

Performance Evaluation of Distributed Computing over Heterogeneous Networks

Ouissem Ben Fredj and Éric Renault

GET / INT — CNRS UMR 5157 SAMOVAR
9, rue Charles Fourier, 91011 Évry, France
Tel.: +33 1 60 76 45 73, Fax: +33 1 60 76 47 80
{ouissem.benfredj,eric.renault}@int-edu.eu

Abstract. RWAPI is a low-level communication interface designed for clusters of PCs. It has been developed to provide performance to higher applications on a wide variety of architectures. We implemented RWAPI on top of the modular software architecture called GRWA. RWAPI supports Ethernet, InfiniBand and Myrinet network interconnects. This paper introduces RWAPI and the design of its network component on top of both InfiniBand and Myrinet interconnects. We obtained a very low latency and high throughput compared to MPI results.

1 Introduction

High-speed network interconnects that offer low latency and high bandwidth have been one of the main reasons attributed to the success of commodity cluster systems. Some of the leading high-speed networking interconnects include Gigabit-Ethernet, InfiniBand [1], Myrinet [2] and Quadrics [3]. Two common features shared by these interconnects are User-level networking and Direct Memory Access (DMA). The best suited communication protocols that use efficiently these new features are one-sided protocols. It means that the completion of a send (resp. receive) operation does not require the intervention of the receiver (resp. sender) process to take a complementary action. RDMA should be used to copy data to (from) the remote user space directly. Suppose that the receiver process has allocated a buffer to hold incoming data and the sender has allocated a send buffer. Prior to the data transfer, the receiver must have sent its buffer address to the sender. Once the sender owns the destination address, it initiates a direct-deposit data sending. This task does not interfere with the receiver process. On the receiver side, it keeps on doing computation tasks, testing if new messages have arrived, or blocking until an incoming message event arises.

At the network layer, many manufacturers have built RDMA features that ease the implementation of one-sided paradigms. For example, the HSL [4] network uses the PCI-Direct Deposit Component (PCI-DDC) [5] to offer a message-passing multiprocessor architecture based on a one-sided protocol. InfiniBand [1] and Quadrics [3] proposes native one-sided communications. Myrinet [2,6] and QNIX [7] do not provide native one-sided communications. But these features may be added (as for example in GM [8] with Myrinet since Myrinet NICs are programmable).

In the past, remote-write has been implemented in generic message-passing libraries like MPI-2 [9] or dedicated message-passing libraries like the PUT interface [10,11]

for MPC-OS, PAPI [12]. However, these implementations were dedicated to a given interconnect and no effort has been made to provide a generic or minimalist API to the remote-write. Gas-Net [13] and MP_Lite [14] provide one-sided communication primitives but without any associated programming model. In this context, we defined RWAPI [15] (the Remote-Write API) as a generic and minimalist API for the remote-write and GRWA (the Global Remote-Write Architecture) which aims at providing a set of independent modules to implement RWAPI on top of any interconnects, any CPU models... This articles describes our implementations of RWAPI for both Myrinet and InfiniBand interconnects and compares performance with other available message-passing libraries (especially MPI).

The document is organized as follows: first we introduce RWAPI [16]; section 3 develops the implementation of RWAPI for Myrinet and InfiniBand; section 4 describes the performance benchmark; the last section before the conclusion provides some performance measurements.

2 RWAPI

A previous study [15] of both hardware and software requirements for high-speed network protocols has led to design the Remote Write protocol. Remote Write is a one-sided communication protocol based on the remote write primitive. It requires the sender of a message to provide all the information needed to copy a contiguous memory area from one node to another node.

RWAPI (which stands for Remote-Write Application Programming Interface) is a lightweight interface designed to provide a single remote-write primitive. The goal we are trying to achieve is to provide the smallest set of functions that enables to write any parallel programs. This way, we expect to achieve the best performance for communications while requiring as less development as possible to port our interface to new architectures.

There are two kinds of messages in RWAPI. The first message type requires the destination node identification, both local and remote addresses and the size of the message. Messages in this case can be of any length. The second message type just requires the destination node identification and the message content; the size is limited to 16 bytes. They may be helpfully used to transfer small amounts of information of any kind from one node to another. However, even if they are not limited to this specific use, they are especially useful to exchange addresses before the other message type transfers can occur.

The API is as follows:

- `int rwapi_init (int, char **)` must be called before any other RWAPI functions in order to set up the communication interface.
- `int rwapi_finalize ()` should be called after all RWAPI functions and before exiting the program. This function ensures that all FIFOs are flushed before leaving.
- `int rwapi_rank ()` returns the rank of the local node in the virtual parallel machine.
- `int rwapi_size ()` returns the number of nodes in the virtual machine.
- `void * rwapi_alloc (size, net *)` allocates a contiguous memory block of the given size. If the underlying network interface requires the use of contiguous physical

memory, it is attached to the application transparently. The value returned by this function is the virtual address in the virtual address space of the process where the contiguous memory block has been attached. The second parameter is the address where the “network” address will be stored when returning from the function. This address is the one that must be used for sending data.

- `int rwapi_free (void *)` deallocates the memory area provided as a parameter.
- `int rwapi_send (node, small)` sends a small message to another node. The value returned by this function is an error code.
- `int rwapi_receive (node *, small *)` returns information about the oldest incoming message that has not been taken into account yet. The value returned by the function is 0 if there is no message pending and 1 if a message has been taken into account. In this case, the node and the small message are stored at the addresses provided as parameters.
- `sid rwapi_write (node, net, net, size)` sends an arbitrary-long message to another. Both local and remote “network” addresses must be provided together with the size of the message. The Send ID (SID) returned by the function can be latter used in order to determine if the message as been sent or not (this is useful to reuse a memory area).
- `int rwapi_issent (sid)` checks wether the message identified by the SID has been sent or not.

Note that, `rwapi_write`, `rwapi_send`, `rwapi_issent` and `rwapi_receive` are non blocking functions.

3 Implementation Details

In order to launch application’s processes according to the SPMD model, we use an SSH-based spawner which creates a *master process* on the current host and one process on each host provided as an argument. Then, each process can communicate with the *master* to get information such as the process rank, the number of processes, the application’s arguments and other control informations. Then, processes perform collective operations on top of socket-based connections to initialize and finalize the application transparently to the user.

To maintain the non-blocking semantic of RWAPI operations (`rwapi_send`, `rwapi_write`, `rwapi_receive`, and `rwapi_issent`), we used a host memory receive list (HMRL). Thus, when a receive message event arises while the process is not waiting for messages, the interface copies the content of the message event in the HMRL. Note that message events do not contain user data except those corresponding to `rwapi_send` operations. In this case, the length of the user data is limited to 128 bits and thus a copy of this message is not expensive.

RWAPI over Myrinet Interconnect: There are many ways to implement the Remote-Write protocol on top of Myrinet. One solution would be a native implementation which would consist in developing a new MCP, a new kernel driver and a new user library. This solution is under development and good performance are expected. However, this would not be portable since an MCP should be provided for every network card version. To

overcome this limitation, we developed the Remote-Write protocol on top of GM. This way, RWAPI is automatically available for all architectures and NIC versions that GM supports.

In order to associate a process rank to its GM ID, processes perform an exchange of GM IDs using the *exchange* operation set up by the spawner's *master process*.

The `rwapi_send` function insures that GM is ready to send messages before calling the `gm_send_to_peer_with_callback` function. The later function is the fastest send function provided by GM since it uses the same port number as the sender at the receiver side. We set up GM to copy the data in the send descriptor to avoid an expensive DMA operation performed by the MCP to copy data from the host memory to the NIC memory. We also use Write-Combining feature instead of PIO or DMA to copy the send descriptor from the host memory to the NIC memory. This feature combines many PIO operations and is adapted to medium message size.

Similar to the `rwapi_send` function, the `rwapi_write` function insures that GM is ready to send messages before calling the `gm_put` function. The `gm_put` function is suited to the remote-write paradigm since it guarantees the minimum number of copies while transferring the data. Indeed, it copies the data from the host memory directly to the NIC memory without any system calls.

RWAPI over InfiniBand Interconnect: RWAPI uses the SSH-based spawner to launch processes in both InfiniBand-based and a Myrinet-based clusters. The first step of the initialization is the creation of a bidirectional channel between each pair of processes. The InfiniBand mechanism that allows the creation of such a channel is the QP (Queue Pair). Each QP is configured for a particular type of service independent from the other. These service types provide different levels of service and different error recovery characteristics. The available transport service types include: Reliable Connection (RC), Unreliable Connection (UD), Reliable Datagram (RD), Unreliable Datagram (UD) and Raw. The transport type used is RC which provides the highest level of reliability and predictability. RC requires for each process an explicit connection with all other processes. Thus, $(N - 1)$ QPs (N being the number of processes in the parallel application) are created by each process.

The second step of the initialization sets up the size of the different communication queues for each QP, including the send queue (SQ), the receive queue (RQ), the send completion queue (SCQ), and the receive completion queue (RCQ). This step finishes by creating a local process ID (LID) and $(N - 1)$ QP IDs. These IDs are broadcasted to the other processes in order to build the channels between local and remote QP.

4 Performance Benchmark and Testbeds

We compared our implementation with the version of MPI, the other existing library available for both Myrinet-2000 Technology (developed on top of GM) and InfiniBand interconnect (developed on top of VAPI [17]). For both MPI and RWAPI we developed our own benchmarks. We use three separate benchmarks (see figure 1). The first benchmark (figure 1(a)) is the classic ping-pong in which a message can only be sent once the previous one has been received. The second one (figure 1(b)) is a bidirectional

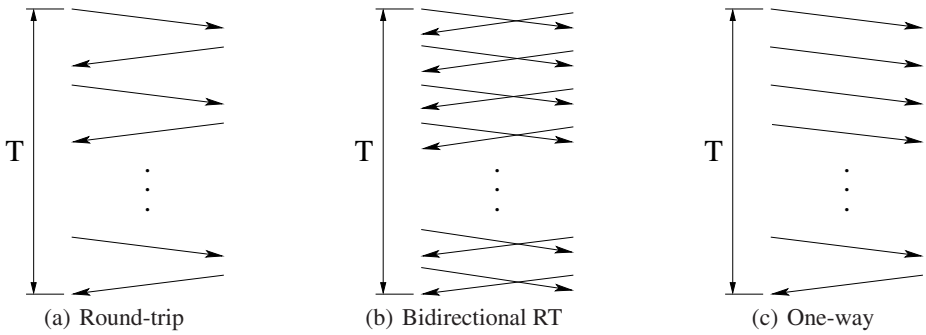


Fig. 1. Performance benchmarks

ping-pong which is used to highlight the capability of a library to take benefits of bidirectional links. The last benchmark (figure 1(c)) is the burst which aims at sending as many messages as possible regardless they have been received or not by the counter part.

The measurement protocol is as follows: for each message size, each benchmark is run ten times. The duration of a run is one minute (this ensures a high consistency in results and we have determined that all confidence intervals are greater than or equal to 90%). The system time is registered before the first message is sent (t_1) and after the last message is received on the same node (t_2). Let the elapsed time t be the time difference between both. For a given run, let s be the size of messages and n be the number of effectively transmitted messages.

Let the end-to-end latency L (in the following we use the term latency) be the ratio between the elapsed time t and the number of effectively transmitted messages n . And let the user throughput T (in the following we use the term throughput) be the ratio between the amount of data (number of effectively transmitted messages times the size of a message) and the elapsed time.

$$t = t_2 - t_1 \quad L = \frac{t}{n} \quad T = \frac{n \times s}{t}$$

Since the performance for both RWAPI and MPI are almost the same for messages size greater than or equal to 1 MB, we do not include data for larger messages.

Our Myrinet-based cluster is POETS, one of the clusters of the Institut National des Télécommunications. POETS is composed of eight nodes connected using both a Myrinet interconnection network for data and a Gigabit Ethernet interconnect as a control network. Myrinet adaptors are 133-MHz LANai-9 (M3S-PCI64B) with a PCI connector. The host adaptor is equipped with 2 MB of memory. The firmware used for testing was GM 2.0.9. Apart from GM, the port of MPICH on top of GM called MPICH-GM version 1.2.6..14b for Linux x86 was also installed. Each node includes a 800-MHz Intel Pentium III processor with 1.2 GB of memory. The front-end of the cluster is an extra node with almost the same characteristics except that it includes no Myrinet NIC.

The InfiniBand-based cluster is MUSES. It is composed of 16 nodes connected using both an InfiniBand 4× interconnection network for data and a Gigabit Ethernet interconnection network as a control network. Each node includes a 1.8-GHz AMD Opteron processor with a 1-MB cache and 2 GB of memory. For mass storage, each node is associated a 40-GB IDE hard drive, except for the first node (the front-end node) which is associated a 80-GB IDE hard drive; note that users accounts are stored on the front-end. We compared our implementation with two other existing libraries on top of the InfiniBand Technology: VAPI [17], the native interface available on top of InfiniBand, and MPICH developed on top of VAPI for InfiniBand.

The MPI benchmark uses the send/receive primitives instead of put with both Mpich-GM and Mpich-VAPI. Send/receive is based on the rendez-vous model which requires that the receive request should be posted before the send request. Otherwise, a blocking wait or a message buffering is performed on the receiver side. Moreover, the eager-message size is kept to the default value. Finally, the RWAPI benchmark and the MPI benchmark use the same buffer for all the iterations of the communications which allows that the memory pages are pinned only one time.

5 Performance Analysis

Figure 2 presents the latency of RWAPI-GM and MPI-GM for various message sizes. Note that for readability reasons, a logarithmic scale have been used for latency curves. All these graphs highlight that, in the general case, performance with RWAPI are usually better than those with MPI. Exceptions are for the One-Way benchmark for messages which size ranges from 1 kB to 16 kB and for the Round-Trip benchmark for messages which size ranges from 4 bytes to 64 bytes. This may be due to the fact that MPI is using Write-Combining to copy data from the host memory to the NIC memory for small messages avoiding the overhead of the DMA start-up. Regarding the latency, an interesting result is that for the Round-Trip benchmark (see figure 2(a)), the minimum latency for RWAPI is as low as 5.1 μs. As a comparison, the minimum latency for MPI is 7.9 μs and is achieved using the Bidirectional Round-Trip (see figure 2(b)).

Figure 3 shows the throughput of RWAPI and MPI for various message sizes. All these graphs highlight that, in the general case, performance with RWAPI are usually better than those with MPI.

These graphs are highlighting three very important results. First, RWAPI is able to achieve a maximum throughput of up to 1.78 Gb/s for large messages (see figure 3(a))

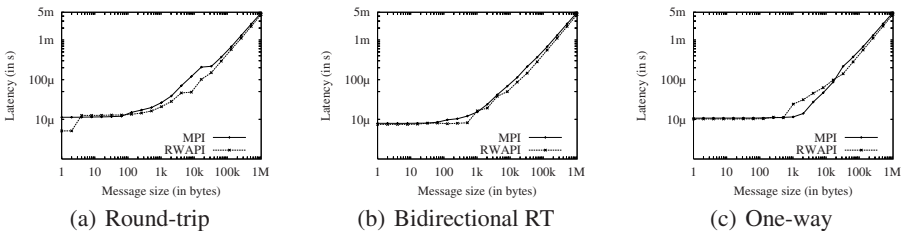


Fig. 2. Myrinet Latency

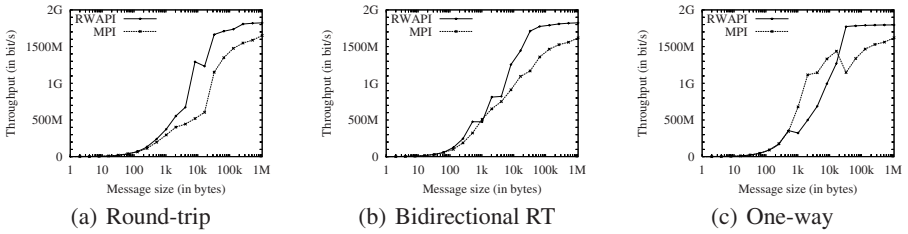


Fig. 3. Myrinet Throughput

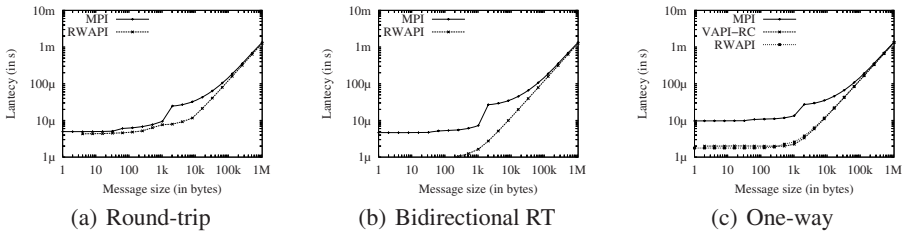


Fig. 4. InfiniBand Latency

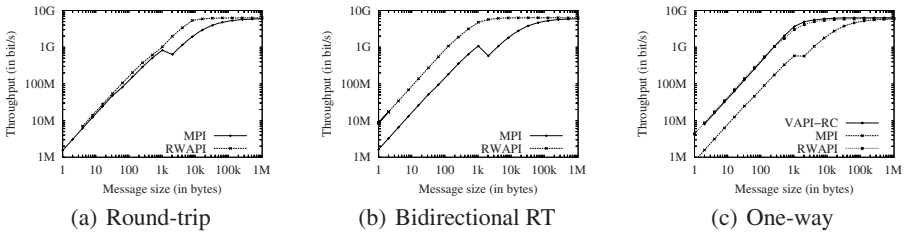


Fig. 5. InfiniBand Throughput

and figure 3(b) for Round-Trip and Bidirectional Round-Trip benchmarks respectively). As a comparison, the maximum throughput provided by MPI is 1.62 Gb/s. This means that MPI cannot offer more than 91% of the maximum throughput offered by RWAPI. In other words, RWAPI is able to deliver large messages 9.8% faster than MPI (in fact, this is not highlighted with latency graphs on figure 2 as a logarithmic scale is used for messages larger than 1 kB).

RWAPI provides a larger part of the bandwidth for smaller messages than MPI. Typically, RWAPI is able to achieve 90% of the maximum throughput for 32-kB messages as MPI requires messages of 256 kB to do the same.

With InfiniBand libraries, in a general way, figure 5 and figure 4 show that RWAPI-VAPI performance are always better than MPI-VAPI performance. More specifically, the maximum ratio between the minimum latency achieved by RWAPI and the minimum latency achieved by MPI is up to 5.5 x for small messages (ie. $1.76 \mu\text{s}$ for RWAPI and $9.71 \mu\text{s}$ for MPI using the one-way benchmark).

Both RWAPI and MPI are able to achieve the maximum user throughput for long messages. However, RWAPI is able to provide this maximum user throughput for messages as short as 4 kB while MPI cannot do the same for messages smaller than few hundreds kilo-bytes.

Finally, the curves on figure 5 and figure 4 show that there is an important difference in the management of short and long messages for MPI represented with a knee between 1 and 2 kB.

6 Conclusion

A previous study led us to design our own implementation of the remote write. In this paper, we have presented the design and implementation of RWAPI over Myrinet-2000 and InfiniBand Interconnects. This design takes full advantages of the network hardware such as OS-Bypass and RDMA, thus eliminating the involvement of the operating system and the receive process. In addition, it allows the overlap between communications and computations.

To decrease the latency of small messages, we used the Programmed-IO facility instead of RDMA to copy data to the network card, removing one long-startup-time DMA transaction.

RWAPI over Myrinet achieves a low latency of 5.1 μs and a high user throughput of 2.43 Gb/s even for relatively short messages (2.26 Gb/s is available for 32-kB messages). On the same platform, the lowest latency provided by MPI is 7.9 μs and the maximum user throughput provided by MPI represents 90% of the maximum user throughput provided by RWAPI (this means that message transfer with RWAPI is up to 9.8% faster than with MPI).

RWAPI over InfiniBand design can achieve a low latency (about 1.76 μs) and a high user throughput (more than 6.3 Gb/s, ie the maximum user bandwidth) even for short messages. As a comparison, the lowest latency provided by MPI over the same platform is 4.96 μs and the maximum user throughput cannot be achieved for messages smaller than several hundreds of kilo-bytes.

Note that the lowest InfiniBand communication layer (VAPI) let the user choose between producing events for transfer operation completion or not. This does not suit RWAPI as disabling events does not allow the user to be informed about the completion of the send and enabling events adds an extra overhead due to the unnecessary receive completions.

Currently, RWAPI uses RC as the type of service with InfiniBand packet management. RC requires a connection between each remote HCA and thus consumes much HCA memory resources. Consumed memory is mainly used to store data reassembly informations for each connection. To achieve better scalability, we are working on applying the RD type of service which bypasses any connection management and maintains a reliable communication.

As a short-term, we have planned to optimize the `rwapi_write` operation by using either PIO, Write-Combining or DMA to copy user data from the host memory to the NIC memory; PIO operations for very small messages, Write-combining for medium messages and DMA for large messages since it involves an expensive start-up overhead.

As a mid-term, we have planned to implement another version of RWAPI which can run simultaneously over heterogeneous architectures composed of different network types and different machine characteristics.

References

1. InfiniBand Trade Association: The InfiniBand Architecture, Specification Volume 1 & 2, Release 1.0.a (2001)
2. Boden, N., Cohen, D., Flederman, R., Kulawik, A., Seitz, C., Selzovic, J., Su, W.: Myrinet - A Gigabit-per-Second Local-Area Network. 15, 29–36 (1995)
3. Beecroft, J., Addison, D., Petrini, F., McLaren, M.: Quadrics QsNet II: A network for Supercomputing Applications. In: Hot Chips 15, Stanford University, Palo Alto, CA (2003)
4. Whitby-Stevens, C., et al.: IEEE Draft Std P1355 — Standard for Heterogeneous Interconnect — Low Cost Low Latency Scalable Serial Interconnect for Parallel System Construction (1993)
5. Greiner, A., Desbarbieux, J., Lecler, J., Potter, F., Wajsbürt, F., Penain, S., Spasevski, C.: PCI-DDC Specifications. Laboratoire MASI, Université Paris VI (1996)
6. ANSI/VITA 26-1998: Myrinet-on-VME Protocol Specification (1998)
7. Vivo, A.D.: A Light-Weight Communication System for a High Performance System Area Network. PhD thesis, Università di Salerno - Italy (2001)
8. Myricom: GM: A message-passing system for myrinet networks 2.0.12 (1995)
9. Message Passing Interface Forum MPIF: MPI-2: Extensions to the Message-Passing Interface. Technical Report, University of Tennessee, Knoxville (1996)
10. Fenyo, A.: Conception et réalisation d'un noyau de communication bâti sur la primitive d'écriture distante, pour machines parallèles de type grappe de PCs. Thèse de doctorat, UPMC LIP6, Paris, France (2001)
11. Desbarbieux, J.L., Gluck, O., Zerrouki, A., Fenyo, A., Greiner, A., Wajsbürt, F., Spasevski, C., Silva, F., Dreyfus, E.: Protocol and performance analysis of the mpc parallel computer. In: IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium, p. 52. IEEE Computer Society, Washington, DC (2001)
12. Renault, E., David, P.: PAPI message passing library: Comparison of performance in user and kernel level messaging. In: Sakellariou, R., Keane, J.A., Gurd, J.R., Freeman, L. (eds.) Euro-Par 2001. LNCS, vol. 2150, pp. 717–721. Springer, Heidelberg (2001)
13. Bonachea, D.: Gasnet specification, v1.1. Technical report, Berkeley, CA, USA (2002)
14. Turner, D., Selvarajan, S., Chen, X., Chen, W.: The MP_Lite Message-passing Library. In: Akl, S.G., Gonzalez, T. (eds.) the proceedings of the Parallel and Distributed Computing and Systems. PDCS 2002, Cambridge, USA, pp. 373–235 (2002)
15. Ben Fredj, O., Renault, E.: A qualitative analysis of the critical's path of communication models for next perform implementations of high-speed interfaces. In: Sobh, I.T., Khaled Elleithy, E. (eds.) Advances in Systems, Computing Sciences and Software Engineering: Proceedings of SCSS 2005, pp. 70–77. Springer, Secaucus, NJ (2006)
16. Ben Fredj, O., Renault, E.: Rwapi over infiniband: Design and performance. In: 5th International Symposium on Parallel and Distributed Computing (ISPDC 2006), July 6-9, 2006, pp. 50–57. IEEE Computer Society, Timisoara, Romania (2006)
17. Mellanox Technologies Inc: Mellanox ib-verbs api (vapi) (2001)