Evaluation of the Angara Interconnect Prototype TCP/IP Software Stack: Implementation, Basic Tests and BeeGFS Benchmarks

Yuri Goncharuk[1], Yuri Grishichkin[2], ✉Alexander Semenov[1][0000−0003−4878−6287], Vladimir Stegailov[2,3,4][0000−0002−5349−3991], and Vasiliy Umrihin[1]

[1] JSC NICEVT, Moscow, Russia
[2] Joint Institute for High Temperatures of RAS, Moscow, Russia
[3] National Research University Higher School of Economics, Moscow, Russia
[4] Moscow Institute of Physics and Technology, Dolgoprudny, Russia
goncharuk@nicevt.ru, gyg@jiht.ru, ✉semenov@nicevt.ru, stegailov.vv@phystech.edu, umrihin@nicevt.ru

Abstract. In the paper, we present a prototype implementation of the TCP/IP software stack over the Angara high performance interconnect. Our approach is to use the standard TCP/IP stack implementation from the Linux kernel, while we implement in the Linux kernel an Ethernet network device driver for the Angara interconnect adapter. The paper presents the latency and bandwidth results and the results of the IO500 suite benchmarks of the distributed storage deployed with BeeGFS on 20 nodes of the Fisher supercomputer in JIHT RAS.

Keywords: Angara · TCP/IP · Ethernet · BeeGFS · IO500

1 Introduction

Parallel file system (PFS) is one of the most essential building blocks of the persistent storage in high performance computing (HPC) infrastructure. It provides fast global access to large volumes of data and ensures data persistence through a high number of distributed storage devices. One of the most critical components having a direct impact on the performance of HPC systems and PFSs is the interconnection network (interconnect). Parallel file system clients accessing data from the file system, communicate with the storage servers via any TCP/IP based connection or via remote direct memory access (RDMA) capable interconnects like InfiniBand [1], Omni-Path [2], Slingshot [3] and RDMA over Converged Ethernet (RoCE), which is a part the Infiniband specification [1].

Currently, the market is dominated by Infiniband. InfiniBand is an open standard for high performance network communications. The most commonly used InfiniBand implementations rely on NVIDIA Mellanox hardware, with switches

typically arranged in a fat tree topology. HDR 200 Gb/s is the sixth generation of the NVIDIA InfiniBand architecture. HDR has 0.6 us low-level latency [4], the obtained MPI latency is 1 us [5]. Infiniband can support TCP/IP protocol as IPoIB (IP over Infiniband), which implements regular networking (through the Linux kernel stack) over Infiniband fabric by wrapping L3/L4 TCP/IP headers with Infiniband headers. Note that L2/L3/L4 layers correspond to the 2nd/3rd/4th layers of the OSI model [6]. The second way is RoCE, which implements RDMA over Ethernet fabric by wrapping Infiniband packets with L2/L3/L4 headers.

Cray (HPE) released the Slingshot interconnection network [3]. Slingshot uses an optimized Ethernet protocol, which allows it to be interoperable with standard Ethernet devices while providing high performance to HPC applications. Slingshot switches have ports with 200 Gbit/s each and support arbitrary network topologies, the default topology is Dragonfly [7]. The low-level latency of Slingshot is 1.85 us between two nodes.

The focus of our work is the Angara interconnect. Angara [8,9] is the low-latency, high bandwidth interconnect with 4D torus topology, the obtained MPI latency between two adjacent nodes is 0.85 us. The Angara-C1 and Desmos [9] cluster systems are based on the Angara interconnect. During the last several years, the Angara interconnect has obtained a history of practical usage [9,10,11], [12,13].

In this paper, we present a prototype implementation of the TCP/IP software stack over the Angara high performance interconnect. For the sake of clarity, our approach is to use the standard TCP/IP stack implementation from the Linux kernel, while we implement in the Linux kernel the Ethernet device driver for the Angara interconnect adapter. The goal is to support parallel file systems on the Angara-based HPC systems.

PFS performance evaluation with network aspects are quite rare in computer science. Papers [14] and [15] with BeeGFS, Ceph, GlusterFS, OrangeFS PFSs performance analysis should be highlighted. Performance improving of large scale geophysical applications using BeeGFS is presented in [16]. Infiniband and TCP/IP software stack are used as a network transport. The emergence in 2017 of the IO500 list [17] played a big role in drawing attention to PFS performance analysis. IO500 is a comprehensive benchmark suite to track storage performance and storage technologies of supercomputers, organized by the Virtual Institute for I/O (VI4IO) [18]. Several works [19,20] address the IO500 performance on HPC systems.

The paper is structured as follows. In section 2 the brief Angara interconnect architecture is given, then the implementation of the Ethernet network device over Angara is described. Section 3 presents the hardware configuration of the Fisher supercomputer and software settings. Then the results of bandwidth, latency and IO500 tests are given and discussed. Section 4 concludes the paper.

2 The Angara TCP/IP Implementation

The TCP/IP software stack for the Angara interconnect is supported by the standard TCP/IP implementation from the Linux kernel and three implemented in this work Linux kernel modules: angara_netdev, angara_rdma and angara_router. The main module is angara_netdev, which implements the Ethernet interface, i.e. the channel or second layer of the OSI model [6].

2.1 The Angara Interconnect Architecture

The Angara interconnect is a Russian-designed communication network with 4D torus topology. The interconnect ASIC was developed by JSC NICEVT and manufactured by TSMC with the 65-nm process. An Angara packet format provides a possibility to address 32K nodes.

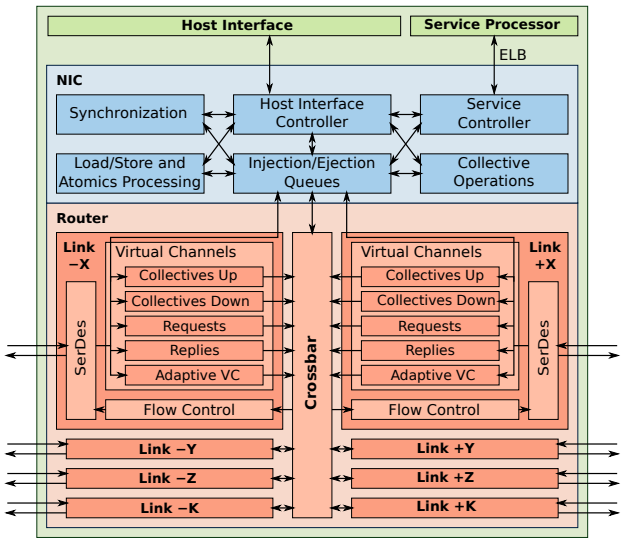

Fig. 1: The Angara ASIC architecture.

Figure 1 presents the Angara ASIC chip, which consists of an adapter and a router. The router contains 8 link blocks that are connected via a crossbar. The crossbar can simultaneously transmit flits (128 bits) from links if there are no conflicts.

The Angara chip supports simultaneous operations with multiple threads/processes of a user task; it is implemented as several injection channels available for use by independent packet buffers.

Each node has a dedicated memory region available for remote access from other nodes. The network adapter at the hardware level supports PIO and

RDMA modes. In PIO mode a processor creates network packets and sends them to the network using Angara PUT operation, thus consuming the resources of a processor core. On the contrary, RDMA (Remote Direct Memory Access) mode does not involve the CPU. Rather, the Angara adapter moves data directly to and from memory, bypassing the CPU altogether. Angara has RDMA write and read operations.

2.2 Ethernet Network Device Driver Implementation

The Ethernet network device for the Angara interconnect is implemented by three Linux kernel modules: angara_netdev, angara_rdma and angara_router. The main module is angara_netdev, which implements the Ethernet interface. The angara_netdev module uses functions that are exported by the angara_rdma and angara_router kernel modules. The angara_rdma module implements a native Angara network messaging interface with RDMA support in the kernel space. The angara_router module manages hardware resources of the Angara network adapter, including access to injection pipelines, control registers, etc.

The angara_netdev module is implemented as an Ethernet device driver with the following features. The Media Independent Interface (MII) [21] features, which include physical layer control and media access link layer control, are not implemented. These functions include managing the speed of the network adapter (10M / 100M / 1G / 10G / 40G), duplex mode (Full / Half), changing the Ethernet frame size (MTU) during the driver execution, monitoring the state of the physical link. The physical layer in the Angara network is custom designed, and at the moment there is no need to implement the standard Ethernet capabilities for managing the physical layer. In the future, these functions may be implemented. Also, the ARP protocol is implemented entirely in software.

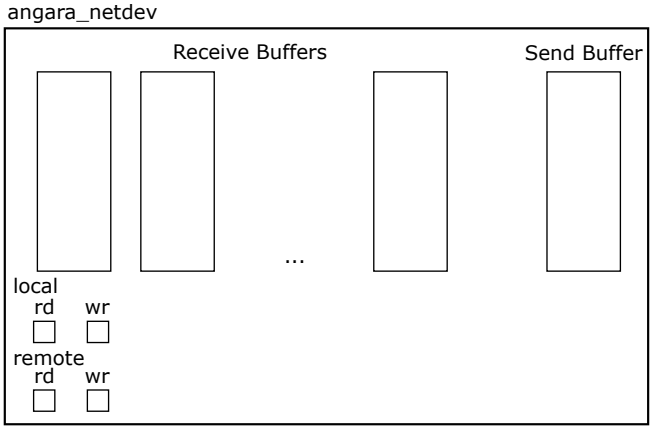

Fig. 2: The angara_netdev module memory scheme.

Receiving and sending data in the angara_netdev module corresponds to the implementation of the Ethernet layer. The angara_netdev module in memory on each node has a receive ring buffer for each other network node, the default buffer size is 8192 4 KB packets, see Fig. 2. There is a ring buffer for sending data to for all other nodes, its size is 8192 packets of a maximum 4 KB size. Local read and write counters and remote read and write counters are supported for each receive buffer. Also in the angara_netdev module there is a Forwarding Database (FDB) table with mapping MAC addresses and physical numbers (NODENUM) of nodes in the Angara network.

The message send consists of the following stages:

1. The angara_netdev module (angara0 interface) receives a socket buffer from the Linux kernel, which is represented by the structure `sk_buff` [22]. The IP address of the recipient is retrieved from the resulting structure.
2. Using the retrieved destination IP address, the ARP table is searched for MAC address by IP address.
3. The received MAC address of the recipient is used to search in the FDB table of the angara_netdev module for the corresponding physical NODENUM number of the destination node in the Angara network.
4. From the `sk_buff` structure data payload is retrieved, as well as its size. The received data is combined and transmitted as the first PUT sending over the Angara network to the node with the destination physical NODENUM number. The sending occurs with the requirement that the local write counter is less than the remote read counter.
5. The second PUT sending to the NODENUM node includes the incremented value of the local write counter and writes it to the corresponding memory location of the remote write counter on the remote NODENUM node. At the same time, the local read counter is also sent, which is written to the remote read counter of the remote NODENUM node to inform it about how many packets the current node has read.

If the recipient MAC address is missing, then the angara_netdev module perform a software broadcast ARP request to all possible physical nodes in the Angara network. The received ARP-request is processed by the angara_netdev module on a node, and sends a response, which eventually ends up in the system ARP table, and the NODENUM of the node goes into the FDB table of the angara_netdev module.

The message send can be performed in two modes:

1. Using the sending thread (parameter `tx_threadless=0` of the angara_netdev module);
2. Without using the sending thread (parameter `tx_threaless=1` of the angara_netdev module).

In the first mode, during the initialization of the angara0 network interface, a TX thread is created, it processes the send buffer. The module's send function adds data to this buffer, and the TX thread performs the described transmission.

In the second mode, the described transmission occurs immediately when the `sk_buff` structure is received from the kernel, but this mode is the softirq interrupt handling mode [23], and long-term processing is undesirable. Since message latency is higher for the first mode, the second mode is used in this paper. The current implementation uses single injection channel of the Angara ASIC. Use of several injection channel allows to improve network bandwidth.

The message receive is organized as follows. During the raising of the angara0 network interface, an RX thread is created, which permanently polls the remote write counter of the receive ring buffer for each possible network node and compares it to the local read counter.

The polling discipline for receive buffers by the RX thread is round robin. During each iteration of the RX thread, if a new buffer element is found, the tasklet is launched. A tasklet [23] is a lightweight thread that does not have its own context, it runs in a separate kernel thread and completely performs the receive function for the specified sender node. The total number of launched tasklets can not be more than the total number of nodes in the network. After data processing, in the absence of new data the tasklet is destroyed.

The tasklet processes packets on each iteration of the receive loop for the difference between the remote write counter and the local read counter. At each iteration of this loop, the tasklet does the following:

1. Allocates memory for the `sk_buff` structure, write payload to this structure.
2. Sends the created structure using the `netif_rx()` to the kernel network stack, which corresponds to the 3rd layer of the OSI model.

The more participants in the data exchange, the longer delay between processing of each individual sender node. To speed up the processing of specific sender nodes, the `angara_parts` parameter (short for Angara participants) has been introduced. This parameter allows to set the list of active members of the Angara network, note that messages from other network nodes will be processed with lower priority. At the same time, it is a disadvantage that the receive buffers are created for all network nodes, and not only for active participants. This shortcoming is planned to be eliminated in future work.

3 Experimental Evaluation

This section presents the performance evaluation of the proposed TCP/IP software stack for the Angara interconnect on the Fisher supercomputer.

3.1 Hardware and Software Setup

The main details about the Fisher supercomputer are given in Table 1 and illustrated in Figure 3. Fisher has a segment with Infiniband FDR and air cooling and a segment with Angara and immersion cooling [24]. The Angara segment is divided into two equal size partitions, the first one is based on the first AMD EPYC

Table 1: The main characteristics of the segment of the Fisher supercomputer used for EoA testing.

Compute Nodes	ang[1-20]
Chassis	Gigabyte H262-Z62
Processor / Memory	2 x Epyc 7301 16c / 128 GB
Storage	Apacer AS2280P2 240GB
Interconnect	Angara switch, 1 Gbit/s Ethernet
OS	openSUSE Leap 15.2
Kernel version	5.3.18-lp152.87-preempt
MPI	MPICH 3.2 for Angara

processor microarchitecture generation called Naples, the second one is based on the second AMD EPYC processor microarchitecture generation called Rome. Each Angara partition has an Angara ES8433 switch, each node of the Angara segment has a low-profile Angara ES8432 network adapter. ES8433 switches are connected by 4 links.

For the evaluation we have initialized Ethernet Angara interface on 20 nodes with AMD EPYC 7301 processers.

BeeGFS [25] is a parallel cluster file system. It was originally developed for High Performance Computing. BeeGFS transparently spreads user data across multiple servers. BeeGFS separates metadata from user file chunks on the servers. The metadata is the "data about data", such as access permissions, file size and the information about how the user file chunks are distributed across the storage servers. BeeGFS clients accessing data from the file system, communicate with the storage servers via any TCP/IP based connection or via RDMA-capable networks.

We use quick deploy with BeeGFS On Demand (BeeOND) option with default parameters that allows to run BeeGFS instances temporary exactly for the runtime of the compute job. BeeGFS version is 7.2.3.

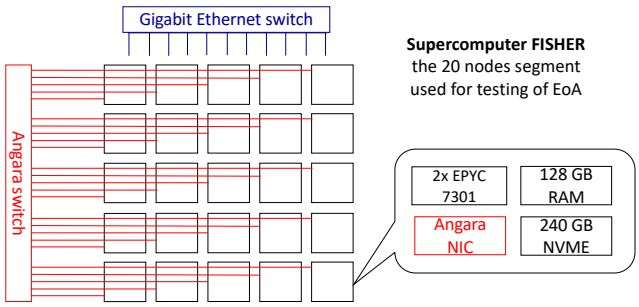

Fig. 3: The segment of the Fisher supercomputer used for EoA testing.

3.2 Benchmarks

We use the MPI-based osu_latency benchmark, version 5.9 for latency evaluation on 2 neighbouring nodes. For bandwidth evaluation we use iperf with 5 parallel client threads.

Table 2: IO500 benchmark suite components.

Component	**Tests**	**Description**
IOR 'easy'	ior_easy_write, ior_easy_read	Bandwidth for well-formed large sequential I/O patterns
IOR 'hard'	ior_hard_write, ior_hard_read	Bandwidth for unaligned (47001 bytes) operation from each client process to a single file
mdtest 'easy'	mdtest_easy_delete, mdtest_easy_stat, mdtest_easy_write	Metadata operations on 0-byte files, using separate directories for each MPI task
mdtest 'hard'	mdtest_hard_delete, mdtest_hard_stat, mdtest_hard_write, mdtest_hard_read	Metadata operations on small (3901 byte) files in a shared directory
Find	find	Finding relevant files through directory traversals

IO500 [17] is a comprehensive benchmark suite to track storage performance and storage technologies of supercomputers. The IO500 benchmark suite consists of data (IOR) and metadata (mdtest) components as well as a parallel namespace scanning test (find), and calculates a single ranking score for comparison. Table 2 provides a list of IO500 tests. The tests represent the best and worst possible scenarios for bandwidth and metadata in the form of 'easy' and 'hard' use cases respectively. The individual IO500 tests are combined as a score using a geometric mean to find the central tendency among the various metrics. While a top score does not indicate that all applications can achieve that performance, the range from the 'hard' to 'easy' on bandwidth and metadata gives bounds for users can expect [19]. The hard and easy tests are carefully interleaved and timed to 5 minutes for create-style operations representing the typical 90% forward progress requirement used in platform purchases.

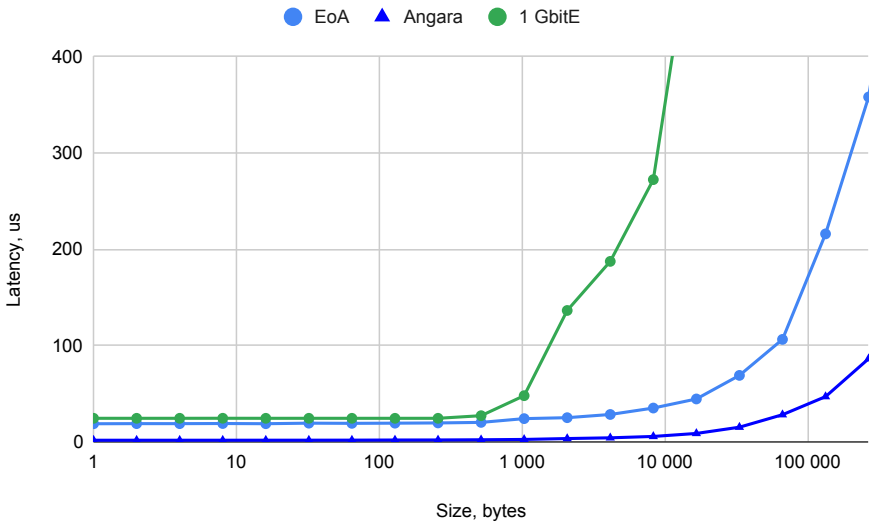

Fig. 4: The obtained osu_latency results on two Fisher nodes. EoA – Ethernet over Angara, Angara – native Angara protocol, 1 GbitE – 1 Gbit/s Ethernet network.

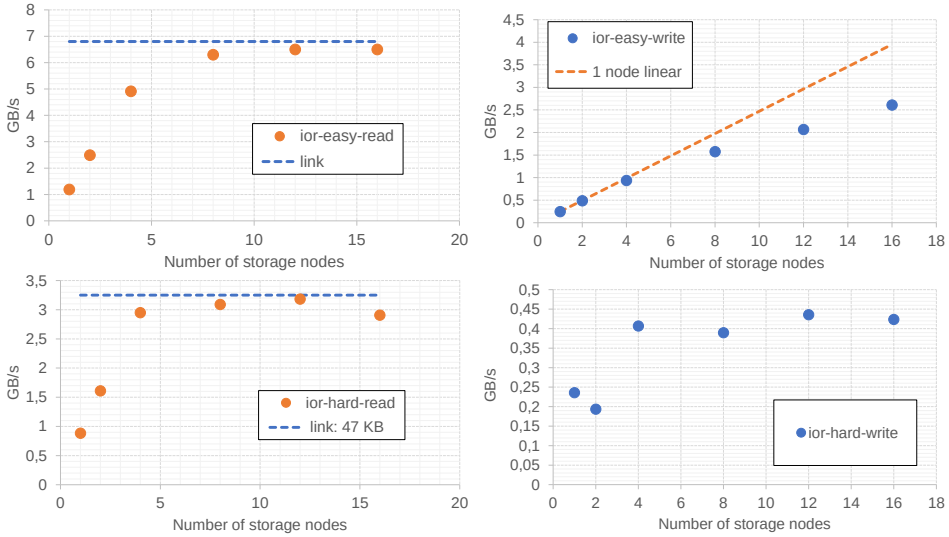

Fig. 5: IO500 results for the bandwidth IOR tests (at 4 client nodes with 16 client processes per node) as a dependence on the number of Fisher nodes that form the BeeOND distributed storage.

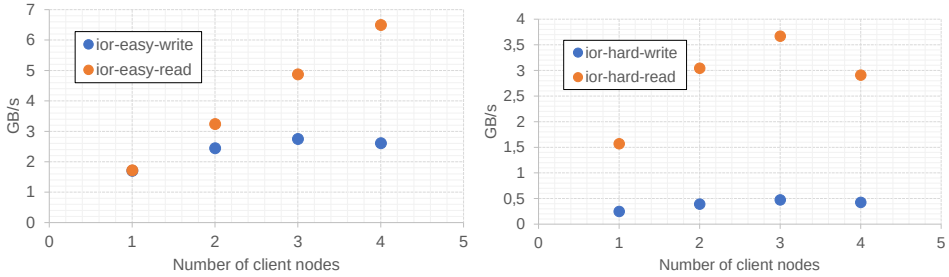

Fig. 6: IO500 results for the bandwidth IOR tests of the BeeOND distributed storage (16 nodes with 1 storage target per node) as a dependence on the number of client Fisher nodes (at 16 MPI processes per node).

3.3 Performance Results

Figure 4 shows the obtained osu_latency results on two Fisher nodes. For 0-byte message we have obtained for Ethernet over Angara (EoA) 19 us, for native Angara protocol (Angara) 1.7 us, for 1 Gbit/s Ethernet (1 GbitE) 25.4 us. Note that Angara interface is initialized on all 20 Fisher nodes. The large performance gap between for Ethernet over Angara and native Angara shows the possibility of further improvement of the latency.

The bandwidth of the Ethernet over Angara obtained by the iperf3 test is approximately 15 Gbit/s.

Table 3: IO500 performance results on the Fisher supercomputer.

Test	**Metric**	**1 GbE**	**EoA**	**EoA / 1 GbE**
ior-easy-write	GB/s	0.46	2.61	5.69
mdtest-easy-write	kIOPS	15.18	17.17	1.13
ior-hard-write	GB/s	0.17	0.42	2.44
mdtest-hard-write	kIOPS	5.10	3.8	0.75
find	kIOPS	175.19	124.55	0.71
ior-easy-read	GB/s	0.45	6.5	14.5
mdtest-easy-stat	kIOPS	74.60	73.69	0.99
ior-hard-read	GB/s	0.46	2.91	6.38
mdtest-hard-stat	kIOPS	67.31	70.3	1.04
mdtest-easy-delete	kIOPS	14.20	10.18	0.72
mdtest-hard-read	kIOPS	14.59	15.03	1.03
mdtest-hard-delete	kIOPS	4.19	5.95	1.42
Total score		2.8	6.35	2.38

Figure 5 shows the dependence of the IOR tests results from the IO500 suite on the number of nodes that form the distributed storage using BeeOND. These

tests have been performed for the fixed number of 4 client nodes with 16 client processes per node. We see that io-easy-read and io-hard-read saturate already for 8 and 4 storage nodes, respectively. This saturation can be explained by the limited aggregated bandwidth of 4 EoA links that come to 4 client nodes for large messages (io-easy-read) and 47 KB messages (io-hard-read). The bandwidth of the Angara link does not limit the io-easy-write performance, for 1–4 nodes the main bottleneck is the hard disk performance for write operation.

Figure 6 show the dependence of the same IOR tests results on the number of client nodes. These tests have been performed for the fixed number of 16 storage nodes with 1 storage target per node. We see that using 4 client nodes on io-easy-read one can not saturate the bandwidth of this distributed storage.

Table 3 shows IO500 performance results for the Fisher supercomputer. We have deployed BeeGFS storage on 16 Fisher nodes, including 1 metadata node. We use 4 client nodes, 16 processes are executed on each client node. The total number of client processes is 64. The advantage of Ethernet over Angara network (EoA) on large messages is approximately 15 times compared to 1 Gbit/s Ethernet (1 GbE). The relative read results for EoA are better than the write results, this can be explained by the bottleneck in the hard disk's poor write performance. For metadata and find tests the results are approximately the same on EoA and 1 GbE, except for mdtest-easy-write and mdtest-hard-write tests. The Angara's power is low latencies, which are important for small metadata requests, therefore detailed profiling is needed.

4 Conclusion

This work reports of a fully functional TCP/IP software stack implemented for the Angara interconnect using the prototype Ethernet over Angara driver implementation. The benchmarks of the distributed storage based on this EoA prototype do not show any evident performance limitations that could be attributed to the performance of the Angara interconnect within the EoA framework.

In our future work we plan to address the TCP/IP over Angara bandwidth, latency as well as a detailed profiling of IO500 results. The next Ethernet over Angara implementation will use multiple injection channels and RDMA operations, which will allow to improve the network bandwidth.

References

1. InfiniBand Trade Association. InfiniBand Architecture Specification. Release 1.0 (2000)
2. Birrittella, M.S., Debbage, M., Huggahalli, R., Kunz, J., Lovett, T., Rimmer, T., Underwood, K.D., Zak, R.C.: Intel® omni-path architecture: Enabling scalable, high performance fabrics. In: 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects. pp. 1–9. IEEE (2015)
3. De Sensi, D., Di Girolamo, S., McMahon, K.H., Roweth, D., Hoefler, T.: An in-depth analysis of the Slingshot interconnect. In: SC20: International Conference

for High Performance Computing, Networking, Storage and Analysis. pp. 1–14. IEEE (2020)
4. Introducing 200G HDR InfiniBand Solutions http://mvapich.cse.ohio-state.edu/benchmarks/. Mellanox Technologies (2019)
5. Ruhela, A., Xu, S., Manian, K.V., Subramoni, H., Panda, D.K.: Analyzing and understanding the impact of interconnect performance on HPC, Big Data, and deep learning applications: A case study with Infiniband EDR and HDR. In: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). pp. 869–878. IEEE (2020)
6. Zimmermann, H.: OSI reference model-the ISO model of architecture for open systems interconnection. IEEE Transactions on communications 28(4), 425–432 (1980)
7. Kim, J., Dally, W.J., Scott, S., Abts, D.: Technology-driven, highly-scalable dragonfly topology. In: 2008 International Symposium on Computer Architecture. pp. 77–88. IEEE (2008)
8. Simonov, A., Brekhov, O.: Architecture and functionality of the collective operations subnet of the Angara interconnect. In: Vishnevskiy, V.M., Samouylov, K.E., Kozyrev, D.V. (eds.) Distributed Computer and Communication Networks. pp. 209–219. Springer International Publishing, Cham (2020)
9. Stegailov, V., Dlinnova, E., Ismagilov, T., Khalilov, M., Kondratyuk, N., Makagon, D., Semenov, A., Simonov, A., Smirnov, G., Timofeev, A.: Angara interconnect makes GPU-based Desmos supercomputer an efficient tool for molecular dynamics calculations. The International Journal of High Performance Computing Applications 33(3), 507–521 (2019)
10. Akimov, V., Silaev, D., Aksenov, A., Zhluktov, S., Savitskiy, D., Simonov, A.: FlowVision scalability on supercomputers with Angara interconnect. Lobachevskii Journal of Mathematics 39(9), 1159–1169 (2018)
11. Khalilov, M., Timofeev, A.: Optimization of MPI-process mapping for clusters with Angara interconnect. Lobachevskii Journal of Mathematics 39(9), 1188–1198 (2018)
12. Polyakov, S., Podryga, V., Puzyrkov, D.: High performance computing in multiscale problems of gas dynamics. Lobachevskii Journal of Mathematics 39(9), 1239–1250 (2018)
13. Tolstykh, M., Goyman, G., Fadeev, R., Shashkin, V.: Structure and algorithms of SLAV atmosphere model parallel program complex. Lobachevskii Journal of Mathematics 39(4), 587–595 (2018)
14. Kunkel, J.M., Kuhn, M., Ludwig, T.: Exascale storage systems: an analytical study of expenses. Supercomputing frontiers and innovations 1(1), 116–134 (2014)
15. Mills, N., Feltus, F.A., Ligon III, W.B.: Maximizing the performance of scientific data transfer by optimizing the interface between parallel file systems and advanced research networks. Future Generation Computer Systems 79, 190–198 (2018)
16. Brzenski, J., Paolini, C., Castillo, J.E.: Improving the i/o of large geophysical models using pnetcdf and beegfs. Parallel Computing 104, 102786 (2021)
17. Kunkel, J.: IO500 (2020), https://www.vi4io.org/io500/start. Accessed: 30.04.2022.
18. Kunkel, J., Lofstead, G.F., Bent, J.: The virtual institute for I/O and the IO-500. Tech. rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States) (2017)
19. Liem, R., Povaliaiev, D., Lofstead, J., Kunkel, J., Terboven, C.: User-centric system fault identification using IO500 benchmark. In: 2021 IEEE/ACM Sixth International Parallel Data Systems Workshop (PDSW). pp. 35–40. IEEE (2021)

20. Hennecke, M.: Daos: A scale-out high performance storage stack for storage class memory. Supercomputing frontiers p. 40 (2020)
21. Ieee standards for local and metropolitan area networks: Supplement - media access control (MAC) parameters, physical layer, medium attachment units, and repeater for 100 Mb/s operation, type 100BASE-T (clauses 21-30). IEEE Std 802.3u-1995 (Supplement to ISO/IEC 8802-3: 1993; ANSI/IEEE Std 802.3, 1993 Edition) pp. 1–415 (1995)
22. Gonsalves, T.: Linux network device drivers: an overview (2020), http://students.iitmandi.ac.in/~tag/csdoc/Linux_Network_Device_Drivers_Overview_2020.pdf
23. Rothberg, V.: Interrupt handling in Linux (2015)
24. Dlinnova, E., Biryukov, S., Stegailov, V.V.: Energy consumption of MD calculations on hybrid and cpu-only supercomputers with air and immersion cooling. In: PARCO. pp. 574–582 (2019)
25. Herold, F., Breuner, S.: An introduction to BeeGFS (2018), https://www.beegfs.io/docs/whitepapers/Introduction_to_BeeGFS_by_ThinkParQ.pdf. Accessed: 01.05.2022.