over InfiniBand* (*IPOIB*).

3.2. Challenges

Although RDMA provides higher performance compared with traditional protocols, the dilemma between RDMA performance achievement and redesign cost of an existing application according to RDMA semantic can be a challenging task. Redesign can be as broad as the whole system or limited to the communication component with the RDMA send/receive counterpart. It should be noted that the performance achievement by the latter approach is restricted.

One of the challenging problems in an RDMA-enabled distributed application is orchestrating local and remote memory accesses, since these are transparent to each other. Practically, synchronizing these concurrent accesses hinder the RDMA performance through the concurrency control mechanism. Typically, applying concurrency control incurs access amplification to guarantee the data consistency. However, there are some limited hardware-based solutions such as *Hardware Transactional Memory* (*HTM*). Furthermore, this concurrency problem is not limited to remote and local memory accesses, and there is a race among the remote memory accesses as well. For such condition, RDMA supports atomic operations such as `CAS` and `FAA`. The performance of these atomic operations intrinsically depends on its implementation in hardware [14]. Furthermore, the size of an atomic operation is limited (e.g., 64 bit). RDMA atomic operations can be configured to global (*IBV_ATOMIC_GLOB*) or local (*IBV_ATOMIC_HCA*) granularity. The local granularity guarantees that all the atomic operations to a specific address within the same HCA will be handled atomically, the global one guarantees that all the atomic operations on a specific address within the system (i.e., multiple HCAs) will be handled atomically. However, until now, only local mode is implemented in the existing HCAs.

3.3. Suitability

RDMA can perform one operation in each *Round Trip Time* (*RTT*), however traditional protocols provide more flexibility by fulfilling multiple operations. Thus, it makes RDMA more suitable for environments with single and quick operations. Furthermore, RDMA is more suitable for small and alike message sizes. Additionally, exploiting RDMA with dynamic connections cannot be beneficial due to the heavy initial cost of RDMA. Moreover, although RDMA provides high-speed communication, it limits scaling in both distance and size, meaning that the distance among nodes and the number of nodes cannot be arbitrarily large [15].

RDMA offers simple READ, WRITE and atomic operations (CAS and FAA) as well as an operation to send messages (SEND). However, it does not support sophisticated operations such as dereferencing, traversing a list, conditional read, on-the-fly operations (e.g., compression and de-compression), histogram computations, results consolidation [16]. Thus, exploiting RDMA in such environment requires precise and fine mechanisms to achieve the best performance.

4. Communications

RDMA allows several communication paradigms based on its primitives. There are two main actors on each communication, those that make requests (clients) and those that respond to the requests (servers). In this section, RDMA primitives, possible communication paradigms, and RDMA optimizations are described.

4.1. Communication Primitives

RDMA operations can be categorized in one-sided and two-sided communications. In one-sided communication (i.e., READ, WRITE, atomic FAA and CAS), the receiving side's CPU is completely passive during the transfer. However, two-sided communication (i.e., SEND) requires the action from both sides.

WRITE is a one-sided operation which has no notification on the remote host. However, sending an immediate data with WRITE notifies the remote host about the operation. The WRITE and immediate notification are an atomic unit, either both arrive or neither arrive. It should be noted that in an immediate operation, a receive request will be consumed and the immediate data will be sent in the message. This immediate data will be available in the CQE that is generated for the consumed receive request in the remote side. The immediate data are racing behind the WRITE, i.e., between the WRITE completing and the immediate data arriving.

READ is a one-sided operation that reads a contiguous block of memory (in virtual address space) from remote side and writes it to the specified local memory buffer. It should be noted that READ does not consume receive request in the remote side.

SEND carries the content of a local memory buffer to be written in a remote memory address that is specified in a pre-posted receive request in the remote side. Since both sides require to send work requests, SEND is counted as a two-sided communication. In the remote side, a work completion will be generated for the consumed receive request. Moreover, SEND supports immediate data as well as WRITE.

4.2. Communication Paradigms

Communication can be categorized as synchronous or asynchronous according to the type of message passing. In a synchronous communication, the client sends a message, and it is blocked until a response arrives, then it returns to its normal execution. In this mode, a message represents a synchronization point between the two processes. In an asynchronous communication, the client sends a message, and then continues its execution until the response is ready.

Since in an RDMA-based synchronous communication there is not a common clock between client and server to agree on a data transfer speed, busy polling or blocking functions is exploited. Meanwhile, an RDMA-based asynchronous communication can be implemented through events with notification.

Figure 3 shows possible synchronous and asynchronous RDMA communications. Firstly, the client sends its request by posting a work request to the send queue through `ibv_post_send()`. Then, it receives the completion of its work request through `ibv_get_cq_event()` or `ibv_poll_cq()`. `ibv_get_cq_event()` is a blocking function while `ibv_poll_cq()` cannot make block with any flag or timeout parameter, thus it can be blocked by busy polling. According to the type of communication, the client can either use `ibv_post_recv()` or busy polling to get its response. Generally speaking, the server receives the request and responds to it on the same manner but in the opposite order.

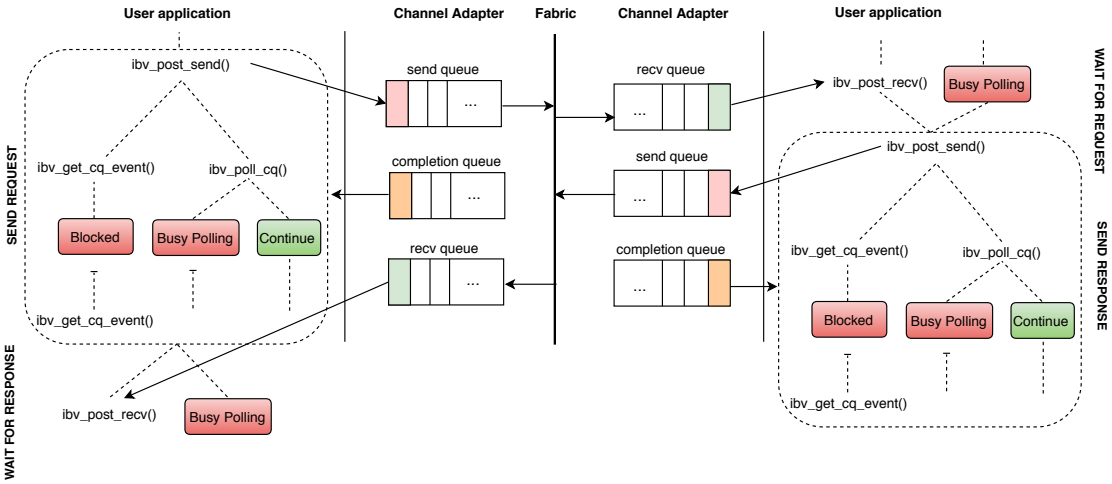

Figure 3. Synchronous and asynchronous RDMA communications.

When statements in a transaction require being executed sequentially, a synchronous communication is exploited. Synchronous communications are challenging due to the blocking essence of them. This blocking mechanism incurs a bottleneck in a system. Therefore, more attention is required in order to select a proper synchronous communication paradigm to achieve the best performance. Thus, this paper focuses on synchronous implementation of the main communication paradigms. However, the possibility of asynchronous implementation of each paradigm is discussed as well.

Figure 4 shows the main communication paradigms without middleware layer exploiting the above-mentioned communication primitives. In Model (a), the client and the server exchange a pre-defined memory address for requests and replies. The server is polling the memory address for a new request, and the client writes its request to the pre-defined memory address in the server. Then, the server replies by writing to the client memory address. In Model (b), the client writes its request in a local memory address and the server polls this memory to find a new request. Afterwards, the server replies to the client by writing to a pre-defined memory in the client. In Model (c), the server is almost passive and the client writes its request in a pre-defined memory address in the server and polls to check the response. In Model (d), the client writes its request in a local memory address, and the server polls over this memory to find a new request. Then, the client polls in a pre-defined memory in the server to check the response. In Model (e), the client sends a request message to the server, then the server returns a response message to the client. Finally, Model (f) is the traditional socket communication request and reply.

Models (a)–(c) rely on sustained-polling mechanism for synchronous communication, which can be implemented in different manners [9,17]. Typically, polling is relying on the last packet of a written message to detect request completion. However, the correctness of this polling-based approach depends on the delivery of the message in order to avoid anticipated last packet delivery or memory overwritten by an older message [18].

According to the mentioned definition, Model (a) can be fully implemented in an asynchronous manner through `WRITE` with immediate data, and synchronously by the polling mechanism. Although the client in Model (b) can be implemented asynchronously, the server cannot be asynchronous due to the lack of notification in `READ`. Model (d) does not support asynchronous communication in either sides. Models (e) and (f) can be implemented in both manners.

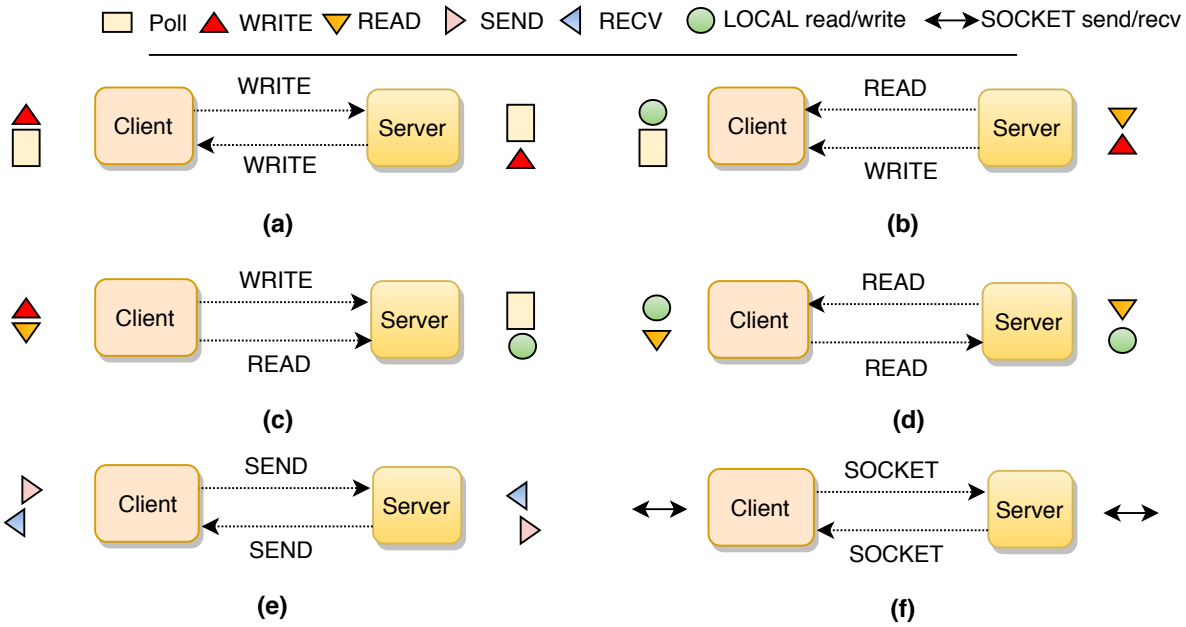

Figure 4. Communication paradigms.

As can be seen, each paradigm has its own unique characteristic which makes it suitable for a particular environment. Models (a) and (d) share the effort for communication in a fair manner. Model (c) puts the burden of communication on the client side and Model (b) on the server side. Model (a) incurs higher CPU usage in both client and server sides due to polling. Models (b)–(d) generate more network traffic for polling remote side. Models (e) and (f) do not induce a pre-defined memory address but they require synchronization in sending and receiving messages.

Each communication paradigm supports a series of connection types according to the adopted RDMA operation. RDMA supports unreliable and reliable connection types. Unlike unreliable, reliable connection guarantees the delivery and the order of the packet through an acknowledgment message from the receiver. Furthermore, RDMA supports unconnected and connected connections. Unlike unconnected, each QP in connected communication connects exactly to one QP of a remote node. This paper only considers the connected connections, i.e, *Reliable Connected* (*RC*) and *Unreliable Connected* (*UC*). Table 1 compares different communication paradigms according to client and server overheads, network traffic, communication, and connection types. Each model is named according to the operations presented in Figure 4.

Table 1. Comparing communication paradigms.

METHOD	CLIENT OVERHEAD	SERVER OVERHEAD	NETWORK TRAFFIC	COMMUNICATION	CONNECTION
WRITE-WRITE	high	high	low	a/synchronous	RC/UC
READ-WRITE	low	high	high	synchronous	RC
WRITE-READ	high	low	high	synchronous	RC
READ-READ	fair	fair	high	synchronous	RC
SEND-SEND	high	high	low	a/synchronous	RC/UC
SOCKET-SOCKET	high	high	low	a/synchronous	RC/UC

4.3. Optimization Considerations

Since RDMA is completely implemented in HCA with limited computing and memory resources, optimization techniques can significantly impact on the performance. In this section, possible optimizations are briefly described.

Data inlining: WQEs are built from different segments depending on the requested operation. One important segment is *Data* which points to the memory that contains the payload. WQEs can optionally contain data known as *inline data*. Inline data will save a memory read access for gathering

the payload. The WQE can contain single or multiple memory pointers as well as inline data, or any combination of these. Although inlining data can eliminate the overhead of memory access, it imposes limitation on the payload size. Anyhow, it is an efficient optimization for small payload sizes.

Payload size: Payload size plays a crucial role in the performance. It can increase the number of communications between CPU, memory, and HCA, consequently degrading the performance. For example, in case of 64 bytes cache line size if payload is increased from 64 to 65 bytes, two can lines must be accessed in transferring the payload to HCA. Furthermore, performance can be affected by the host bus (i.e., *Peripheral Component Interconnect Express* (*PCIe*)) parameters. For example, *Maximum Payload Size* (*MPS*) defines the maximum transfer size in each PCIe communication. If MPS is low, it can increase the number of communications between HCA and memory [19].

Transport types: RMDA supports three different connection types: *Reliable Connected* (*RC*), *Unreliable Connected* (*UC*), and *Unreliable Datagram* (*UD*). UD is an unreliable and unconnected connection which does not guarantee the delivery as well as the order of the messages. The connection types not only affect on the network communication traffic but also on the amount of consumed resources in HCA [14]. For example, UD connections consume much less resources comparing to RC and UC. The UC/UD may have higher performance comparing to RC at the expense of losing the reliability. However, zero packet loss was experienced in transferring 50 PB of data over an unreliable connection Kalia *et al.* [20]. Furthermore, it should be noted that each connection type does not support all RDMA operations. Table 2 shows the supported operations by each connection type.

Table 2. Supported operations in each connection type.

Connection Type	READ	WRITE	FAA/CAS	SEND
RC	✓	✓	✓	✓
UC	✗	✓	✗	✓
UD	✗	✗	✗	✓

Unsignaled work request: When a work request is processed, a work completion element is generated by HCA. Generating this element incurs overhead on the application to collect it. RDMA allows to send a work request without receiving the completion element. However, this approach cannot be adopted continuously due to the resource depletion. Thus, a completion work request must be requested periodically to release the taken resources. Furthermore, RDMA allows reading multiple completion elements at once.

5. Experimental Evaluation

RDMA presents asymmetric performance characteristics on either sides (i.e., client and server) [14]; however, the performance of the client side was investigated. All experiments were compiled with the gcc version 4.4.7 with 50 s warmup and 25 s measuring time. Experiments were run on a small cluster with 2 sockets AMD Opteron 6276 (Bulldozer) 2.3 GHz equipped with 20 Gbps DDR ConnectX Mellanox on PCI-E 2.0. The network topology is a direct connection with an Infiniscale-III Mellanox switch. Each machine has four NUMA nodes connecting to two sockets. Libibverbs (a part of OFED) was exploited to manage RDMA resources. Moreover, a single polling thread per machine was considered to handle requests. To compare RDMA paradigms with traditional counterparts, socket-based approach was investigated as well. Furthermore, the implementation of experiments are publicly available (https://github.com/mashemat/Communication). In each communication paradigm, throughput, latency, and scalability were investigated.

To measure the variation in the throughput of RDMA operations, four client processes reside on one host connected to one server, performing consecutive RDMA operations. This experiment was repeated 100 times and each time a client was executed for 50 s. The throughput of each client was measured, and then they were coalesced to find the total value. Figure 5 shows the results of this experiment. As can be seen, the variance between the observed throughputs is negligible, with only a few outliers.

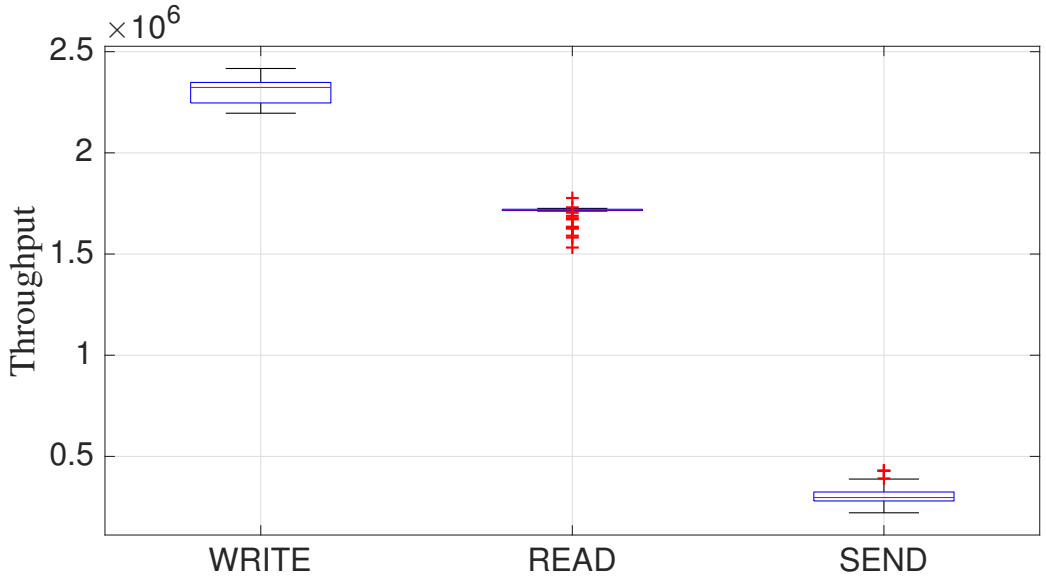

Figure 5. Variation on the RDMA operations.

Experimental Results

The optimizations described in Section 4.3 make a substantial difference in the overall performance of RDMA-based applications. Thus, primarily the experimental results of applying optimizations are evaluated. Then, the results of communication paradigms are discussed.

Figure 6 shows the impact of payload size on the throughput. In the experiment, one client performs RDMA operations with different payload sizes. As can be seen, increasing the payload size incurs the performance degradation, since the payload size affects the CPU-HCA interactions as well as the number of exchanged messages. Moreover, Figure 6 illustrates that `WRITE` delivers an higher throughput than `READ` despite both operations have identical InfiniBand path. The reason behind is that `WRITE` requires less states to be maintained both at the RDMA and at the PCIe level [8]. In a `WRITE` operation, the client does not need to wait for a response. However, a `READ` request must be maintained in the client's memory until a response arrives. Furthermore, at the PCIe level, `READ` is performed using heavier transactions comparing to `WRITE`. In addition, `SEND` presents much lower performance comparing to the `READ` and `WRITE`, since it does not bypass the remote CPU, and it requires `RECV` at the server side which is a slow operation due to its DMA interaction for writing data and CQE [8].

When an WQE completes its operation, it pushes a completion signal to the CQ via a DMA write. Pushing this signal adds extra overhead on the operation due to its PCIe interaction [8]. Figure 7 shows the throughput of RDMA operations when a selective signal is used. In this experiment, consecutive operations with 8 bytes payload size are sent unsignaled, i.e., completion signals are not generated for these operations, and then a signaled operation is sent to release the taken resources. It should be noted that the number of unsignaled operations cannot exceed the size of the queue pairs, which is set to 2048 units in this experiment. As Figure 7 illustrates, increasing the unsignaled operations increases the performance.

An WQE can inline its payload up to the maximum programmed input/output size, otherwise the payload can be fetched via a DMA read [8]. Although reliable and unreliable connection (RC/UC) types have the same header size (i.e., 36 B), they present different performances. For example, both connected transports (RC/UC) require as many queue pairs as the number of connections in HCA comparing to unconnected connections. These queue pairs can increase memory consumption in HCA and may consequently affect the performance because of the increased number of cache misses.

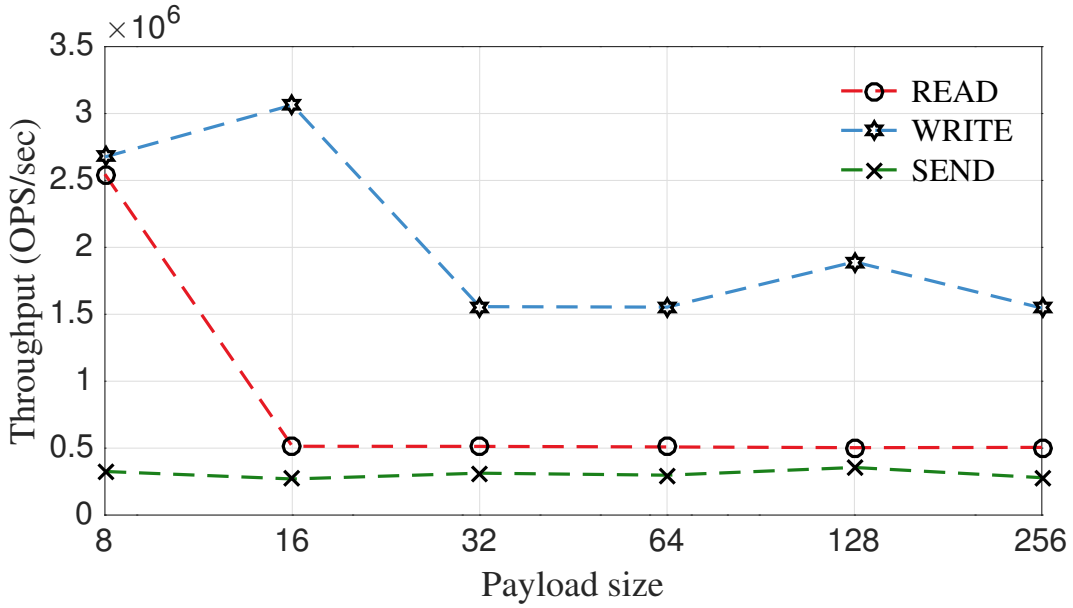

Figure 6. Payload size impact on performance.

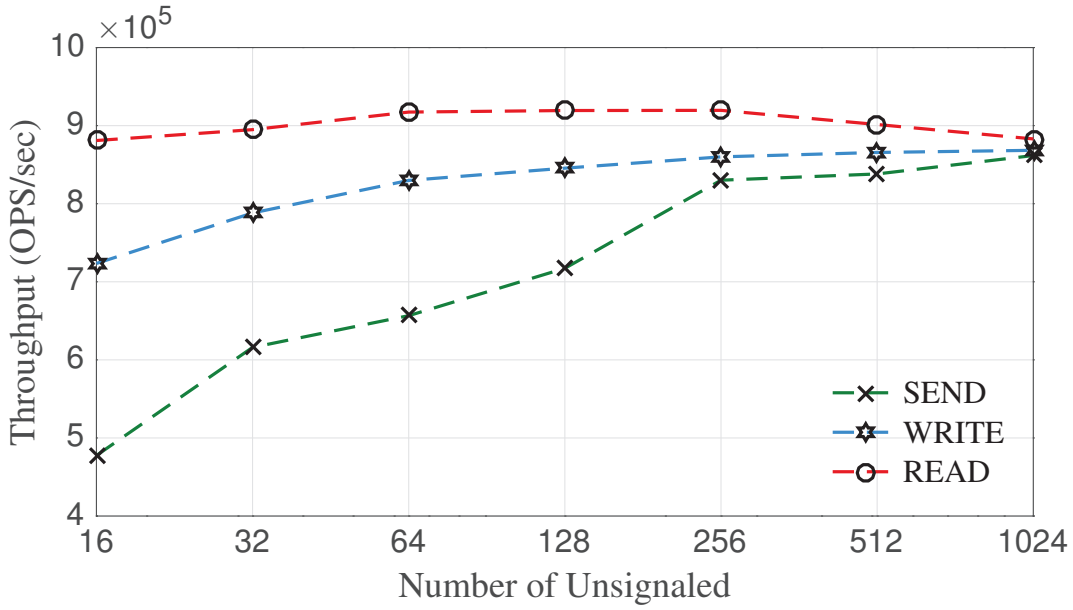

Figure 7. Unsignaled operations impact on performance.

Figures 8 and 9 show the impact of inline messages and connection types on `SEND` and `WRITE` while increasing the number of unsignaled operations. `READ` operation is not reported since it only supports reliable connections. Furthermore, it does not transfer payload, thus it does not support inline features. In this experiment, one client performs RDMA operations with 8 bytes payload size. As can be seen, the impact of inline and connection type is not the same on `SEND` and `WRITE` operations. `WRITE` is more sensible to inline and `SEND` is more sensible to connection type. These features can increase the performance up to 4.9 times in either cases.

Figures 10 and 11 illustrate the comparison of `READ` with `SEND` and `WRITE` in reliable and unreliable connections. As can be seen, `WRITE` with inline payload outperforms the other RDMA operations in both RC and UC connections.

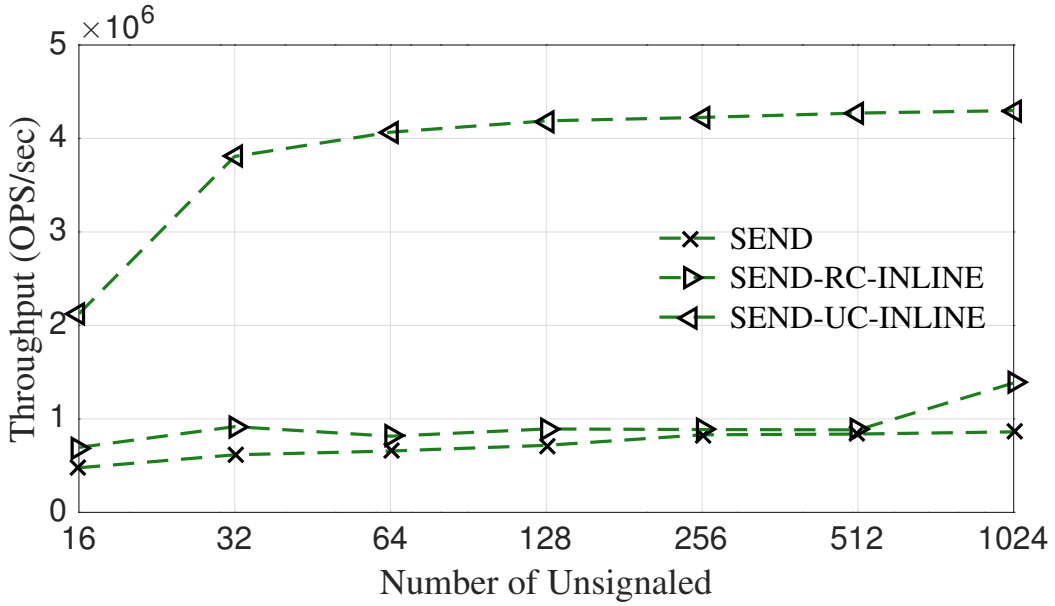

Figure 8. Inline and connection type impact on the `SEND` performance.

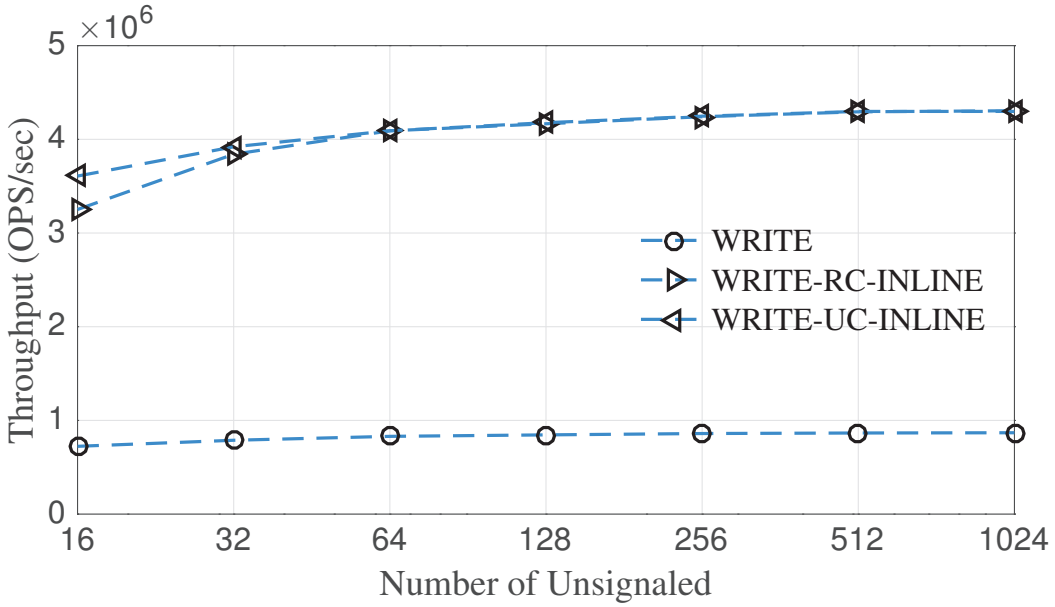

Figure 9. Inline and connection type impact on the `WRITE` performance.

Scaling is an important experiment because in real cases more than one client is typically connected to the server. Thus, an experiment was devised to investigate the impact of scaling on the performance. In this experiment, several clients send their requests towards the server. The server processes new requests in a round robin fashion. If the server finds a request from a client, it responds to the request. Each client is mapped to one dedicated core to avoid context switching. Each client sends 8 bytes message request with 512 unsignaled operations.

Figure 12 shows the performance by increasing number of clients. The degradation of `SEND` and `WRITE` throughput happens because the `SEND`s and `WRITE`s are unsignaled, i.e., client processes get no indication of operation completion. This leads the client processes overwhelming with too many outstanding operations, causing cache misses inside the HCA. As can be seen, `WRITE` and `READ` outperform the other operations. Furthermore, with few clients the `WRITE` outperforms the `READ` operation. Surprisingly, `SEND` underperforms socket communication with higher number of clients. It should be noted that socket is implemented in non-blocking mode in this experiment. Furthermore,

only one machine is adopted to deploy the client processes. However, increasing the number of client machines can alter the performance because the overhead of maintaining the QP states are distributed among different machines [14].

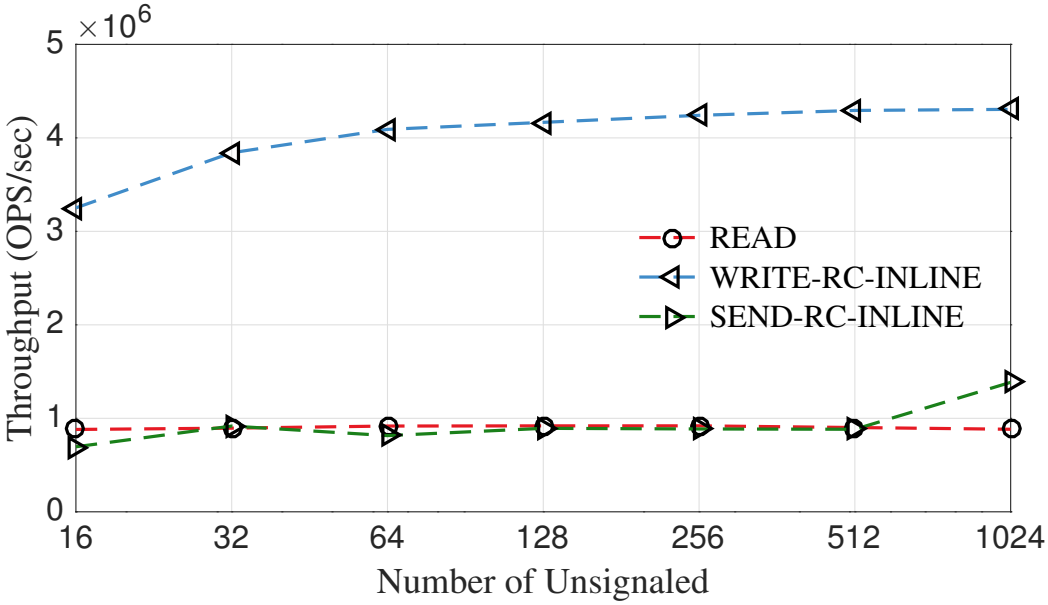

Figure 10. Performance comparison on unreliable connections.

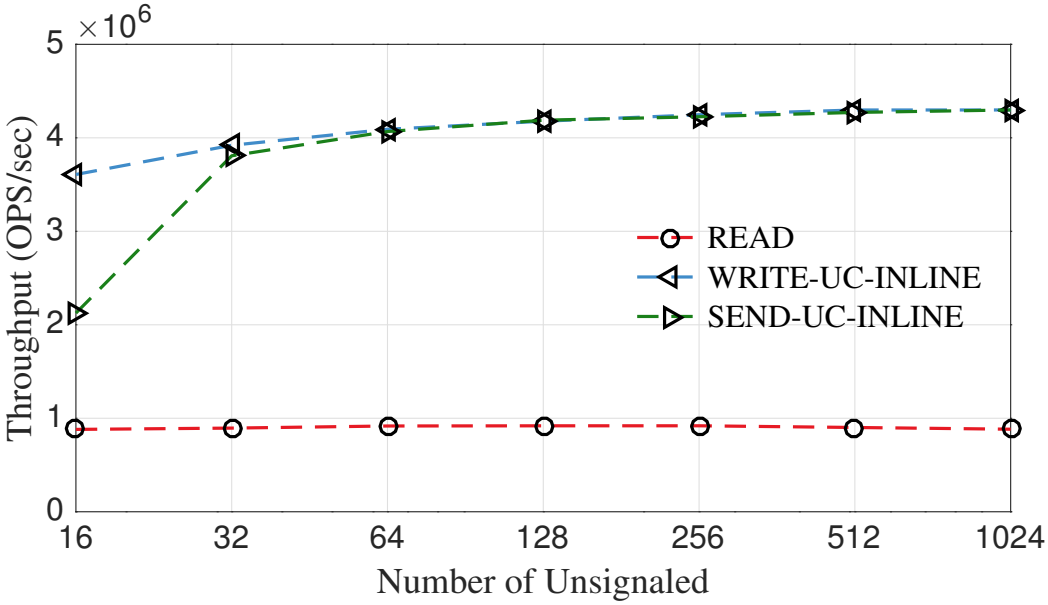

Figure 11. Performance comparison on reliable connections.

The communication paradigms are evaluated and discussed in the following. Figures 13 and 14 show the performance and the latency of communication paradigms, respectively. According to the experiments reported in Figure 12, `WRITE` operation has the best performance, thus it was expected that *WRITE–WRITE* outperforms the other communication paradigms. Surprisingly, the *WRITE–READ* has much larger (i.e., 2.4 ×) throughput than *WRITE–WRITE*. The reason is that in *WRITE–WRITE* communication, a client sends its request and polls the response area, and increasing the number of clients overwhelms the server to process requests and replies to them by `WRITE` operation. However, *WRITE–READ* paradigm lightens the server burden, since server replies to requests by writing to local memory. Thus, it can process more requests and reduce the response time of clients,

consequently achieving a better throughput. Furthermore, Figure 14 demonstrates that the latency of *WRITE–READ* paradigm is the lowest among the other paradigms. Unlike *WRITE–READ*, *READ–WRITE* overloads server by reading requests through `READ` and replies by `WRITE`. Thus, the *READ–WRITE* underperforms the *WRITE–READ*. The *READ–READ* has better throughput than the *READ–WRITE* because server only performs a `READ` operation comparing to a `READ` and a `WRITE` in *READ–WRITE* paradigm. As demonstrated in Figure 13, *SEND–SEND* and *SOCKET–SOCKET* perform poorly due to their heavy operations, which was expected based on the results in Figure 12.

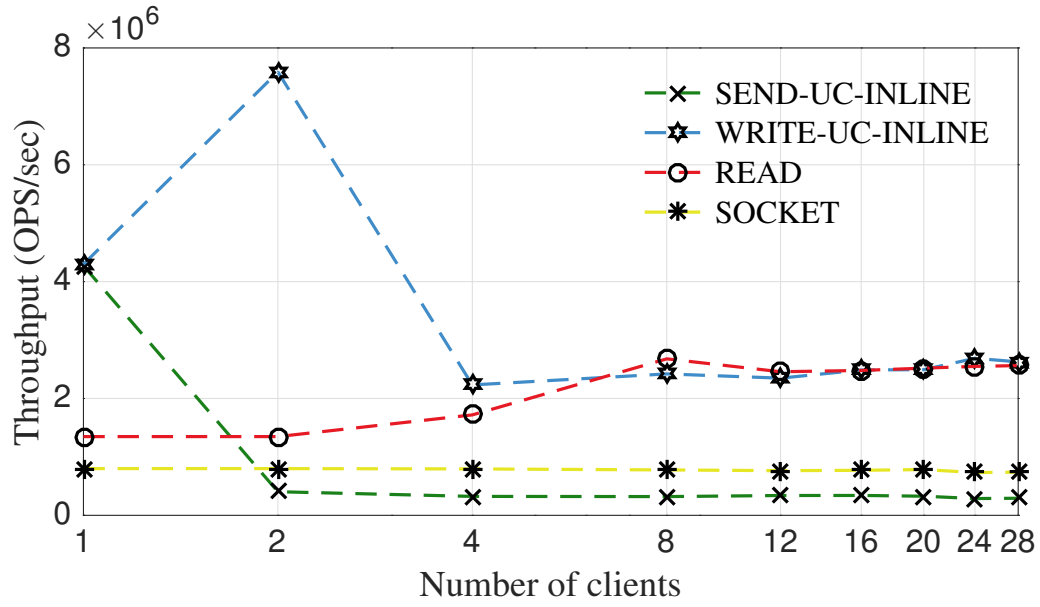

Figure 12. Scaling different operations.

Figure 13. Throughput of communication paradigms.

WRITE–READ is the only paradigm in the experiment that scales well by increasing the number of clients, however the primary drawback of this approach is that the client requires more effort in exchanging request-response. Thus, it is a good option for environments where clients are placed with other processes on the same machine because the amount of CPU that must be allocated to client

processes is significantly high. *READ–READ* is a good option to act fairly on both client and server sides. However, it performs 3.9 times slower than *WRITE–READ*. Moreover, *READ–READ* incurs huge network traffic to poll remote host.

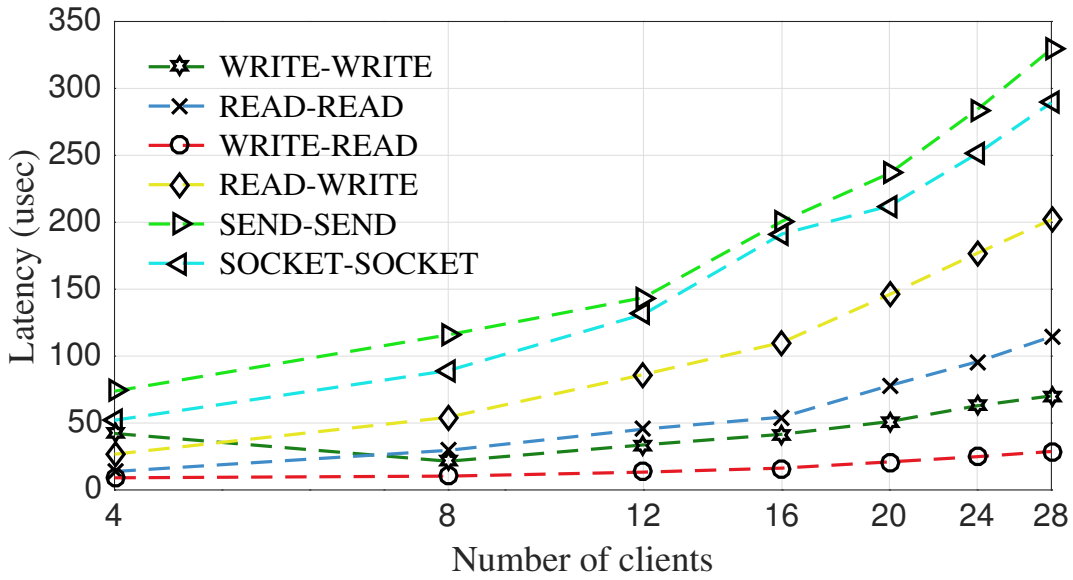

Figure 14. Latency of communication paradigms.

6. In-Memory Key-Value Store

In-memory key-value stores are vital in accelerating memory-centric and disk-centric distributed systems [21]. They are widespread in large-scale Internet services to enable managing massively distributed data. They offer a flexible data model with weaker consistency to partition data across many nodes on a computer network. In-memory key-value stores support single statement transactions consisting of either `Get` or `Put` to read or update a data. Recently, in-memory key-value stores are exploiting RDMA to reduce the communication overhead [7–9,17,22].

HydraDB is a general-purpose RDMA-based in-memory key-value store designed for low latency and high availability environment [17]. The HydraDB partitions data into different instances called *shard*. Each shard maintains a cache-friendly (i.e., cache line aligned bucket size) hash table to store the location of a key-value instead of their actual content. Figure 15 shows the schematic view of HydraDB. The scheme shows the indexing, communication protocol, and the supported operations (*Get* and *Put*).

Clients use *WRITE–WRITE* to send and receive requests/responses. In the case of *Get*, shard firstly finds the corresponding key-value address in the hash table. Then, it replies to the client the address of the key-value pairs. Thus, for the next request of the same key from the client, it exploits the `READ` based on the cached address.

Put operation fully relies on the server; the server finds the key in the hash table, then it updates the key-value out-of-place. Shard exploits out-of-place update in combination with lease-based time for the sake of data consistency and memory reclamation [23].

To explore the findings of this paper in a real-world application, the HydraDB is partially implemented and the *WRITE–WRITE* communication has been substituted with *WRITE–READ* in the *Put* transaction. *Get* is not considered because it mostly performs `READ` operation. Afterwards, some experiments are designed to compare the new method with the original one.

Query distribution is one of the main parameters in performing experiments on in-memory key-value stores. Facebook analysis reported that web requests follow Zipfian-like distribution and the majority of in-memory systems present experiments based on the Zipfian distribution with high α ratio ($\alpha = 0.99$) indicating skew curve. In this paper, for comprehensiveness, two distributions (Uniform and Zipfian) are considered that closely model the real-world traffic.

The number of keys is an important parameter in the experiment. For example, the server needs to register corresponding memory size to the number of keys which can increase the cache misses in the HCA to fetch the page table entries and influence on the performance. In this paper, 4 million keys are used with the key and value size of 16 and 32 bytes, respectively, close to real-world values.

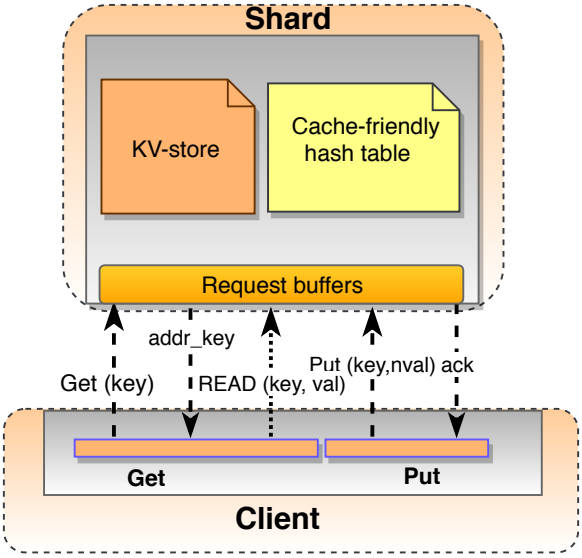

Figure 15. HydraDB schematic view. Dotted line represent the `READ` and dashed line represents the `WRITE`.

Figure 16 shows the results of the experiments. As can be seen, the modified HydraDB with *WRITE–READ* communication outperforms the original implementation by up to seven times. Moreover, *WRITE–READ* communication presents a better scalability by increasing the number of clients.

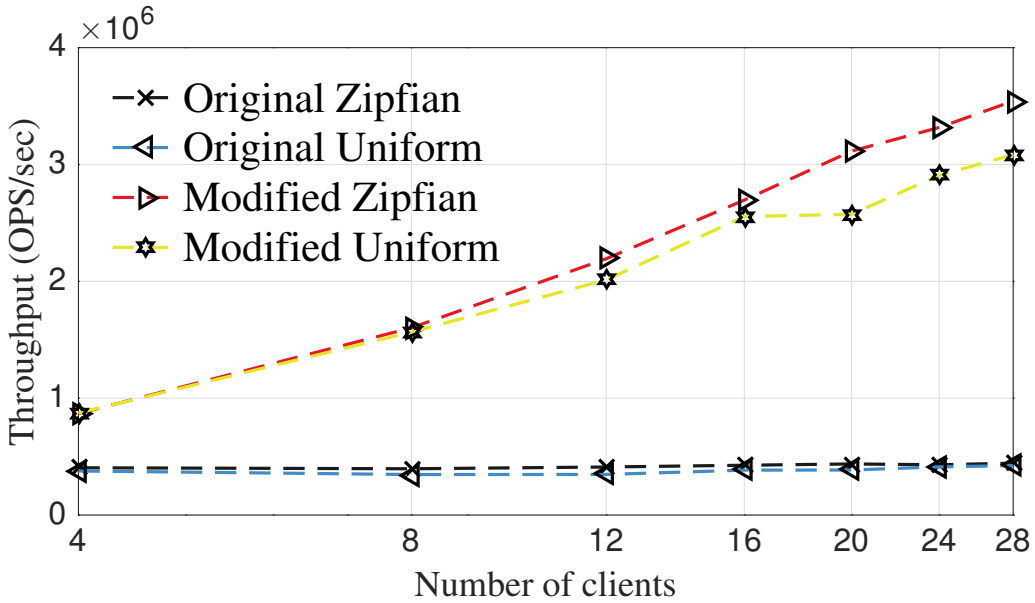

Figure 16. HydraDB *Put* transaction comparison.

7. Conclusions

This paper explores communication paradigms in designing high-performance distributed systems, focusing on RDMA-based communication in datacenter applications. The opportunities,

challenges and suitability of RDMA protocols are discussed. Several RDMA-based communication paradigms were experimentally evaluated. This paper describes a detailed analysis of how to use RDMA operations to construct RDMA-based communications. The results encourage researchers and developers to develop an improved system through RDMA optimizations. The investigation on different communication paradigms show that *WRITE–READ* outperforms the other paradigms by up to 10 times. *WRITE–READ* has been substituted with *WRITE–WRITE* in a real-world distributed in-memory key-vale store, and the performance has been enhanced by up to seven times. However, lessons from experiments suggest future distributed systems to use *WRITE–READ* communications in environment where the CPU is not a critical resource in the client machines.

As a future work, the impact of defining a window size for sending requests and replying responses will be investigated. Furthermore, applying multiple server processes and multiple polling threads per process are considered as future directions. Moreover, applying the proposed communications to well-known protocols such as Paxos [24,25] and Raft [26] are considered to be evaluated as a future direction.

Author Contributions: M.H. contributed to conceive, design, and perform the experiments as well as analyzing the data, and writing the paper. B.M. and M.R. contributed to conceptualize and review the paper.

Funding: This research received no external funding.

Acknowledgments: Authors would like to thank HPC project within the Department of Control and Computer Engineering at the Politecnico di Torino (http://www.hpc.polito.it). We would like to say thanks to Dotan Barak for providing helpful advice on the RDMA implementation on InfiniBand adapters, and Aleksandar Dragojevic for discussing fundamental issues on RDMA.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. A Remote Direct Memory Access Protocol Specification. Available online: https://tools.ietf.org/html/rfc5040 (accessed on 12 July 2018).
2. InfiniBand Trade Association. Available online: http://www.infinibandta.org (accessed on 12 July 2018).
3. Callaghan, B.; Lingutla-Raj, T.; Chiu, A.; Staubach, P.; Asad, O. NFS over RDMA. In Proceedings of the ACM SIGCOMM workshop on Network-I/O Convergence: Experience, Lessons, Implications, Karlsruhe, Germany, 25–27 August 2003; pp. 196–208.
4. Xue, J.; Miao, Y.; Chen, C.; Wu, M.; Zhang, L.; Zhou, L. RPC Considered Harmful: Fast Distributed Deep Learning on RDMA. *arXiv* **2018**, arXiv:1805.08430.
5. Ren, Y.; Wu, X.; Zhang, L.; Wang, Y.; Zhang, W.; Wang, Z.; Hack, M.; Jiang, S. iRDMA: Efficient Use of RDMA in Distributed Deep Learning Systems. In Proceedings of the High Performance Computing and Communications; International Conference on Smart City, Bangkok, Thailand, 18–20 December 2017; pp. 231–238.
6. Jia, C.; Liu, J.; Jin, X.; Lin, H.; An, H.; Han, W.; Wu, Z.; Chi, M. Improving the Performance of Distributed TensorFlow with RDMA. *Int. J. Parallel Program.* **2017**, *46*, 1–12. [CrossRef]
7. Christopher Mitchell, Y.G.; Li, J. Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store. In Proceedings of the USENIX Annual Technical Conference (USENIX ATC 13), San Jose, CA, USA, 26–28 June 2013; pp. 103–114.
8. Kalia, A.; Kaminsky, M.; Andersen, D.G. Using RDMA efficiently for key-value services. In Proceedings of the ACM Special Interest Group on Data Communication, Chicago, IL, USA, 17–22 August 2014; Volume 44, pp. 295–306.
9. Dragojević, A.; Narayanan, D.; Hodson, O.; Castro, M. FaRM: Fast remote memory. In Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation, Seattle, WA, USA, 2–4 April 2014; pp. 401–414.
10. Chen, Y.; Wei, X.; Shi, J.; Chen, R.; Chen, H. Fast and general distributed transactions using RDMA and HTM. In Proceedings of the ACM Eleventh European Conference on Computer Systems, London, UK, 18–21 April 2016; pp. 1–17.

11. Hemmatpour, M.; Montrucchio, B.; Rebaudengo, M.; Sadoghi, M. Kanzi: A distributed, in-memory key-value store. In Proceedings of the ACM Posters and Demos Session of the International Middleware Conference, Trento, Italy, 12–16 December 2016; pp. 3–4.
12. RoCE vs. iWARP Competitive Analysis. Available online: https://www.mellanox.com/pdf/whitepapers/WP_RoCE_vs_iWARP.pdf (accessed on 12 July 2018).
13. OpenFabrics Alliance. Available online: https://www.openfabrics.org/ (accessed on 12 July 2018).
14. Kaminsky, A.K.M.; Andersen, D.G. Design guidelines for high performance RDMA systems. In Proceedings of the USENIX Annual Technical Conference, Denver, CO, USA, 22–24 June 2016.
15. Romanow, A.; Bailey, S. An Overview of RDMA over IP. In Proceedings of the First International Workshop on Protocols for Fast Long-Distance Networks, CERN, Geneva, 3–4 February 2003.
16. Barthels, C.; Alonso, G.; Hoefler, T. Designing Databases for Future High-Performance Networks. *IEEE Data Eng. Bull.* **2017**, *40*, 15–26.
17. Wang, Y.; Zhang, L.; Tan, J.; Li, M.; Gao, Y.; Guerin, X.; Meng, X.; Meng, S. HydraDB: A resilient RDMA-driven key-value middleware for in-memory cluster computing. In Proceedings of the IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, TX, USA, 15–20 November 2015; pp. 1–11.
18. Mittal, R.; Shpiner, A.; Panda, A.; Zahavi, E.; Krishnamurthy, A.; Ratnasamy, S.; Shenker, S. Revisiting Network Support for RDMA. *arXiv* **2018**, arXiv:1806.08159.
19. Understanding Performance of PCI Express Systems. Available online: https://www.xilinx.com/support/documentation/white_papers/wp350.pdf (accessed on 12 October 2017).
20. Kalia, A.; Kaminsky, M.; Andersen, D.G. FaSST: Fast, Scalable and Simple Distributed Transactions with Two-Sided (RDMA) Datagram RPCs. In Proceedings of the USENIX Operating Systems Design and Implementation, Savannah, GA, USA, 2–4 November 2016; pp. 185–201.
21. Lu, X.; Shankar, D.; Panda, D.K. Scalable and Distributed Key-Value Store-based Data Management Using RDMA-Memcached. *IEEE Data Eng. Bull.* **2017**, *40*, 50–61.
22. Cassell, B.; Szepesi, T.; Wong, B.; Brecht, T.; Ma, J.; Liu, X. Nessie: A Decoupled, Client-Driven Key-Value Store Using RDMA. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 3537–3552. [CrossRef]
23. Wang, Y.; Meng, X.; Zhang, L.; Tan, J. C-Hint: An Effective and Reliable Cache Management for RDMA-Accelerated Key-Value Stores. In Proceedings of the ACM Symposium on Cloud Computing, Seattle, WA, USA, 3–5 November 2014; pp. 1–13.
24. Lamport, L. Paxos made simple. *ACM Sigact News* **2001**, *32*, 18–25.
25. Lamport, L. The part-time parliament. *ACM Trans. Comput. Syst.* **1998**, *16*, 133–169. [CrossRef]
26. Ongaro, D.; Ousterhout, J.K. In search of an understandable consensus algorithm. In Proceedings of the USENIX Annual Technical Conference, Philadelphia, PA, USA, 19–20 June 2014; pp. 305–319.