Data structures access model for remote shared memory

Anatoliy Nyrkov[1,*], *Konstantin* Ianiushkin[1], *Andrey* Nyrkov[1], Yulia Romanova[1], and *Vagiz* Gaskarov[1]

[1]Admiral Makarov State University of Maritime and Inland Shipping, 5/7, Dvinskaya Street, 198035, Saint-Petersburg, Russia

Abstract. Recent achievements in high-performance computing significantly narrow the performance gap between single and multi-node computing, and open up opportunities for systems with remote shared memory. The combination of in-memory storage, remote direct memory access and remote calls requires rethinking how data organized, protected and queried in distributed systems. Reviewed models let us implement new interpretations of distributed algorithms allowing us to validate different approaches to avoid race conditions, decrease resource acquisition or synchronization time. In this paper, we describe the data model for mixed memory access with analysis of optimized data structures. We also provide the result of experiments, which contain a performance comparison of data structures, operating with different approaches, evaluate the limitations of these models, and show that the model does not always meet expectations. The purpose of this paper to assist developers in designing data structures that will help to achieve architectural benefits or improve the design of existing distributed system.

1 Introduction

Distributed computing is used in a wide variety of fields. This includes scientific research, technical developments such as facial recognition, control systems and autopilots [1-4]. In general, data analysis is finding more and more applications, and it is safe to say that it will not lose popularity in the near future. In fact, we are now experiencing a transition from the era of cloud computing, where applications and the speed of service deployment were the most important factors, to the era of data processing, including through the use of artificial intelligence algorithms. Many of them will consider this area as the main one for the modernization of business processes and the basic tool for making business decisions in the future. This means that each of these companies will need some kind of raw data processing — most likely through distributed clusters.

With the growing popularity of distributed computing, the volume of traffic exchanged between individual data center nodes increases. Usually most attention comes to the growth of traffic between the data center and end users, and it is really growing. But the increase in

* Corresponding author: apnyrkow@mail.ru

horizontal traffic within distributed systems far exceeds everything that users generate [5]. These kind of systems uses data structures which may be hard to implement using traditional message passing interfaces in a distributed memory environment. Recently developed, distributed data structure libraries are founded on remote direct memory access (RDMA). While RDMA was previously limited to high-performance computing environments with specialized Infiniband networks, RDMA is now available in cheap Ethernet networks using technologies such as RoCE [6, 7] or iWARP [8]. The main benefit of RDMA is one-sided operations, which permit a program to directly read and write the memory of a remote node without the involvement of the remote CPU.

The first section describes the problem — distributed system data processing using the network to access to shared memory. Background section provides basic information and different approaches to access data in a large scalable distributes systems. Section 3 provides describe our solution and show architecture of a system using provided approach. We perform sets of experiments in section 4. First, we perform benchmarks to gather the costs of the component operations that make up RDMA-based data structure implementations, then we evaluate expected latencies of data processing and compare with results on real system. The next section provides reviews of related works, which were studied and used to compare or improve our system and literature on different approaches to solving the problem that have been investigated. Conclusion and future work section are used to discuss the results of experiments and possible open problems for future research, such as high performance file systems, security and reliability of RDMA-based solutions.

2 Background

Distributed systems based on message passing model require to move data between nodes and processes to make it available to other nodes. In these sockets networks process always request network resource from the operating system (OS) through an API.

Shared memory model allows all the nodes and processes read or write at any address of the global shared memory. The data is shared between all the nodes. Direct Memory Access (DMA) is an ability to access host memory directly. Remote Direct Memory Access (RDMA) is the ability of accessing memory on a remote node without interrupting the processing of the CPU on it. With RDMA processes can directly exchange messages without further OS intervention. A message can be Read / Write operation or Send / Receive operation [9].

Currently, there are three technologies that support RDMA: InfiniBand, Ethernet RoCE and Ethernet iWARP. These technologies share a common user API, but have different physical and link layers. InfiniBand provides various technology or solution speeds ranging from 10 GB/s (SDR) up to 56 GB/s (FDR) per port, using copper and optical fiber connections [6, 10].

RDMA over Converged Ethernet (RoCE) RoCE is a standard for RDMA over Ethernet. RoCE provides true RDMA semantics for Ethernet as it does not require the complex and low performance TCP transport, which needed for iWARP. RoCE requires a low CPU overhead.

Message passing model, such as Remote Procedure Call (RPC), uses the data structures and architecture, which usually consists of two functions. First one is a handler, which will be executed on by the target thread. Second is the function, which will be called directly by the user on the initial thread and issue the RPC request. These requests can be put into a queue or even sent to another free node. Depending on the usage type the handler function will be called serially or simultaneously, the handler function may require local atomic operations or other mechanisms of synchronization. RDMA data structures require special

mechanisms to keep data consistency while parallel processing might invalidate data segments.

Two modes of communication are supported by RDMA. One-sided communication used to transfer data without the usage of remote CPU. Another one, two-sided use traditional send-receive model, which requires the remote CPU to be used to handle the requests. Read / Write one-sided operations deliver higher throughput, while Send / Receive two-sided operations offer more flexibility.

RDMA Read operation read a section of memory from the remote host. The caller specifies the remote virtual address as well as a local memory address to be copied to. Remote host not getting notified about RDMA read operations being conducted. For both RDMA read and write, the remote side isn't aware that this operation being done. API functions of our library GRead() and GWrite() are directly based on RDMA Read and Write operations with additional synchronization primitives, while GMalloc() and GFree() implemented through RPC.

The atomic fetch and add operation atomically increments the value at a specified virtual memory address. The previous value is returned to the caller function. The atomic compare and swap will atomically compare the value at a specified virtual memory address with a specified value and if they are equal, a value will be stored at the memory address. These operations used to implement synchronization counters.

3 System architecture

Our main goal is to use existing systems with RDMA efficiently, keeping them flexible enough to remain able to rescale. Target environment is enterprises using rack-scale systems, consisting of one or a few racks with up to 16 machines in total. In this section, we describe system architecture, in which we try to utilize the benefits of RDMA over previous architecture based on RPC (eRPC), while discussions about benefits are still ongoing [11-13]. Results of comparison and micro-benchmarks are presented in section 4.

Related works show how to use RDMA to improve the performance of distributed in-memory storage systems [14, 15]. RDMA is now available in cheap Ethernet networks [6], but still difficult to use, as it requires expert knowledge of the low-level protocols and APIs. However, RDMA is widely believed to have scalability issues, due to the amount of active protocol state that needs to be cached in the limited NIC (Network Interface Controllers) cache. We also revised latest works, which address this as a solved issue [11, 12, 16, 17]. At this point, we decided to adapt data structures, which appear to be the bottleneck in architectures actively using parallel data reads over writes. In the same time, writes are important to be reliable, strictly ordered and transactional. For the RDMA-based implementations, we will use solutions based on provided in the Berkeley Container Library (BCL) data structures. BCL is a cross-platform library of distributed data structures [18].

Described system architecture shown on Figure 1. Global memory consists of distributed pieces, which accessed directly using RDMA, but most reads stay inside local cache. Synchronization level keeps track over memory pointers, working in standard multi-threaded manner, using RPC for managing. We used an incremental counter variable to keep a local cache up to date with global memory.

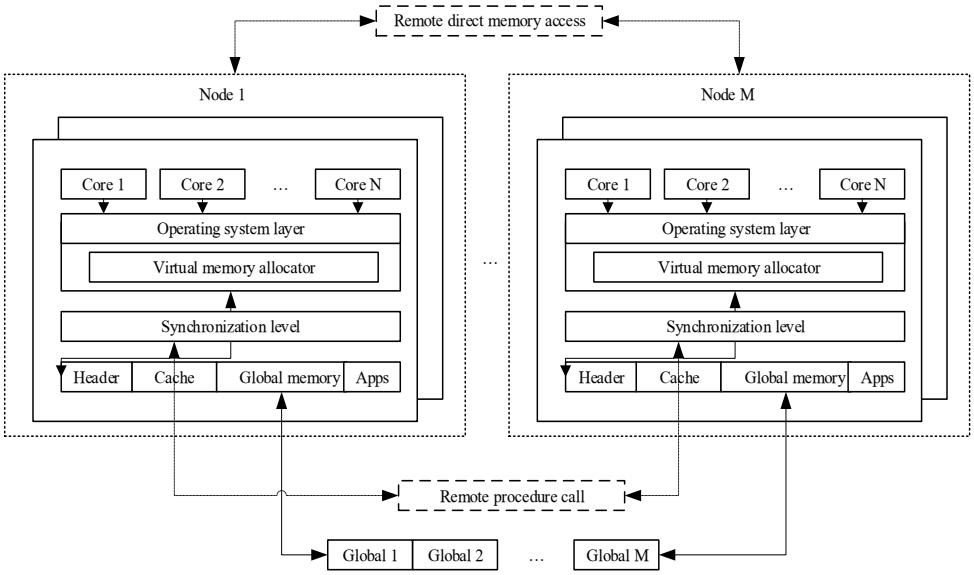

Fig. 1. System design.

During initialization, Synchronization level request and register nodes with available global memory, setting cash sizes and memory management headers. Implementation of global memory and cashing subsystem such this described in [19]. Memory writes only available through RPC, while reads provided through RDMA and RPC (with optimized data structures). Each single thread, which responsible for memory write operations inside the global memory of the current node, is registered by Synchronization level to provide memory for further GMalloc() / GFree() / GWrite() operations queue. RPCs are asynchronous to avoid stalling on network round trips. Thread concurrency is provided using an event loop. RPC servers register request handler functions, clients use these request types when issuing RPCs, and get continuation callbacks on RPC completion. Clients store RPC messages in queue. At higher network speeds, inter-thread communication is not cheap: it adds up to 400 ns to request latency [7], so local cache is only locked to update if the counter variable differs from its global value. Counters values are logged for debugging and further performance analysis.

Our usage pattern is mostly read-based, therefore the implemented cache must contain data for more than one request. The algorithm for reading and updating the cache is shown in Figure 2. Each process contains a vector of cached counter variable values. Each time a process thread requests a read operation GRead(), the value of the cached counter vector is compared to its original counterpart using an atomic operation. If the value has not changed, the thread gets a local copy of the existing cache. If there is no cache or the value has changed, the process locks the local cache and tries to acquire a shared lock on the global data segment. On acquiring a lock, the process copies (using an RDMA read operation) a portion of the global data into the local cache, atomically incrementing the global counter (using atomic fetch and add) and updating the local counter cache vector. The global shared lock on the segment is then released.

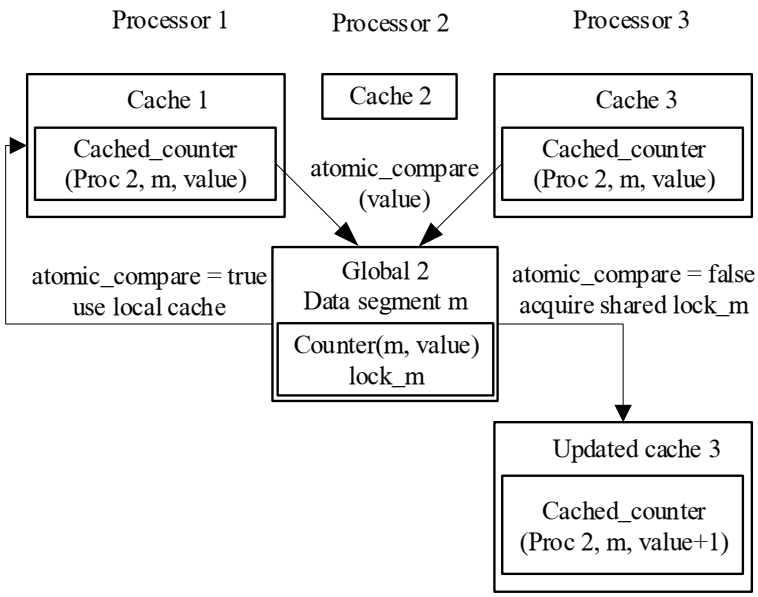

Fig. 2. Processors 1 and 3 read global data segment of Node 2.

Writing operations using RPC, which have the advantage of being more expressive than RDMA. Each node maintains Synchronization level threads, which are used to deliver messages to perform GMalloc() / GFree() and GWrite() operations. These messages are queued and after processing, a callback function is called to deliver the result.

Process of Synchronization level of the node on which the global memory segment is located processes messages from the distributed queue, exclusively locking this segment. Once an update is made, global counter vector variable increments and segments unlocks. Interaction between processes when message processing occurs during data copying is shown on Figure 3.

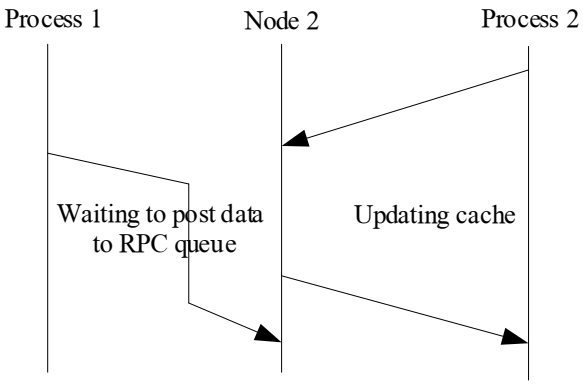

Fig. 3. Process 2 updating cache while Process 1 post message to Node 2 processing thread.

We used known algorithms [20] to keep track of the logical time of the process to order requests and prevent race conditions. The vector clock for a given process contains the logical time of each other process at the moment when request performed. Since the shared memory area is locked during update, there can be no race condition between remote memory accesses.

4 Results

In this section, we describe our experimental results. For our experiments, we used the following hardware and software:

- Servers 2 x Dell EMC PowerEdge R7525.
- Processors AMD EPYC 7252 3.10GHz, 8C/16T, 64M Cache.
- 256 GB Memory.
- Switch Dell PowerSwitch S4148U.
- Ethernet Adapter with RDMA 2 x QLogic FastLinQ QL41112 Dual Port 10GbE CNA SFP+.
- Software part consist of OS SUSE Linux Enterprise Server 15, we used GCC 10.1 as compiler and C++17, STL, BCL, BOOST as libraries/components.

RoCE relies heavily on DCB (Data Center Bridging) configuration, such as ETS (Enhanced Transmission Service) and PFC (Priority Flow Control), which might become a problem if network switches are not configured properly, while iWARP does not require any switch configuration. These options were used as recommended.

For benchmarking, we used distributed hash table, which use memory layered memory blocks for buckets. This key-value storage used to evaluate workloads imitated by single-object reads. Key-value lookups uses our implementation to look up random keys. Each bucket has a various number of slots for data, which had two levels of blocks as described in [21] and were counted as one segment. As for special cases, colliding items are kept in a linked list when the bucket capacity is exceeded. When the hash table is heavily loaded, linked list traversal with full value memory comparison are required to find the key. Minimal latency for basic operations shown in Table 1 and Table 2. Data structures operations can be characterized in terms of an analytical cost model of basic operations.

Table 1. Latency of operations with RDMA, single node.

Operation name	Latency (µs)
Write()	5.3
Read()	4.8
GWrite()	154.5
GRead()	6.3
GMalloc()	55.7
GFree()	65.9

Table 2. Latency of operations with RDMA, parallel access, 16 nodes.

Operation name	Latency (µs)
Write()	5.6
Read()	4.8
GWrite()	321.2
GRead()	7.8
GMalloc()	84.3
GFree()	68.7

The access pattern of the workloads is controlled via percentage of Read() / GRead(), data locality, sharing ratio. Each node contributes its free memory to the global space, and employs a least recently used cache, which is configured to use 30% of its own working set. For this benchmark, each node launches a process to independently generate mostly reads workload. For each system, we run the benchmark few times. The first run is used to warm up the cache, and results of other runs are averaged for the report. Figure 4 shows the results of accessing the filled data structure. Find / Insert / Remove latency was measured

on a real system, while EFind / EInsert / ERemove shows us corresponding expected results.

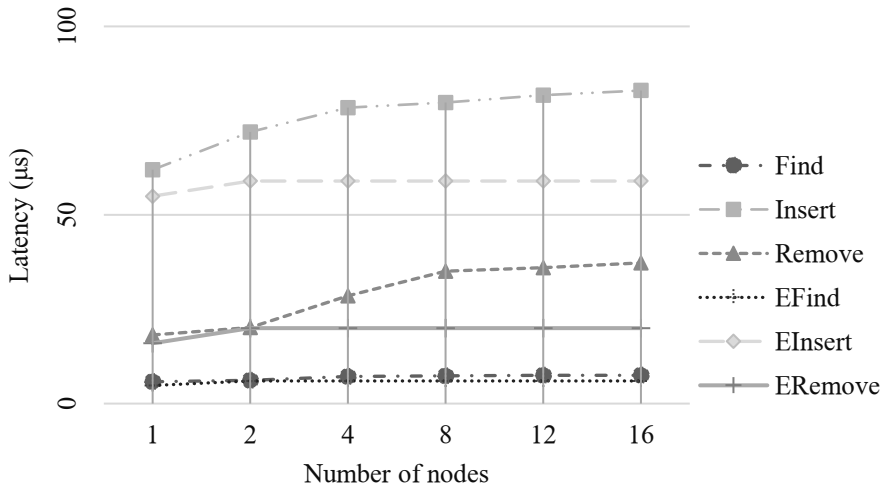

Fig. 4. Average latencies of data structure operations.

In each of the above benchmarks, each thread issue a single request of each type, process result and repeat operation. Above 70% of reads supposed to hit local cache. Hash table collisions calculated to be rarer, not more than two messages with the same hash.

5 Related work

There are a huge number of papers on the subject of using different approaches to work with various data structures using RPC in distributed and scalable systems [14, 18, 22, 23]. Some of them allow us to achieve rates close to those achieved by the latest RDMA-based solutions [11].

Solution based on RPC show how to use raw datagrams Sprite RPC [23] to perform retransmissions only at clients, similar solutions were proposed in Other RPC systems that use RDMA [14 – 17]. The majority of scalable, high performance systems use message passing model and some kind of cashing. Data locality leads to high-performance code, but makes it harder to work productively with the data. Several approaches address this problem by providing higher-level data structure abstractions and supporting a global view across multiple compute nodes [13]. Systems are emulating shared memory using RDMA [16, 19], which allow use remote pages as local memory.

There are many more works, such as RDMA-based transaction processing systems [14, 19, 24], distributed lock management [25], resource desegregation [16, 17]. In these systems RDMA part provides primitives like one-sided RDMA read/write and can be used with fast RPC implementation. Distributed memory provides a global memory model whose interface allows us to allocate / free / read / write data to different clusters of machines. This approach requires significant performance overhead, although it provides a convenient level of abstraction.

HERD [26] is a system designed to use the Unreliable Datagram transport, which allows a thread to use a queue to communicate with all the machines in the cluster.

Systems which use one-sided reads and two-sided RPC and Storm [15], which is transactional, more general-purpose and use RPC primitives. Fasst [17] provides key

insights about the choice of RDMA primitive for each phase of a two-phase commit protocol using both unreliable and reliable transports.

RDMA was widely adopted for performance benefits, while few have attempted to improve the security of such systems. Some recent works demonstrate a number of security threats in RDMA NICs [28].

6 Conclusion and future work

In this paper, we introduced the access model to shared data structures located in virtual global memory space. This model considers interactions between nodes and processes, taking into account the characteristics of the equipment used to implement such systems. The solution was designed to operate in a space with average data volume, wide variety of data sources with low update frequencies, but intensive read requests and requirements to perform queries with low latency.

From the results of the RDMA-based model, we conclude that network performance is a bottleneck for distributed memory applications based on processing large chunks of data. Operations seem to roughly match their associated performance models, with some increase in the real benchmark runtime. Our previous model, which used only RPC-based data access with the same usage pattern, showed lower find operation performance, while writes were almost twice as fast. In a more realistic use case, the data processing engine will have a huge impact on latency, but this requires further research.

We describe and benchmark the data model for mixed memory access with ordering between events in a distributed system. This model can be implemented in a data processing support system library that executes a program on a distributed system. Our model allows us to take a new look at distributed algorithms. In the future, we plan to try to add operations related to additional caching of large blocks to NVRAM. It is worth considering allocating a separate process that will distribute the write requests without re-acquiring global memory areas.

The data access model presented in this paper can be optimized to work with recently proposed high performance file systems such as Octopus. Octopus is an RDMA-enabled Distributed Persistent Memory File System Non-volatile memory (NVM) and RDMA, which provide very high performance in storage and network hardware [27].

Security of data processing using proposed RDMA model requires additional research. Secure RDMA is increasingly popular for low-latency communication in data centers, marking a major change in how we build distributed systems. The study of the security aspects of RDMA systems is an actual and a popular topic for research. We have reviewed some recent publications [28, 29] and plan to use them to improve the proposed model in the future. Unfortunately, as we pursue significant system re-designs inspired by new technology, we have not given equal thought to the consequences for system security. In our future research, we plan to investigate security issues introduced to systems by implementing RDMA models and challenges in building secure data processing systems. These challenges include changes in RDMA reliability guarantees and unauthorized access to data.

Failures of Nodes: Another possible topic for research may be the topic of node fault tolerance and failure recovery without violating low-level transactionality.

References

1. A. Stanciu, *21st International Conference on Control Systems and Computer Science (CSCS) IEEE* (2017) http://dx.doi.org/10.1109/cscs.2017.102

2. A.P. Nyrkov, A.A. Zhilenkov, S.S. Sokolov, et al., Autom Remote Control **79**, 195 (2018) https://doi.org/10.1134/S0005117918010174
3. A.P. Nyrkov, A.A. Zhilenkov, V.V. Korotkov, S.S. Sokolov, S.G. Cherniy, Journal of Physics: Conference Series **803(1)**, 012108 (2017) https://doi.org/10.1088/1742-6596/803/1/012108
4. S. Sokolov, A. Zhilenkov, A. Nyrkov, S. Chernyi, Computational Intelligence in Data Mining. Advances in Intelligent Systems and Computing **556**, 421-427 (2017) https://doi.org/10.1007/978-981-10-3874-7_39
5. F. Li, L. Zhao, Springer International Publishing (2019) http://dx.doi.org/10.1007/978-3-319-32903-1_325-1
6. G. Chen, Y. Lu, B. Li, K. Tan, Y. Xiong, P. Cheng, et al., IEEE/ACM Transactions on Networking. Institute of Electrical and Electronics Engineers (IEEE) **27(6)**, 2308–23 (2019) http://dx.doi.org/10.1109/tnet.2019.2948917
7. A. Kalia, M. Kaminsky, D.G. Andersen, *Proceedings of the 2014 ACM conference on SIGCOMM* (2014) http://dx.doi.org/10.1145/2619239.2626299
8. R. Recio, B. Metzler, P. Culley, J. Hilland, D. Garcia, *A Remote Direct Memory Access Protocol Specification* (RFC Editor, 2007) http://dx.doi.org/10.17487/rfc5040
9. *17th IEEE Symposium on High Performance Interconnects* (2009) http://dx.doi.org/10.1109/hoti.2009.31
10. R. Mittal, A. Shpiner, A. Panda, E. Zahavi, A. Krishnamurthy, S. Ratnasamy, et al., *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication ACM* (2018) http://dx.doi.org/10.1145/3230543.3230557
11. A. Kalia, M. Kaminsky, D. Andersen, *Datacen-ter RPCs can be General and Fast InNSDI* (2019) https://www.usenix.org/system/files/nsdi19-ousterhout.pdf
12. J. Nelson, R. Palmieri, 2020 *International Symposium on Reliable Distributed Systems (SRDS) IEEE* (2020) http://dx.doi.org/10.1109/srds51746.2020.00036
13. A. Kalia, M. Kaminsky, D.G. Andersen, *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (2016) https://www.usenix.org/system/files/conference/osdi16/osdi16-kalia.pdf
14. X. Wei, J. Shi, Y. Chen, R. Chen, H. Chen, *Proceedings of the 25th Symposium on Operating Systems Principles ACM* (2015) http://dx.doi.org/10.1145/2815400.2815419
15. S. Novakovic, Y. Shan, A. Kolli, M. Cui, Y. Zhang, H. Eran, et al., *Proceedings of the 12th ACM International Conference on Systems and Storage* (2019) http://dx.doi.org/10.1145/3319647.3325827
16. A. Dragojević, D. Narayanan, M. Castro, O. Hodson, 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14) (2014) https://www.usenix.org/system/files/conference/nsdi14/nsdi14-paper-dragojevic.pdf
17. A. Kalia, M. Kaminsky, D.G. Andersen, *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)* (2016) https://www.usenix.org/system/files/conference/osdi16/osdi16-kalia.pdf
18. B. Brock, A. Buluç, K. Yelick, *BCL Proceedings of the 48th International Conference on Parallel Processing ACM* (2019) http://dx.doi.org/10.1145/3337821.3337912
19. Q. Cai, W. Guo, H. Zhang, D. Agrawal, G. Chen, B.C. Ooi, et al., *Proceedings of the VLDB Endowment. VLDB Endowment* **11(11)**, 1604–17 (2018) http://dx.doi.org/10.14778/3236187.3236209
20. L. Lamport, *Association for Computing Machinery* (2019)

http://dx.doi.org/10.1145/3335772.3335934

21. A.P. Nyrkov, K.A. Ianiushkin, A.A. Nyrkov, Y.N. Romanova, V.D. Gaskarov, *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)* (2020) http://dx.doi.org/10.1109/eiconrus49466.2020.9039354
22. M. Stumm, S. Zhou, Institute of Electrical and Electronics Engineers (IEEE) **23(5)**, 54–64 (1990) http://dx.doi.org/10.1109/2.53355
23. B.B.Welch, *The Sprite Remote Procedure Call System. Defense Technical Information Center* (1986) http://dx.doi.org/10.21236/ada619316
24. J. Jose, H. Subramoni, M. Luo, M. Zhang, J. Huang, M. Wasi-ur-Rahman, et al., *2011 International Conference on Parallel Processing* (2011) http://dx.doi.org/10.1109/icpp.2011.37
25. S. Narravula, A. Marnidala, A. Vishnu, K. Vaidyanathan, D.K. Panda, *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)* (2007) http://dx.doi.org/10.1109/ccgrid.2007.58
26. A. Kalia, M. Kaminsky, D.G. Andersen, *2016 USENIX Annual Technical Conference (USENIX ATC 16)* (2016) https://www.usenix.org/system/files/conference/atc16/atc16_paper-kalia.pdf
27. J. Yang, J. Izraelevitz, S. Swanson, *17th USENIX Conference on File and Storage Technologies (FAST 19)* (2019) https://www.usenix.org/system/files/conference/atc17/atc17-lu.pdf
28. A.K. Simpson, A. Szekeres, J. Nelson, I. Zhang, *12th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 20)* (2020) https://www.usenix.org/system/files/hotcloud20_paper_simpson.pdf
29. S. Bailey, T. Talpey, *The Architecture of Direct Data Placement (DDP) and Remote Direct Memory Access (RDMA) on Internet Protocols* (RFC Editor, 2005) http://dx.doi.org/10.17487/rfc4296