Efficient Reliability Support for Hardware Multicast-based Broadcast in GPU-enabled Streaming Applications

C.-H. Chu\(^*\), K. Hamidouche\(^*\), H. Subramoni\(^*\), A. Venkatesh\(^*\), B. Elton\(^\dagger\) and D. K. Panda\(^*\)

\(^*\)Department of Computer Science and Engineering
\(^\dagger\)Engility Corporation
The Ohio State University
{chu.368, hamidouche.2, subramoni.1, venkatesh.19, panda.2}@osu.edu bracy.elton@engilitycorp.com

Abstract—Streaming applications, which are data-intensive, have been extensively run on High-Performance Computing (HPC) systems to seek the higher performance and scalability. These applications typically utilize broadcast operations to disseminate real-time data from a single source to multiple workers, each being a multi-GPU-based computing site. State-of-the-art broadcast operations take advantage of InfiniBand (IB) hardware multicast (MCAST) and NVIDIA GPU Direct features to boost intra-node communications performance and scalability. The IB MCAST feature works only with the IB Unreliable Datagram (UD) mechanism and consequently provides unreliable communication for applications. Higher-level libraries and/or runtime environments must handle and provide reliability explicitly. However, handling reliability at that level can be a performance bottleneck for streaming applications. In this paper, we analyze the specific requirements of streaming applications and the performance bottlenecks involved in handling reliability. We show that the traditional Negative-Acknowledgement (NACK) based approach requires the broadcast sender to perform re-transmissions for lost packets, degrading streaming throughput. To alleviate this issue, we propose a novel Remote Memory Access (RMA) based scheme to provide high-performance reliability support at the MPI-level. In the proposed scheme, the receivers themselves (as opposed to the sender) retrieve lost packets through RMA operations. Furthermore, we provide an analytical model to illustrate the memory requirements of the proposed RMA-based scheme. Our experimental results show that the proposed scheme introduces nearly no overhead compared to the existing solutions. In a micro-benchmark with injected failures (to simulate unreliable network environments), the proposed scheme shows up to 45% reduction in latency compared to the existing NACK-based scheme. Moreover, with a synthetic streaming benchmark, our design also shows up to a 56% higher broadcast rate compared to the traditional NACK-based scheme on a GPU-dense Cray CS-Storm system with up to 88 NVIDIA K80 GPU cards.

I. INTRODUCTION

Towards improving performance and scalability high-performance commodity interconnect networks, such as InfiniBand (IB), and accelerators, such as Graphics Processing Units (GPUs), are increasingly prevalent in high-performance computing (HPC) systems. In the current Top500 list [4], 40.8% and 18.8% of the top 500 systems employ IB networks and accelerators, respectively. The Message Passing Interface (MPI) is the de facto parallel programming model used for HPC applications, with many highly optimized implementations being available. On upcoming exascale systems [18], new HPC applications (advanced techniques of machine learning and deep learning [5], [15]) may be comprised of over a billion processes and incorporate GPUs to process and/or produce large amounts of data in a near real-time manner. This is also true of streaming applications, which are typically data-, communications-, and compute-intensive. For instance, numerical weather prediction models require processing large amount of source data coming in streams to generate accurate weather forecasts [8]; similar requirements can be seen in time-sensitive electromagnetic or acoustical reflectivity map generation applications, such as in proton computed tomography (pCT) [10].

A typical scenario of streaming applications is illustrated in Figure 1. Streaming applications typically function in a producer-consumer pattern [13] and consist of two concurrent phases: (1) A communication phase, in which one node acts as a producer to disseminate the—typically live—source data to multiple GPU-based worker sites, i.e., consumers, through a pipeline of broadcast-type operations, and (2) a compute-intensive phase, in which multiple workers on GPU nodes perform the main computations, e.g., deep learning algorithms, and along with intra-node Peripheral Component Interconnect Express (PCIe) based communications among the host CPU and GPUs.

State-of-the-art designs for broadcast operations leverage hardware multicast features of IB (MCAST) [11] in an effort to improve the performance and scalability. Previous work [19], [7] uses an IB Scatter-Gather List (SGL) based design that combines MCAST and GPUDirect RDMA (GDR) features to boost the performance of streaming applications on heterogeneous systems. This scheme incorporates hierarchical and topology awareness to take advantage of GDR, MCAST, and CUDA inter-process communications (IPC) features. However, IB MCAST feature employed is based on the IB Unreliable Datagram (UD) transport protocol and, hence, requires re-transmission in an upper-layer (middleware/runtime) to handle and provide reliability. While the state-of-the-art Negative Acknowledgment (NACK)-based scheme [11] may be suffi-
cient for traditional HPC applications, it presents performance issues for streaming applications. Indeed, the NACK-based scheme, which is a Window-based technique, requires the source to wait for a NACK message from the receivers before it retransmits a lost packet. This approach negatively impacts streaming throughput due to introducing significant delays for lost packet retransmission. This can be an issue for real-time streaming applications, causing them to face either lagging behind or losing data to process, neither of which may be acceptable.

In this paper, we build on our prior efforts [19], [7] to propose a new design to provide reliability for the MPI_Bcast operation in a way that addresses the high performance/high throughput requirements for real-time streaming applications on modern heterogeneous IB clusters. To the best of our knowledge, this is the first work to provide the efficient reliability support for IB hardware multicast-based broadcast operations, in support of demanding streaming applications, on modern GPU-enabled HPC systems. This work makes the following key contributions:

- Analyzing the bottlenecks of the state-of-the-art solutions by extending the existing NACK-based design to handle data loss for GPU-Resident Data
- Proposing a Remote Memory Access (RMA)-based design to handle data lost without interrupting the source for streaming applications
- Providing an analytical model to illustrate the requirement of proposed RMA-based design
- Providing evaluations and analyses for the proposed designs with different failure/loss injection rates in computational experiments.

We find the proposed designs provide efficient MPI-level reliability in support of demanding streaming applications with virtually no overhead. Compared to the NACK-based approaches, experimental results exhibit up to 45% and 18% latency reductions in the Ohio State University (OSU) Micro-Benchmark (OMB) suite [6], [14] and a streaming benchmark, respectively. We have employed a streaming application benchmark towards assessing whether to invest more significant resources in further investigations with an actual streaming application. The proposed design achieves up to a 56% higher broadcast rate than does the existing NACK-based scheme in the streaming application benchmark in a modeled unreliable network environment.

The rest of the paper is organized as follows. Sections II and III provide background knowledge and the state-of-the-art related to this work. Section IV details the proposed designs. Section VI presents performance evaluation results. We highlight the related works in Section VII. Finally, we summarize the conclusions and potential future work in Section VIII.

II. BACKGROUND

New features of HPC-critical components like IB can significantly benefit software libraries and applications. This section introduces those novel features that are applicable to the proposed designs.

A. InfiniBand features

InfiniBand (IB) [1] is an interconnect standard that enables some of the most powerful HPC systems in the world. Remote Direct Memory Access (RDMA) is the major feature provided by IB to achieve high-performance communication. Four main transport modes are supported: Reliable Connection (RC), Reliable Datagram (RD), Unreliable Connection (UC), and Unreliable Datagram (UD). IB hardware guarantees successful transmission for RC and RD modes but not for UC and UD.

1) Hardware Multicast: A multicast communication is traditionally accomplished by issuing multiple point-to-point communications combined with a tree-like communication structure to achieve high performance. This is also known as software multicast. Compared to software multicast, by leveraging the IB hardware multicast feature, network traffic can be reduced significantly by avoiding the transmission of duplicate data. Specifically, the multicast sender needs to issue but one multicast communication instead of multiple point-to-point communication operations (that involve the initial sender and all receivers with many receivers acting as intermediary senders). As a result, it is possible to leverage multicast communications to significantly improve performance and scalability for large-scale systems.

B. MPI libraries for GPU-enabled streaming applications

The Message Passing Interface (MPI) is the standard parallel programming model for the applications in the HPC systems. Most of existing MPI libraries such as MVAPICH2 [14] and OpenMPI [3] take advantage of IB hardware multicast features as mentioned in Section II-A1 to enhance the performance and scalability of several collective operations on large-scale IB clusters. Moreover, they leverage NVIDIA GPUDirect RDMA (GDR) technology to improve the performance of GPU-to-GPU communications.
1) Hardware Multicast based Broadcast: IB hardware multicast has been widely applied to enhance the performance and scalability of broadcast operation in MPI [1], [11], [14]. However, the existing MPI libraries do not provide the level of performance for demanding, real-time streaming applications. In [19], [7] we take advantage of technologies such as IB Scatter-Gather-List and NVIDIA GPUDirect RDMA (GDR) to design a high-performance broadcast operation for GPU clusters. However, the existing designs either do not provide reliability support for GPU-resident data nor are optimized for streaming applications, as reflected in Table I.

III. ANALYSIS OF THE EXISTING NACK-BASED SCHEME

A. NACK-based Retransmission Scheme

Traditional multicast schemes address reliability by having the receiver inform the sender when a data loss event is detected. When such a loss is detected (and the sender is so informed), the sender performs retransmission for the lost packets. This retransmits the dropped packet to all receivers again. In existing MPI libraries, this is accomplished by sending a Negative Acknowledgment (NACK) packet to the sender process in order to trigger the retransmission process. A NACK packet is sent when a receiver does not receive an expected MCAST packet within a given time period; this is also known as a “timeout” event. Note that out of performance concerns no acknowledgment (ACK) packet is sent to confirm successful transmissions. To ensure that the sender is aware of NACK packets, the sender has to enter a “progress engine” phase to query the Host Channel Adapter (HCA). The most common method to maintain an MCAST sending window for the MCAST packets. Once the window is full, all processes in the multicast group need to be synchronized (via a barrier), to let the sender know that all the packets in the sending window have been delivered successfully.

As illustrated in Figure 2, once a receiver experiences a timeout, it then issues a NACK packet, which includes the expected sequence number of the MCAST packet, to the sender and resets the timer to wait for the expected MCAST packet. Once the sender receives the NACK packet (at the MPI-level), it re-sends the specified MCAST packet to the (entire) multicast group; this is also called a Selective NACK (SNACK)-based scheme. Another alternative is to retransmit all packets in the sending window. This can induce communication that unnecessarily consumes network bandwidth. Notwithstanding, this may be useful when a burst error occurs in extremely unreliable network environments. However, such levels of unreliability are not typical in today’s IB clusters. Thus, we focus only on the SNACK-based scheme in this paper.

B. Extension and Analysis

When leveraging hardware multicast features of IB, as described in Section II-A, application data are packetized and sent by using the Unreliable Datagram (UD) transport protocol. As a result, multicast (MCAST) packets can be lost or dropped in unreliable or crowded network environments. A reliable MPI library aims to guarantee delivery of each piece of data to the receivers. Without modifying the existing semantic of demanding streaming applications and underlying hardware configurations, we focus on designs combining IB hardware multicast and NVIDIA GPUDirect RDMA that yield a high performing, yet reliable implementation of MPI_Bcast.

To this end, we extend the Selective NACK-based (SNACK) retransmission scheme, introduced in Section III, to provide reliability support for GPU-resident data. Basically, MPI_Bcast participants (sender and receivers) maintain synchronized sending and receiving windows to track sent and received MCAST packets, respectively, for Host and/or GPU data. By taking advantage of features such as the Scatter-Gather List of IB and GPUDirect RDMA (GDR) [2] of NVIDIA GPUs, the IB adapter can write directly to GPU memory the retransmitted MCAST packets, bypassing the CPU. A drawback of this scheme is that the sender must stall to allow regular polling of the progress engine to check receipt of NACK packets. This increases overhead at the source and can potentially degrade streaming throughput. A future effort extending this work to non-blocking MPI broadcasts may allow for the application to subsume this overhead.

IV. PROPOSED RMA-BASED SCHEME

To avoid stalling the pipeline of broadcast operations and potentially limiting the throughput of demanding streaming applications due to retransmission of lost packets, we propose an MPI-3 Remote Memory Access (RMA)-based scheme. The idea is that the sender process maintains an additional window of backup buffers to reserve a copy of MCAST packets that have been broadcasted, and rely on receiver-initiated RMA operations and the associated semantics to retrieve the lost packets from that reserve copy held with the sender. When a receiver requires retrieving lost MCAST packets, it accesses this backup window directly via (point-to-point) RMA operations. For performing the RMA operations, the sender process exposes a window of memory to other processes (via MPI_WIN) in the multicast group when creating the multicast group. Thus, other processes in the multicast group can access this window directly without interrupting the sender process.
TABLE I: Feature Comparison in the Literature for Reliable MCAST-based Broadcast

<table>
<thead>
<tr>
<th>Source</th>
<th>Streaming Support</th>
<th>Utilize Hardware Multicast</th>
<th>Host Data Support</th>
<th>GPU Data Support</th>
<th>Reliability Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ref. [11]</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Ref. [19]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ref. [16]</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>❌</td>
<td>✓</td>
</tr>
<tr>
<td>Proposed design</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

As shown in Figure 3, once a receiver experiences a timeout for a given packet, it performs RMA operations, i.e., MPI_Get and MPI_Flush, to retrieve the lost MCAST packet from the sender’s backup buffers as mentioned above. While the receiver is performing the RMA operations for the lost MCAST packets, the sender can continue to broadcast data. To ensure that the receiver obtains correct MCAST packets through the RMA operations, each backup buffer has a 4-byte header to indicate the sequence number of the MCAST packet. It is worth noting that the MPI_Get operation will be mapped to an RDMA_Read operation, meaning that the HCA on the receiver performs a peer-to-peer write operation to directly write data to GPU memory under the GDR support. Conveniently, this avoids the well-known peer-to-peer read bottleneck [17], [7]. This allows the RMA-based design to efficiently retrieve large messages, with retrieving in parallel a batch of Get operations, due to the Maximum Transmission Unit hardware limit, as indicated below.

![Fig. 3: Proposed RMA-based Reliable Broadcast Scheme](image)

A. Message-based RMA Flush

Due to an IB Maximum Transmission Unit (MTU) hardware limit of 2 KB (in the IB HCA), a large message is divided into multiple small messages. In other words, multiple MCAST packets are sent when broadcasting a large message. In a highly unreliable network, bursty packet loss can occur frequently. As per MPI-RMA semantics, a synchronization/completion operation is required to guarantee the completion of the data movement operations like MPI_Get, i.e., data is transmitted. To minimize the overhead of the synchronization protocols we utilize a passive synchronization protocol. During the window creation phase, each receiver performs a Win_lock operation with shared flag. Further, since the readers (receivers) and the writer (root) do not share the same segment at a given time, the root (sender) also performs a Win_lock operation with a shared flag on the same window. During the Finalize operation, before its Win_Free operation, each process calls the Win_Unlock operation. In order to assure completion of the Get operations, an RMA flush-type operation, e.g., MPI_Win_flush, is required. However, these can be expensive. In our proposed RMA-based design, we minimize the impact by issuing but one flush operation for the entire message.

B. Helper Thread

Any reliability enhancing scheme for demanding streaming applications, including NACK-based and the proposed RMA-based schemes, require a backup buffer at the source. Maintaining a backup of broadcast data at the source requires additional memory copy operations and must be performed sequentially for each packet. To remove this extra operation from the critical path of the root (sender) of the multicast and further improve the broadcast performance and its throughput, we introduce a helper thread to perform the backup process while the main thread focuses on actual communications. As this thread is not involved in the communication process directly, the MPI runtime is not required to be initialized with threads support and, hence, avoids the known overhead of multi-threaded MPI support. We implement a simple memory flag-based synchronization and coordination between the helper thread and the root. Typically, we can always expect that data are copied before reaching remote GPU buffers because of the significant difference in bandwidth for host-to-host memory copies and PCIe communication (for copying to GPUs) [17]. To ensure correctness, the MCAST header is also copied to the backup buffer to give time for remote processes to verify the data in the (we believe, unlikely) case in which the helper thread suffers from insufficient memory bandwidth.

V. DISCUSSION AND ANALYSIS

In this section, we highlight the design choices and their expected impacts. As mentioned earlier, for streaming-throughput critical applications, the NACK-based scheme stalls the streaming pipeline for MCAST-based retransmissions. In stalling the pipeline, an MCAST-based retransmission adds more load on the network as the packet is retransmitted to all receivers in the MCAST group. Such a scheme might lead to network congestion. In contrast, the RMA-based design provides the following benefits:

- Fine granularity of retransmissions by allowing the specific receiver to initiate the retransmission of an explicit packet.
- Low overhead of retransmission due to passive synchronizations
- Higher throughput by removing the copy overhead from the critical path and offloading it to a helper-thread.

As mentioned earlier, we use a backup buffer at the source to provide reliability. To minimize the memory footprint, we designed the backup buffer in a circular fashion. The size of the buffer and the wraparound time can be determined as a function of system-specific rates, e.g., network speed of the transmission. Using this technique, we can determine the memory space (for the backup buffer) required for a specific application on a specific system to ensure reliability. Producer/consumer rates can be used to determine the lower bound limit of the backup buffer.

A. Analytical Model of Persistent Window Length

Upon transfer of a frame from the window, there is a requirement to retain buffer contents for a minimum time of $K \ast RTT$, where $K$ is a constant and $RTT$ is the round-trip time on the network, in the case where retransmission of data becomes necessary. $K \ast RTT$ represents the maximum timeout duration as part of an exponentially increasing timeout duration. As the window from which data is broadcast is sufficient to avoid repeated registration, buffer reuse through a wraparound scheme is necessary. With this, it is also necessary to guarantee that the reused buffer is currently not being read as part of a “get” operation within a $K \ast RTT$ period of time. This can be done by having the window be sufficiently long in such a way that the time to wraparound is always greater than the duration within which retransmission can occur. For a given network with bandwidth $B$-bytes/second and use of $f$-byte framed window, the time to consume a frame is at least $f/B$ (from a sender’s perspective). Hence, for a window of size $W$, the net time to wraparound is $W \ast (f/B)$. To ensure that the window wraparound does not occur during a $K \ast RTT$ period, windowing parameters $W, f, B, K$, and $RTT$ must be such that $(W \ast (f/B)) > (K \ast RTT)$. Thus, the minimum required window size is as follows:

\[ W > \frac{B \ast (K \ast RTT)}{f}. \]  

VI. PERFORMANCE EVALUATION

In this section, we present the performance evaluation and analysis of the proposed reliable schemes. Combined with our previous work [19], [7], these designs are implemented on top of MVAPICH2-GDR 2.2rc1, which is a CUDA-aware MPI library that leverages modern NVIDIA GPU Direct RDMA and InfiniBand features.

A. Experimental Environments and Metrics

Experiments were carried out on two GPU clusters: (1) An Ohio State University (OSU) Computational Science and Engineering (CSE) Department RI2 cluster, which has 16 nodes with one NVIDIA K80 GPU card per node and EDR IB HCAs, and (2) a Swiss National Supercomputing Centre (CSCS) Cray CS-Storm GPU-based cluster system, each node being equipped with eight NVIDIA K80 GPU cards and two FDR IB HCAs. We were allowed to experiment on the CSCS system with up to 88 NVIDIA K80 GPU cards spread across eleven nodes.

To simulate various unreliable network environments, we developed a probability-based packet dropping module in the MPI library. Each received MCAST packet could be forcibly dropped based on a uniform distribution. In practice, InfiniBand has implemented a hardware congestion control mechanism to achieve lossless interconnection networks [9], which typically provides a 0.01% or lower packet loss likelihood. In this paper, we simulated packet loss rates of 0.01%, 0.1%, and 1% to understand the performance of the proposed designs in realistic and in extreme network environments.

The following metrics characterize the performance of reliable broadcast operations for demanding streaming applications:

- **Overhead**: The extra time (latency) that comes from the designs for reliability support in the ideal network environment (i.e. no packet loss) compared to the existing scheme without reliability support.
- **Overall latency**: The overall latency of the broadcast operation under the above-mentioned simulated packet dropping scenarios. The lower overall latency of a broadcast design with reliability support shows the capability of providing high overall performance for streaming applications in unreliable network environments.
- **Broadcasting rate**: The amount of valid data broadcast per unit time (typically expressed as MB/s). A higher broadcasting rate helps applications disseminate incoming source data to receivers (workers) faster and achieve a higher processing rate, which is a key performance metric for streaming applications [13].

All the metrics mentioned above were measured and analyzed over five runs of experiments. Each experiment averaged performance data from 1,000 iterations of broadcast operations. We used the OSU Micro-Benchmark (OMB) suite [6], [14] to evaluate the performance of the proposed RMA-based design compared to existing NACK-based design under different simulated network environments on GPU clusters. Furthermore, we developed a synthetic streaming benchmark [7] to mimic the behavior of streaming applications. This streaming benchmark was used to understand how full the pipeline of broadcast operations can be kept for real streaming applications with unreliable networks. In our experiments, the size of the backup buffer is set accordingly as described in Section V-A. Finally, to avoid repetition of similar results between the two test systems, we provide micro-benchmark level results on the RI2 cluster and streaming benchmark results on the GPU-dense CSCS cluster.

B. Micro-Benchmark Level Evaluation

With reliability support, the overhead is introduced by the backup process as illustrated in Section IV. In the NACK-based scheme, the sender requires having a backup window of MCAST packets. Our experiments used a window size
of 256 packets. In the proposed RMA-based design, we use a helper thread to perform the backup process as explained in Section IV-B. As shown in Figure 4(a), the NACK- and (proposed) RMA-based schemes generally have nearly no overhead. However, the NACK-based scheme shows significant overhead with message sizes larger than 1 KB. This is because maintaining the backup window requires copying memory, which can become a major performance bottleneck with increasing message sizes. In Figure 4(b), it is clear that NACK-based design has about a 70% higher overall latency while the proposed RMA-based design still yields nearly no overhead for large messages. We see very low overhead in the RMA-based design since the sender process (root) is not involved in retransmissions. In this design, when the backup operation is offloaded to another thread, copying and multicast processing overlap. In this way, the overhead of maintaining the backup window is hidden. In the NACK-based scheme, there is about a 30% latency penalty since the root (the sender) has to wait for a timeout at each window interval. Also, the root process performs the backup operation and communications sequentially, absent any overlapping of these activities.

Next, we compared the overall latency of broadcast operation between NACK- and proposed RMA-based designs across a range of packet loss probability scenarios, with packet loss likelihoods ranging from 0.01% to 1%. In Figure 5(a) depicts the two reliability designs for small messages having comparable performance, with differences being less than 1 μs across all three scenarios for packet loss likelihoods. In this case, the smaller messages are split up less often than with larger messages. Thus, fewer multicast packets are needed when compared to larger messages. Indeed, we see that for smaller messages the benefits of the proposed RMA-based design is not so noticeable. Figure 5(b) exhibits the measurements observed for larger messages. In this case, we see that the proposed RMA-based design significantly outperforms the NACK-based design, with the RMA-design having about 30%, 45% and 30% less latency than the NACK-based design for 0.01%, 0.1% and 1% packet loss likelihoods, respectively. Figure 6 shows the normalized latency to RMA-based scheme. It is obvious that RMA-based scheme always yields comparable or significantly lower latency than the NACK-based design under different unreliability scenarios. The main benefit of the RMA-based design is that a multicast receiver can use RMA-based operations to retrieve a lost packet immediately while the traditional NACK-based scheme blocks all receivers from retransmission until the multicast sender is aware of NACK packet and triggers the retransmission process.

Finally, we measured broadcast rates of the two schemes under different degrees of unreliability (packet loss likelihood scenarios). We see in Figure 7 that when the message size is larger than 2 KB, which is the maximum transmission unit (MTU) (as indicated in Section IV-A) for hardware multicast in our experiments, the broadcast rate increases significantly for both schemes. As larger messages are used, more MCAST packets are injected into the network and the benefits of the RMA-based design can be readily observed. As depicted in Figure 8, the RMA-based design yields up to 96%, 298%, and 77% higher broadcast rates compared to the NACK-based scheme in the 0.01%, 0.1%, and 1% packet loss likelihood scenarios, respectively, for 4 MB messages. This is favorable for data-intensive streaming applications, which typically involve larger message sizes.

C. Streaming Benchmark Level Evaluation

Towards mimicking the behavior of real streaming applications on HPC systems, in addition to broadcast operations our synthetic streaming benchmark includes executing computational kernels on GPUs and moving data between Host and GPU memory. In the synthetic streaming benchmark, note that all GPU related application program interfaces (APIs) are non-blocking and that multiple blocking broadcast operations are issued back-to-back in order to simulate the effect of streaming applications dealing with multiple data streams. We ran computational experiments for this benchmark on the multi-GPU CSCS cluster.

Similar to the micro-benchmark level assessment of the previous section, we conducted computational experiments with the two reliability designs on a pristine network environment, i.e., one absent packet loss, and made comparisons with those to using the MVAPICH2-GDR library with reliability support disabled. Figure 9 presents latencies of the three cases. As shown in Figure 9(a), the NACK-based scheme shows no significant performance degradation compared to the case of using MVAPICH2-GDR with reliability disabled. This degradation is as discussed in the micro-benchmark assessment of the previous section. And the proposed RMA-based design exhibits nearly no overhead, as we also observed in the micro-benchmark assessment. For large messages, as seen in Figure 9(b), the NACK-based scheme, which performs backup window maintenance sequentially, has a 43% latency penalty, while the proposed RMA design exhibits nearly no overhead. This implies that the proposed RMA design can achieve high broadcast rates absent significant overhead while providing reliability when performing back-to-back broadcast operations in streaming applications.

Next, we examined the overall latency of back-to-back broadcast operations in the streaming benchmark under different degrees of unreliability (packet loss likelihood scenarios). In Figure 10(a), the proposed RMA-based scheme consistently shows 4% to 10% reduction of latency for small messages when more than one MCAST packet is lost. Figure 10(b) depicts the performance of the two schemes for large messages. The proposed RMA-based design reduces the latency of back-to-back broadcast operations up to 30%, 36%, and 24% for 0.01%, 0.1%, 1% packet loss likelihoods, respectively. In Figure 11, we see that the proposed RMA-based design performs less favorably commensurate with increased packet loss likelihood. In the proposed RMA-based design in network environments with low reliability, i.e., packets are lost on multiple receivers or the sender side, multiple MCAST receivers with lost packets use RMA operations to get
Fig. 4: Overhead latencies for OMB on the RI2 cluster

(a) Small Messages

(b) Large Messages

Fig. 5: Overall latencies for OMB on the RI2 cluster

(a) Small Messages

(b) Large Messages

Fig. 6: Comparison of overall latencies for OMB on the RI2 cluster (normalized to NACK-based scheme)
the MCAST sending window size. Therefore, the RMA-based rate of the proposed RMA-based scheme is not limited by likelihood as presented in Figure 13. Primarily, the broadcast compared to the NACK-based scheme under a 1% packet loss based scheme still has up to a 10% higher broadcast rate packet loss likelihoods. Notwithstanding, the proposed RMA-difference between the two schemes lessens for increased scenario in today’s IB clusters. As mentioned earlier, the packet loss likelihoods below 0.01%, which is a realistic 56% higher broadcast rate than the NACK-based scheme for Figure 12, the proposed RMA-based scheme yields up to a Figure 7: Broadcast rates (MB/s) for OMB on the RI2 cluster (normalized to NACK-based scheme)

Fig. 8: Comparison of broadcast rates (MB/s) for OMB on the RI2 cluster (normalized to NACK-based scheme)

the same (missing) MCAST packets. As reliability decreases, this causes increased contention. In this case, the NACK-based scheme can issue one single multicast to retransmit the lost packet for multiple receivers. In a severely unreliable IB network, the NACK-based approach may result in some receivers with lost packets receiving those lost packets faster than in the proposed RMA-based design. In practice, however, that such high unreliability in IB networks is not a common case in current IB clusters.

Lastly, we examine the broadcast rate of the proposed RMA-based scheme in the streaming benchmark case. As shown in Figure 12, the proposed RMA-based scheme yields up to a 56% higher broadcast rate than the NACK-based scheme for the packet loss likelihoods below 0.01%, which is a realistic scenario in today’s IB clusters. As mentioned earlier, the difference between the two schemes lessens for increased packet loss likelihoods. Notwithstanding, the proposed RMA-based scheme still has up to a 10% higher broadcast rate compared to the NACK-based scheme under a 1% packet loss likelihood as presented in Figure 13. Primarily, the broadcast rate of the proposed RMA-based scheme is not limited by the MCAST sending window size. Therefore, the RMA-based scheme can achieve maximum broadcast rates for streaming applications while the NACK-based scheme requires a reasonably small MCAST sending window in order to have faster response times for any required retransmissions as explained in Section III.

VII. RELATED WORK

Liu et al. [11] and Mamidala et al. [12] leverage the IB multicast feature to enhance broadcast operations for host-resident data on large-scale IB clusters. Venkatesh et al. [19] further utilize the GPUDirect technology to design a high-performance hardware multicast-based broadcast operation for the GPU-enabled IB clusters. In [7], we extend the hardware multicast-based broadcast operation to support heterogeneous multi-GPU systems for streaming applications by applying several hardware features such as IB Scatter-Gather-List and NVIDIA Inter-Process Communication (IPC).

Data-intensive streaming applications on HPC systems become popular in the recent years. Faeldon et al. [8] present a data streaming based numeric weather model to process time-sensitive sensor data to build an accurate and efficient weather forecasting system. Peng et al. [16] propose a data streaming library atop MPI to achieve a producer-consumer communication pattern for streaming applications. In [13], Markidis et al. further present the main metrics to characterize the performance of streaming applications. Although the reliability is guaranteed in their work since they use point-to-point communications, which employ the Reliable Datagram IB transport mechanism, scalability is limited on large-scale HPC systems. In this work, we propose RMA-based reliability support atop the IB hardware multicast-based broadcast operation in order to provide high scalability and high-performance communication middleware for streaming applications on HPC systems.

VIII. CONCLUDING REMARKS

As data-intensive streaming applications begin to leverage HPC resources, new technologies such as InfiniBand hardware multicast is used further to improve the performance and scalability of these applications. However, the unreliable transport protocol of InfiniBand hardware multicast leads the high-performance broadcast operations to be vulnerable in unreliable network environments. Moreover, the traditional NACK-based scheme for reliability support interrupts the pipeline of broadcast operations in streaming applications. In this paper, we present a new RMA-based design at the MPI-level to provide reliable broadcast operations without interrupting the pipeline view of the sender for the GPU-enabled streaming applications. As a result, the applications not only can fully take advantage of cutting-edge features such as hardware multicast and NVIDIA GPUDirect RDMA technologies to gain high-performance communications and processing, but also the reliable communication to avoid non-deterministic results. Our evaluation shows that the proposed RMA-based design for providing reliability has negligible overhead under various unreliable network environments. Our streaming benchmark evaluation reveals that the proposed
Fig. 9: Overhead latencies for the streaming benchmark on the CSCS cluster

Fig. 10: Overall latencies for the streaming benchmark on the CSCS cluster

Fig. 11: Comparison of overall latencies for the streaming benchmark on the CSCS cluster (normalized to NACK-based scheme)
design achieves up to a 56% higher broadcast rate compared to the existing NACK-based scheme, which implies that the proposed design is capable of facilitating high processing rates as desired in streaming applications. Based on the promising results reported herein, our future work includes exploring designs for other commonly used operations, such as reduce, gather, and scatter, and assessing the impact of our designs on real streaming applications, such as proton computed tomography (pCT) [10].

ACKNOWLEDGMENT

The authors gratefully acknowledge the support of this project under the United States Department of Defense (DOD) High Performance Computing Modernization Program (HPCMP) User Productivity Enhancement and Technology Transfer (PETTT) activity (Contract No. GS04T09DBC0017 through Engility Corporation). The opinions expressed herein are those of the authors and do not necessarily reflect the views of the DOD or the employer of the author.

We thank Drs. Sadaf Alam and Carlos Osuna for providing access to the Swiss National Supercomputing Centre (CSCS) testbed.

REFERENCES