DerechoDDS: Efficiently leveraging RDMA for fast and consistent data distribution

Lorenzo Rosa
University of Bologna
Bologna, Italy
lorenzo.rosa@unibo.it

Sagar Jha
Cornell University
Ithaca, New York, USA
srj57@cornell.edu

Ken Birman
Cornell University
Ithaca, New York, USA
ken@cs.cornell.edu

***Abstract*—Safety-critical applications have always struggled to balance strong consistency with high performance. Kernel-bypassing technologies like RDMA (Remote Direct Memory Access) promise to ease this tension by offering fast communication options that can fully leverage modern hardware. We introduce DerechoDDS, an OMG-compliant data distribution service layered over Derecho, an open-source C++ library that combines the consistency guarantees of atomic multicast with the speed of RDMA networks. In adopting this layering, we encountered a significant challenge: DDS is dominated by small messages, whereas Derecho was optimized for large ones. Hence, we identified a set of optimization techniques for small-message atomic multicast over RDMA, obtaining significant performance improvements both for the Derecho library and for DerechoDDS.**

***Index Terms*—DDS, RDMA, consistency, performance**

I. Introduction

In safety-critical settings such as automotive or avionics, applications rely on communication middleware solutions to enable fast and robust data distribution. The increasing adoption of IP-based networking even in critical settings makes it possible for applications to leverage standard interfaces such as the OMG Data Distribution Service (DDS) [1], a publish-subscribe middleware specification that guarantees predictable data delivery configurable through *QoS policies*. The QoS options permit developers to select communication properties matched to the requirements of their applications. For example, an automotive or avionics DDS that will support safety-critical system-management applications for vehicle or aircraft control might require a costly but strongly assured option, whereas a cheaper but less assured QoS level suffices for the passenger entertainment tasks. It is appealing to adopt a single API but then vary the QoS settings as needed.

The DDS communication model is based on the abstraction of a global data space, where topics are represented by objects of a given type. Each application can access the data space via an API that maps a region of local memory. Under the covers, the DDS intercepts updates, sending messages that will replicate the action on remote nodes. QoS options allow the developer to control the level of consistency for each topic: eventual consistency or weak consistency. Under the eventual consistency model, DDS guarantees that all subscribers to the topic will eventually see the same data as in the publisher's local data store. Weak consistency is used when the main goal is low latency and some message loss is acceptable.

In the current OMG standard, there is no way to obtain stronger consistency, such as the classic state machine replication guarantee that every member of a subscriber group will apply the identical updates in the identical order. This reflects a technical limitation at the time the OMG standards were debated: with the prevailing hardware of that period, and the known protocols for achieving stronger properties, the overheads were felt to be too high.

Modern hardware brings this assumption into question. Kernel-bypassing techniques such as Remote Direct Memory Access (RDMA) achieve high throughput ($>$ 200 Gbps) and low latency ($\sim$ 1-2μs), enabling direct data transfers between the virtual memory of processes on remote machines with no operating system involvement and no extra copying. Once available only on supercomputers, today this hardware is an enabling technology for next-generation autonomous systems. Our work begins with the insight that these developments make it feasible to offer stronger DDS QoS options.

This paper presents *DerechoDDS*, a DDS implementation that leverages RDMA to offer high consistency at high speed. DerechoDDS layers the standard DDS API over a mature, RDMA-capable, open-source library, Derecho [2], which offers point-to-point and multicast communication options, supporting failure atomicity, total ordering, and optional message logging with durability [2]. The Derecho protocols have been proved correct using standard techniques and the implementation checked using the Ivy protocol verifier. Derecho can run on fast datacenter TCP but is optimized for RDMA. It sends small messages with one-sided RDMA writes, and large ones in a binomial tree pattern using two-sided RDMA transfers [3].

Our initial finding was that although highly efficient when exchanging large messages, Derecho was less efficient for small ones. This posed a problem, because most DDS updates are a few KB or less. We microbenchmarked our solution to understand the performance-limiting code paths, then introduced a set of optimization techniques that yielded substantial speedups. Interestingly, the techniques required were quite general, and hence should also be of value in other coordination-based middleware components. They include a new model for coupling application threads to an underlying platform thread, an innovative and highly general form of opportunistic batching, and the use of zero-sized message sends to avoid pauses.

II. Derecho Background

We start with a quick introduction to Derecho. The system was created as a C++ library that can be used to organize a group of processes (*nodes*) as a distributed service that can execute either on physical servers or in a virtual environment. Derecho manages the group's membership, which evolves through a series of views using partition-free state machine replication. Derecho applications are constructed around "replicated objects," each of which maps to a *subgroup* consisting of some subset of the top-level group membership. An operation that mutates the state of such an object occurs as a multicast carrying the parameters to the operation, and then will be performed by all replicas.

To support this, Derecho implements a version of Paxos optimized for RDMA, but modified to use the traditional quorum interaction only for membership updates. Data delivery is via atomic multicast, and offers a critical path free of locks and that avoids copying. All messages are delivered to the application by every member in the same order with consistency maintained across failures. Message delivery employs a round-robin ordering, one message per sender per round, ordered by the sender ordering in the subgroup membership list. Members deliver a message only when every other subgroup member has confirmed its reception. Cleanup after a failure is automatic and transparent: either the operation is completed in the current view, or any pending messages are deleted, and the entire multicast is resent in the next membership view.

Derecho transports data over a data plane, which is implemented differently for large and small messages. In the case of the small messages that arise in Derecho DDS, the key library components are a control layer, modeled as a shared state table (SST), a data layer protocol (Small-Message Multicast, SMC), and a polling framework that orchestrates the two.

Derecho's protocols look quite different from a classic Paxos protocol. A standard Paxos protocol is structured as a 2-phase exchange of information in which a leader proposes a message (or batch of messages) and a delivery "slot", participants respond, and then the leader commits after a quorum is achieved. In Derecho, the same *information* is shared, but the communication pattern is completely different. Rather than have leaders that make decisions, all nodes exchange status information through the SST using a direct remote memory update to report new events. Every node sees the evolving state of all its peers, and independently deduces protocol progress.

Each node runs a predicate thread that polls a series of predicates, using its local copy of the SST as input, invoking the corresponding *triggered code* if a predicate is satisfied. For example, if node p has received reports from every node that all messages have been received through message k, p can independently deduce that it is safe to deliver its local copies of the messages up to k in the predetermined round-robin order. In this example we see how *monotonicity* is useful in Derecho: SST data is expressed using counters that only increase, booleans that flip from false to true but never back, etc. The consequence is that predicates often can discover a property (such as deliverability) in a single action that may cover multiple messages.

The SMC protocol is the workhorse for DerechoDDS. It manages a set of ring-buffers within the SST: one per sender. To send a message, an application must wait until a slot is free, fill it with relevant data, and then increment a "messages ready" counter. SMC will then issue the corresponding RDMA write to first push the data to the remote subgroup members, and then the counter. Ideally, the size of the buffer (a configuration parameter) should be large enough to avoid senders running out of free slots, thus enabling continuous sending.

Although all of this logic is expressed using predicates, three are of special interest when exchanging small messages. A *send predicate* detects whether the application has prepared new messages that are ready to be sent. A *receive predicate* monitors the SMC slots to discover new messages, and the corresponding trigger acknowledges the reception of these messages to the other members. Finally, a *delivery* predicate checks the SST for messages whose reception has been acknowledged by all the members, and thus are ready to be delivered to the application. The performance of these predicates is crucial for the performance of the applications, hence we targeted them in our optimization work.

III. Small Message Optimizations

As a first step, we evaluated the performance of Derecho with sending patterns and message sizes typical of DDS: messages of up to a few KBs, a few dozen topics with overlapping subgroups. We noticed that the system was unexpectedly slow, and identified three main causes for this poor performance and three optimization techniques to address them. Those solutions turned out to have general applicability in other coordination-based distributed middlewares running on high-speed networks. Due to page-limit constraints, we limit ourselves to a summary.

Opportunistic batching. A key source of inefficiency in SMC was that the latency of sending control data (e.g., acks for receiving a message) is comparable to the latency of sending the application messages themselves. Figure 1 shows that the RDMA write latency for small messages increases only marginally with the data size, rising from 1.73 μs for 1-byte data to 2.46 μs for 4KB data. This behavior particularly affects the Derecho predicate thread described in Sec. II, which generates an ack for every new receive and delivery event. Worse, posting each RDMA request to the NIC takes $\sim 1 \mu s$

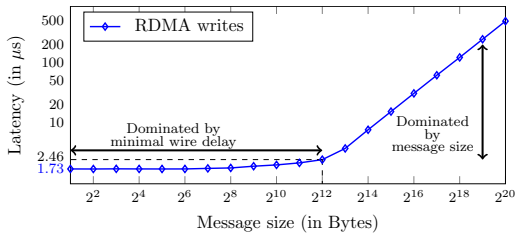
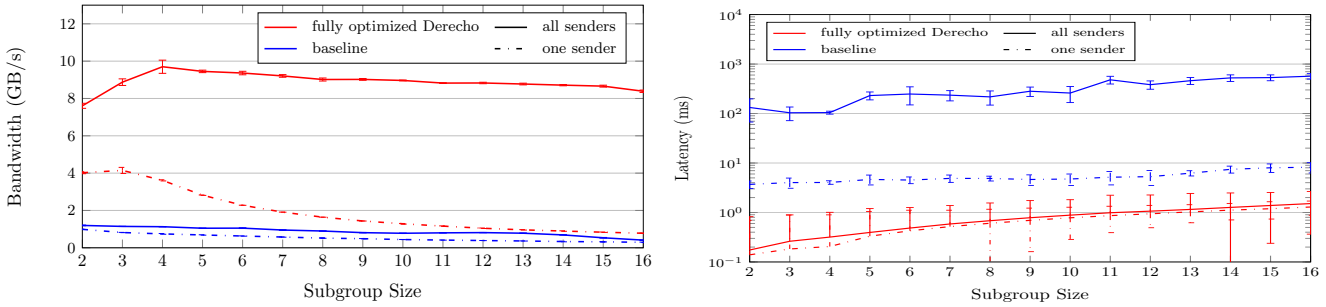


Figure 1: RDMA latency vs data size. Latency remains almost constant for up to 4KB message size.



(a) Average bandwidth utilization for different group sizes.

(b) Average latency for different group sizes

Figure 2: Performance of the atomic multicast protocol with one subgroup and 10KB message size, before and after our proposed optimizations.

in our setting, a significant delay in light of the critical role of the predicate thread for Derecho. We address this issue by batching events at different stages of the communication pipeline: send, receive and delivery. Instead of letting the system wait to accumulate a fixed-sized batch of messages, which would disrupt performance, we adopt a form of self-balancing *opportunistic batching*: a batch can be smaller or larger depending on the number of events a predicate discovers as it loops. For example, the send predicate checks to see if multiple messages are ready. If so, it aggregates them on the fly, and sends a batch. Interestingly, the opportunistic batching of messages or acknowledgments leads to an improvement of both throughput and latency. In contrast, traditional batching mechanisms wait to collect each batch, hurting latency.

Null-send scheme. Recall that Derecho employs a fixed, ordered membership for each epoch, delivering messages in round-robin order by sender. However, this implies that if a sender is not ready to send its next message, the delivery of messages from other senders could be delayed. The problem is that application sending rates might not be steady, and even if they are, delays can be introduced by the OS (e.g., scheduling or interrupt-servicing). To address this issue, we introduce a *null-send scheme*: when a sender node detects that it is due to send a message but has none ready, it sends a dummy zero-sized message (called *null*), permitting continued delivery of messages from other senders. In typical DDS use cases, this scheme introduces a negligible bandwidth overhead while making the system adapt very well to real-time delays.

Efficient thread synchronization. Applications access shared data structures when accessing messages or preparing new ones to send. Here, locking protects against concurrency conflicts but can delay the predicate thread. Additionally, many predicates were interleaving accesses to that state (SST) and RDMA write operations. As the latter are costly (they can consume 20-50% of the total predicate time), we refactored the predicate code and placed RDMA operations only at the end. Since the logic of a predicate does not depend on the state at a remote node, but only on what is present in the local SST, it is safe to release locks before proceeding with the time-consuming communication operations. Moreover, this optimization increases batch sizes.

We assess the performance impact of those three optimizations by comparing the optimized system against a baseline version of Derecho protocols. Our performance tests employ an application replicated on an increasing number of nodes (subgroup size from 2 to 16). In one test every member is a sender, whereas there is a single sender in the other test case. Each sender node sends a total of 1 million messages using the new, strongly consistent, QoS option. All members receive every message, and deliver them in the same order. In this graph we fix the message size at 10KB, but we also evaluated smaller sizes and obtained very similar results. The tests were executed on our local cluster equipped with 16 physical machines connected with a 12.5GB/s (100Gbps) RDMA Infiniband switch. Each machine has 16 physical cores and 100GB of RAM. Figure 2 plots the results. Overall, we see that Derecho's bandwidth utilization increased from 1GB/s to 9.7GB/s in the "all senders" case, and network utilization improves from 10% to 77.6%. Even with just one sender, where performance declines with the subgroup size due to increased coordination overheads, our optimizations significantly improved both bandwidth and latency.

IV. DerechoDDS

DerechoDDS is a prototype implementation of the OMG DDS API, called Data-Centric Publish-Subscribe (DCPS) [1], which uses Derecho as the underlying communication support. To the best of our knowledge, this is the first attempt to layer the DCPS API on a kernel-bypassing technology like RDMA. Beyond the standard QoS options, DerechoDDS introduces new consistency-related QoS policies to meet the requirements of mission-critical uses, sustaining very high speeds even with the most stringent QoS policy.

Applications use DerechoDDS through the standard DCPS interface. DerechoDDS layers this interface over Derecho by forming a single "top-level" group that includes all publishers and subscribers. The user then defines data types and topics, modeled as subgroups that include only those processes that will publish or subscribe to them. Although our prototype also supports "external clients", nodes that are not in the group but connect through an extra relaying step, space constraints precluded evaluation of that configuration.

DerechoDDS supports a zero-copy data path by allowing developers to build messages "in place," i.e., directly in the

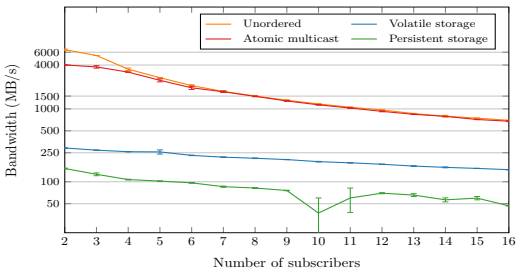


Figure 3: DDS bandwidth for its 4 QoS levels.

memory area that Derecho will write via RDMA. This is crucial: even a single data-copying step would sharply reduce performance on high-speed networks. The system offers four consistency QoS policies. 1. **Unordered**: This level corresponds to the *eventual consistency* of the standard DDS. Data is delivered without waiting for stability, and will not be retained after delivery. 2. **Atomic multicast**: This is layered directly on the Derecho's atomic multicast. Again, data is discarded after delivery. 3. **Volatile storage**: In addition to the atomic multicast guarantees, the most recent data is retained in receiver memory, allowing late subscribers to catch up. 4. **Persistent storage**: Data is additionally appended to a log file on persistent storage for debugging, additional fault tolerance or time-series data analysis.

We evaluate the performance of DerechoDDS for a single topic with one publisher and an increasing number of subscribers. We define a *Sequence* data type representing a byte sequence to be exchanged among the entities. The publisher publishes 1 million samples of the same type, each of 10KB size. Publishers and subscribers are all on different nodes to stress the network performance. Figure 3 shows the bandwidth of DerechoDDS for the offered levels of consistency, running on the optimized version of the Derecho library. Interestingly, we observe that for unordered and atomic multicast mode the performance is very similar, and corresponds to the Derecho atomic multicast performance showed in Figure 2a for the single sender case. This means that the atomic multicast consistency level does not add a significant overhead over the eventually consistent mode. Moreover, that also demonstrates that our DDS implementation is extremely efficient in preserving Derecho's high performance. Speed of the volatile case is limited by memory copying, while the persistent case is limited by DMA data rates to the storage device.

V. Related Work

Traditionally, each application domain defines its own communication middleware: e.g., SOME/IP [4] is prominent in the automotive community. However, usually these solutions lack the scalability and the extensive QoS support of DDS, which is domain-agnostic and yet integrated with domain-specific platforms, such as AUTOSAR for automotive.

All the major DDS implementations, either commercial or open-source, provide a shared-memory, zero-copy data path to applications co-located on the same host. However, they cannot offer these features for inter-host communication since they use the TCP/IP networking stack. Instead, DerechoDDS leverages RDMA to share memory among remote applications, and relies on the Derecho atomic multicast protocol to offer much stronger consistency guarantees.

The formal model employed by DerechoDDS is discussed in [5] and its protocols are proved correct in [2]. Work on machine-checked proofs (using the Ivy protocol checker and a Coq-based prover) is underway.

A significant part of our work was dedicated to optimizing Derecho for small messages. Prior work on improving performance on RDMA networks was highly influential for systems dominated by one-to-one interactions. In particular, the technique of *opportunistic batching* was also explored in Kalia et al. [6]. However, whereas Kalia focused on one-to-one interactions in a key-value storage, DerechoDDS maps to atomic multicast, requiring a more complex approach that optimizes all stages of the communication pipeline.

VI. Conclusion

We presented DerechoDDS, an OMG-compliant automotive and avionics data distribution service that leverages modern RDMA hardware to offer a zero-copy data path for publish-subscribe communication among remote hosts. DerechoDDS offers strong consistency guarantees for message delivery order and data durability, as required by safety-critical applications. We obtained these properties by layering the standard OMG DDS interface on top of Derecho, a mature library for state machine replication over RDMA that supports a range of QoS options and employs provably correct Paxos-based protocols. At the outset Derecho was poorly optimized for the DDS traffic pattern, but we successfully improved performance using a set of techniques that should also be broadly useful in other systems that exchange small messages via kernel-bypassing techniques. Finally, we evaluated the performance of a simple DDS application to demonstrate its effectiveness. In the future, we plan to compare DerechoDDS with other DDS implementations, looking at both performance and suitability for mission-critical tasks.

Acknowledgments The authors wish to acknowledge the support received from AFRL/RY (Wright-Patterson), Microsoft, Siemens and Nvidia corporations. We additionally wish to express our gratitude to the anonymous reviewers for their constructive comments and suggestions.

References

[1] "OMG Data Distribution Standard." [Online]. Available: https://www.dds-foundation.org/omg-dds-standard

[2] S. Jha *et al.*, "Derecho: Fast state machine replication for cloud services," *ACM Trans. Comput. Syst.*, vol. 36, no. 2, Apr. 2019.

[3] "Rdma aware networks programming user manual." [Online]. Available: https://www.mellanox.com

[4] AUTOSAR, "SOME/IP Protocol Specification." [Online]. Available: https://www.autosar.org/standards/foundation

[5] K.Birman *et al.*, "Communication support for reliable distributed computing," in *Proceedings of the Asilomar Workshop on Fault-Tolerant Distributed Computing*. London, UK: Springer-Verlag, 1990, pp. 124–137.

[6] A. Kalia *et al.*, "Using RDMA efficiently for key-value services," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: ACM, 2014, pp. 295–306.