Procedia Computer Science

Volume 66, 2015, Pages 515–524

YSC 2015. 4th International Young Scientists Conference on Computational Science

Platform for parallel processing of intense experimental data flow on remote supercomputers

Vladislav Shchapov[1,2] and Grigoriy Masich[2,1] and Alexey Masich[2]
[1]*Perm National Research Polytechnic University, Perm, Russia*
[2]*Institute of Continuous Media Mechanics UB RAS, Perm, Russia*
shchapov@icmm.ru, masich@icmm.ru, mag@icmm.ru

Abstract
Modern experimental facilities generate large amounts of data that should be processed, saved to hard disk and presented to the user as fast as possible. However, in-situ data analysis requires technical resources which are often not available. The existence of accessible high-speed networks allows to forward data processing and storage to a remote supercomputer centers and datacenters. These capabilities can be realized through the development of architectural solutions for effective data transmission trough a long-distance high-speed networks, data input/output and data distribution over computers and data storage systems. In this paper, we describe the results of investigations into the development of a software platform for parallel processing of intense experimental data-streams on ICMM UB RAS (Perm) and IMM UB RAS (Yekaterinburg) supercomputers, interconnected by a high-speed network. The reported studies was partially supported by RFBR, research project No. 14-07-96001-r_ural_a and by Program of UD RAS, project No 15-7-1-25.

Keywords: Long fat network, Supercomputer, Parallel data processing, Middleware, Distributed system

1 Introduction

Recently, well-known projects in the field of e-Science have touched upon processing of larger and larger data sets received from remote experimental setups (e.g., the CERN LHC in high energy physics and the Dutch LOFAR project in astronomy). Initially, almost all distributed computing were based on widespread among users Internet TCP/IP networks. The current stage of distributed computing technologies development is focused on using national and regional research and educational optical networks (e.g., Geant2 in Europe, Internet2 in the United States and Initiative GIGA UrB RAS in Russia). The trend of increasing network bandwidth and reducing delay in data transmission (Rumble, Ongaro, & Stutsman, 2011) made it possible to build distributed systems where the data sources and the processing supercomputers can be located in different locations.

Selection and peer-review under responsibility of the Scientific Programme Committee of YSC 2015

doi:10.1016/j.procs.2015.11.058

In this context, current research and development efforts are aimed at solving two related problems: (1) effective use of high-speed (10-100Gbps) and long (thousands kilometers) telecom links; (2) methods of organization a high-speed input/output of data into a supercomputer (Yildirim, Arslan, Kim, & Kosar, 2015). For example, Pittsburgh Supercomputer Center projects (Advanced Networking, Three Rivers Optical Exchange, Web10G) aimed at increasing data storages access speed and tuning TCP protocol (Pittsburgh Supercomputing Center, 2015). Among the specialized protocol for data transmission over high-speed long fat networks can be noted UDT protocol (Gu & Grossman, 2007).

This paper describes platform for experimental data-intensive flow parallel processing, created in UB RAS. Platform is based on high-speed optical DWDM-backbone interconnecting the Supercomputing Center of IMM UB RAS (Yekaterinburg) and data center ICMM UB RAS (Perm). Infrastructure and middleware of this platform implemented as a classical model of interaction between supercomputers and experimental setups, as well as developed model of direct dataflow input into computing nodes of supercomputers for parallel processing. An example of the practical use of this platform is the project “Distributed PIV”, the essence of which lies in the processing of the flow measurements, obtained by PIV-method, on a remote supercomputer in real time to provide feedback and control of the experiment (Stepanov, Masich, & Masich, 2009).

2 Components of the distributed computing environment of UB RAS

Distributed computing environment of UB RAS is based in two data centers, which are located in ICMM (Perm), PSC (Perm) and IMM (Yekaterinburg). Supercomputer “URAN” with peak performance of 225.85 teraflops and three servers of Supermicro dCache distributed data storage system (DSS) are located in the IMM data center. Computing cluster “Triton” with peak performance of 4.5 teraflops and one server of Supermicro dCache distributed data storage are located in ICMM data center.

Communication environment of the distributed system, as shown in Figure 1, is formed by Ethernet switches, connected to the regional R&E DWDM backbone called “GIGA URAL” through direct Perm-Yekaterinburg 30Gbps link (Masich & Masich, 2009). The 40Gbps “ICMM-PSC” CWDM metro network in Perm interconnects computing resources of ICMM and DWDM backbone. For greater flexibility of studies, ECI AS9215 L2-switches and Extreme Summit X670-48x L3-switches are placed in IMM, ICMM and PSC and forming together with Ethernet ports of DWDM multiplexers guaranteed and not guaranteed 1-10 Gbps links.

3 Architecture of the data network platform

“TRITON” computing cluster consists of three HP BladeSystem c7000 enclosures. Each enclosure includes 16 HP Proliant BL 460c servers. There are two networks used for data transfer between computing nodes of the cluster. The first one is main MPI interconnect which operates on InfiniBand 4xDDR 20 Gbps. The second one is an additional 1 Gbps Ethernet network, which is used for the task flow control and distributed between computing nodes network file system. Each enclosure is equipped with a built-in 1 Gbps Ethernet switch. These built-in switches are connected to an external AS9215 switch by four aggregated 1 Gbps links using Link Aggregation Control Protocol (LACP). The AS9215 switch is connected to the backbone by 10 Gbps link.

“URAN” supercomputer has similar architecture. The backbone network provide Ethernet communication between interconnect of the “URAN” supercomputer and dCache servers located in

IMM and interconnect of the “TRITON” computing cluster and dCache server located in ICMM by two 10Gbps links. “TRITON” and dCache servers operates in the same L2-network segment, but “URAN” interconnect has its own. IP-traffic between segments is routed through master node of the “URAN” with a 1Gbps limited throughput.

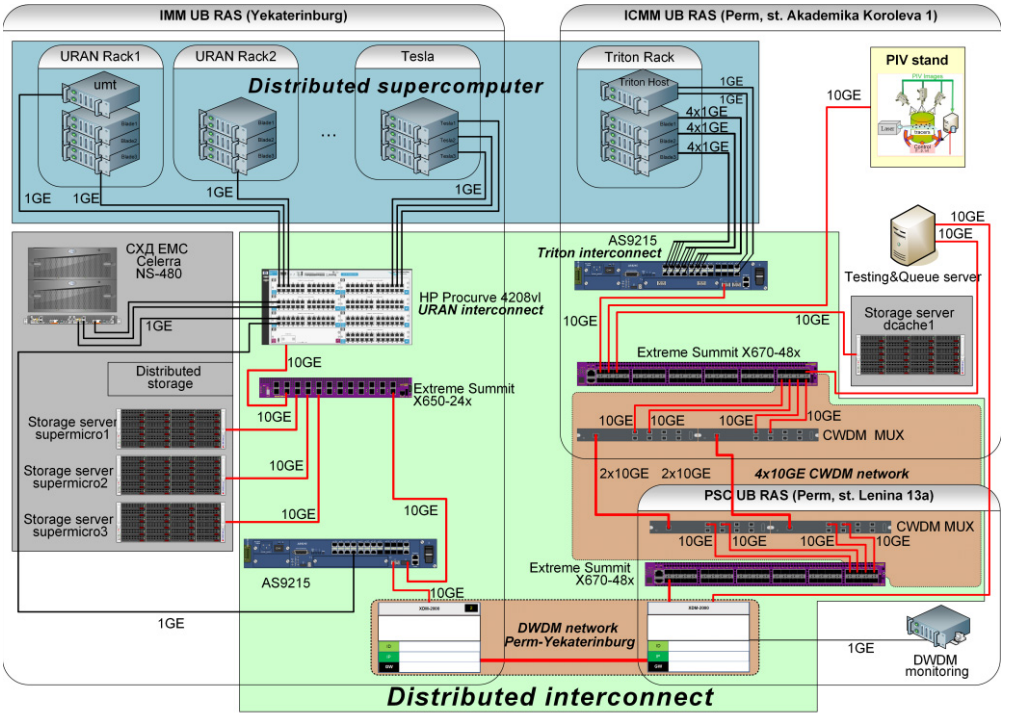

Figure 1: The architecture of the distributed computing environment of UB RAS

4 Models of parallel processing of data flows

Usually, supercomputer’s storage system is being used as a source of big data for future processing. In this "classic mode" processing occurs in three phases, as shown in Figure 2:

1. Data download into the supercomputer’s local storage.
2. Data processing on a supercomputer.
3. Upload data processing results to the supercomputer’s local storage.

Data upload/download to/from the storage (steps 1 and 3) and data processing (step 2) associated with intermediate read/write operations on storage.

The most common way to exchange data with storage is to use file transfer protocols such as FTP/GridFTP and SCP. Another direction of this approach is direct access to the data storage by using network file system protocols, such as CIFS and NFS/pNFS. This method allows connecting supercomputer’s storage to the data source as a remote file system and performing record data without using any specialized software. However, these solutions are not good enough for a long fat network (Hildebrand, Eshel, Haskin, Kovatch, & Andr, 2008), (Weikuan Yu, 2008). At the same time, performance degradation of geographically distributed applications occurs due to insufficient efficiency of TCP protocol in long fat networks.

The second approach, which was developed in our laboratory, assumes that the elements of the dataflow are transmitted directly to the memory of computing nodes ("Memory to Memory" method, as shown in Figure 2). In this case, the necessity for ensuring an effective data transmission to the computing nodes of the supercomputer and finding solution to the problem of dataflow elements distribution between computing nodes, comes to the foreground. The prototype of the proposed data processing model is data queue concept. It allows to implement parallelism on the data processing and transmission level and to perform distribution of dataflow elements between computing nodes by a separate component of the system – queue manager.

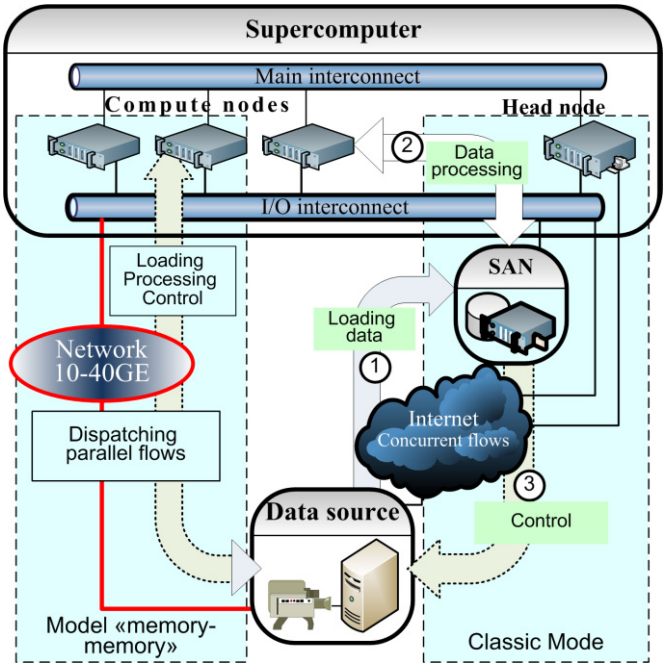

Figure 2: Existing (classic mode) and developed (memory to memory) architectural design of the data processing on the supercomputer

This approach has the following distinctive features:

- avoid using of interim storage;
- direct data input into computing nodes;
- parallelism of data transfer connections between end systems.

Parallelism of connections fixes above-mentioned problem of efficient use of reliable transport protocols in long fat network. Thus, it is possible to use standard TCP and new protocols e.g. UDT, as the transport protocol (OSI RM L4).

5 "Memory-to-Memory" dataflow model

There are many tasks requiring dataflow processing, including dataflow processing of PIV experiment. The raw dataflow can be represented as a sequence of individual messages (measurements), which can be processed by applied algorithms independently. So, we proposed a dataflow model as shown in Figure 3 (Shchapov, Masich, & Masich, A model of stream processing of experimental data in distributed systems, 2012):

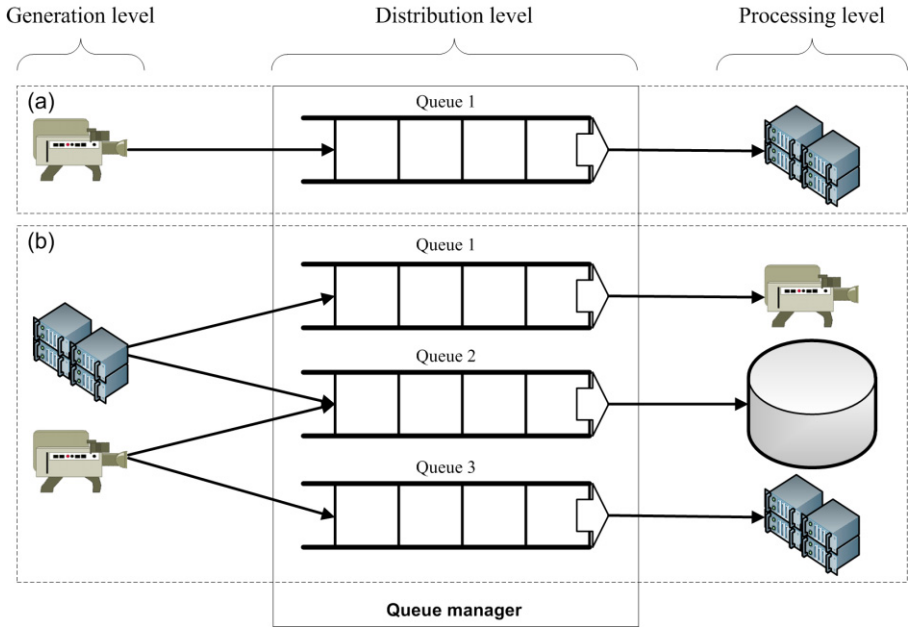

Figure 3: Dataflow processing model

- there are three consecutive stages of dataflow processing (levels):
    1. Source data generation;
    2. Source data distribution by handlers;
    3. Data processing by applied algorithms.
- queue manager represents dataflow from the source as the unified queue of messages;
- messages are distributed between computing nodes by their request to the queue manager in the FIFO (First In, First Out) order.

There are some features and flexibilities of the proposed model. Dataflow is always one-sided. Data comes from generation level through distribution level to processing level. In the general case, same device or processing application can be simultaneously placed on the generation level as well as on the processing. For example, in terms of raw data, processing application belongs to processing level, but in terms of transmitting of processed data application belongs to generation level.

When new data appears on the generation level then these data have to be processed. It may be an experimental setup that generates experimental data or results of calculation algorithm, which must be saved.

Processing level is responsible for the raw data calculation using applied complicated algorithms, storing data in high-performance storages or in a large number of local hard drive disks and so on.

Distribution level performs data transfer from generation level into the one or several queues of the queue manager and performs data transfer from queues of the queue manager to the applications of processing level in response to their queries. Ability of placing data into multiple queues simultaneously allows to solve the problem of saving the transmitted data, while one queue is used by calculating applications another one is used by application for saving data on storages, as shown in Figure 3 (b).

In this model, the queue manager must be placed as close to the data source as possible (10-1000m) to eliminate the problems of transport protocols. The efficiency of data transfer between

manager and processing level, which is a supercomputer with a number of nodes, is achieved in the following ways.

1. Parallelism in data transfer (Figure 4) for better long fat network bandwidth utilization.
2. Automatic load balancing of computing nodes becomes possible by the dataflow element distribution algorithm.
3. Tuning operating system's network stack for optimum performance.

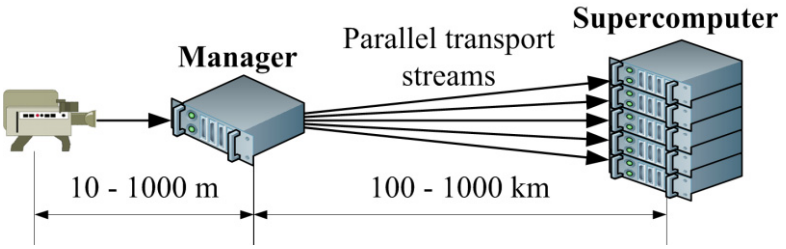

Figure 4: Data transfer parallelism in high-speed long fat networks

6 "Memory-to-Memory" software infrastructure model

Analysis of existing queuing technologies in distributed systems (ZeroMQ, AMQP protocol and it implementation RabbitMQ) shows that they are not satisfies the requirements in full (performance, ease of use, third-party transport protocols support).

ZeroMQ library does not provide control of system resources, used by the application (basically RAM), and does not provide access to the system socket for the network stack parameters control and does not support third-party transport protocols (e.g., UDT).

Ready to use queue managers, such as RabbitMQ, does not allow using third-party transport protocols for the investigation of their impact on the system capacity. Also, existing solutions system requirements exceeds available system resources in the beginning of the study. Thus, data distribution was performed by the PIV experiment setup control machine or by the server SUN Fire X2100M2 (Dual-Core AMD Opteron (tm) Processor 1214, 2.2 GHz, 2 GB RAM), where dataflow rate was less than 800 Mbps.

So we decided to develop our own data transfer protocol (SciMP) and middleware (SciMQ) for data transmission between levels of the system.

6.1 SciMP protocol

The designed SciMP protocol is an application layer protocol (Figure 5) and it works by request-response scheme (Shchapov & Masich, 2012). Any reliable transport level protocol can be used by SciMP. Current version supports TCP and UDT protocols. SciMP designed for transfer of named blocks of binary data. One data packet, as shown in Figure 6, can contain up to 65,535 blocks up to 4 Gbyte each, without guarantee of proper block order preservation. Names of blocks are 32-bit integers.

The reason of our own application layer protocol development is the need for operation at more than 10Gbps data rates. Under these conditions, the time spent on the analysis of complicated protocols formats, is a substantial and can lead to increased system requirements.

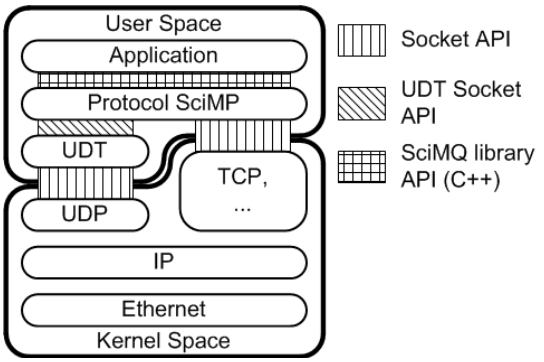

Figure 5: Position of the SciMP protocol in stack of network protocols

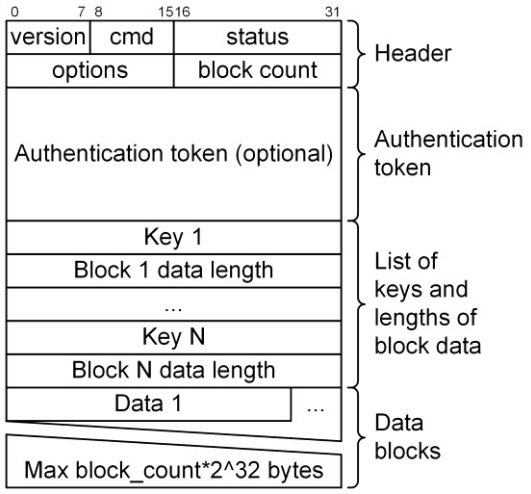

Figure 6: The SciMP protocol packet format

Proposed protocol is focused on the possibility of a single-pass parsing data packing format. It allows to reduce time for service header field's manipulation and to reduce the number of operations of memory allocation. Also, it allows avoiding data copy between memory buffers in a user space. Flat structure of protocol format, where the large blocks of data are sequentially arranged, allows using Zero-Copy technology for the communication between software application and operating system network subsystem.

6.2 Middleware

The proposed model of dataflow processing in distributed systems is implemented as the SciMQ middleware (Shchapov V. A., Programmnaja arhitektura sistemy peredachi intensivnogo potoka dannyh v raspredelennyh sistemah, 2013). Interaction of the middleware components logic scheme is shown in Figure 7.

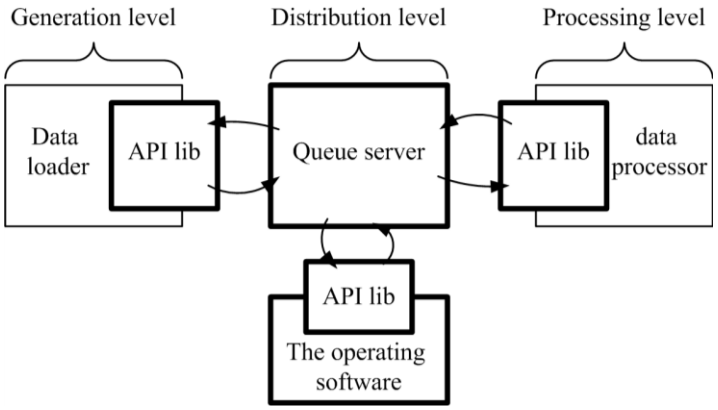

Figure 7: The logic scheme of the middleware components interaction

The main part of middleware is distribution level software called queue server, which is communicated with generation and processing levels applications (end systems). Queue server is controlled by WEB and Command Line interfaces. All components of the software complex are written in C++ programming language, using the Boost libraries.

End-system applications interact with queue server using SciMP protocol. These applications can independently implement SciMP protocol or use C ++ API client libraries to communicate with the queue server. API library hides from end-user SciMP protocol implementation and network operations, thereby allows focusing on the development of a complicated computing algorithm, rather than the implementation of the SciMP protocol.

The queue server is the core of software complex and implements basic functionality of the whole middleware. Queue server's main task is to manage messages in the queues, including their temporary storing, distribution upon requests and reception/transmission from/to external systems.

The operating software is designed to perform administration and testing software. It allows also performing operations on the creation, deletion and modification of queues and on generation, reception or redirection dataflow between queues, that allow to do load testing of the software complex for optimization and tuning network stack of the server that is used for software components.

7 Measurements

7.1 Testing TCP congestion control strategies

TCP congestion control strategies efficiency research was made by using bash-scripts and Iperf application. Tests were performed using 1Gbps dedicated and 1Gbps common network links. In case of common link, up to 100 Mbps of Internet traffic was transmitting at the same time. Testing results are shown in Figure 8 and Figure 9. The graphs show that the presence of even a small amount (about 10%) of third-party traffic substantially reduces the effectiveness of many TCP congestion control algorithms. However, most of the algorithms reach the aggregate data transfer rate close to the maximum link bandwidth when the number of parallel streams in the range of from 4 to 16. Therefore, it is reasonable to improve data transmission efficiency for the infrastructure with great outgoing traffic by selecting optimal TCP congestion control algorithm and tuning buffers size, providing TCP window size parameter not less than BDP (Bandwidth Delay Product). The effect of TCP Slow Start was also confirmed, so that the dynamics enhancing of transmission speed restrains the input of intensive dataflow to computing nodes.

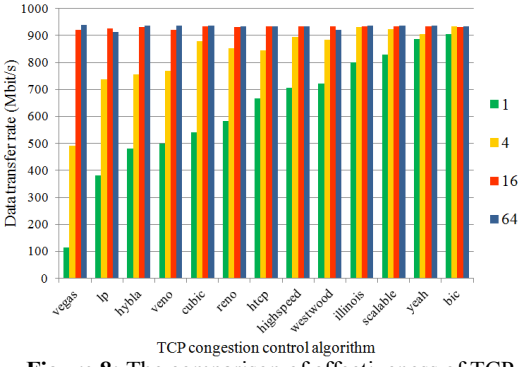

Figure 8: The comparison of effectiveness of TCP congestion control algorithms on the Perm - Yekaterinburg 1 Gbps common channel with Internet traffic (0,1 Gbps) and the number of parallel flows (1, 4, 16, 64)

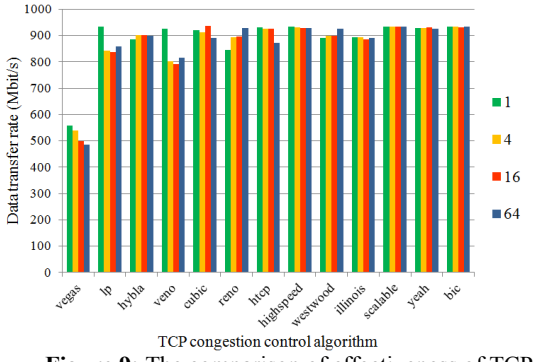

Figure 9: The comparison of effectiveness of TCP congestion control algorithms on the dedicated Perm - Yekaterinburg 1 Gbps dedicated channel with the number of parallel flows (1, 4, 16, 64)

Measurements made using "TRITON" supercomputer, showed that the optimal number of parallel TCP connections in the range of from 8 to 10 for each 1 Gbps physical channel in the LACP, which is

an bottleneck of supercomputer's network. Also, it is important to monitor the uniformity of the traffic distribution between physical channels in the LACP. Using L2-balancing implemented in ECI, in case of consecutive MAC addresses, gives more uniformity. In this case, the total bandwidth used by LACP depends on the active at the moment computing nodes location. It is clear, that L2-balancing effectively works only when all computing nodes from one aggregation is being actively used. It should be noted, that the available in Extreme switches L3/L4-balancing feature is able to solve this problem, because the outgoing TCP connections ports are chosen randomly, that provides a uniform channel loading and increases network bandwidth utilization.

7.2 Performance testing of the developed software

The performance testing of the developed software was performed using two HP ProLiant DL360p Gen8 servers (2x Intel Xeon CPU E5-2660, 2.20 GHz, 16 threads; RAM 128 Gb; operating system CentOS 6.5), interconnected by 10 Gbps and 10 meters length dedicated link.

Figure 10 shows the dependencies of the developed SciMQ and third-party RabbitMQ queue server's performance on the message size and the number of parallel requests to the server. The frame_max RabbitMQ configuration parameter was set to the default value. The graph shows that in case of message sizes 2-22 Mbyte, network performance is limited by the available bandwidth for SciMQ software. Comparing results of SciMQ and RabbitMQ testing we can say that SciMQ allows to process messages above 1 Mbyte 3-4 times faster than RabbitMQ.

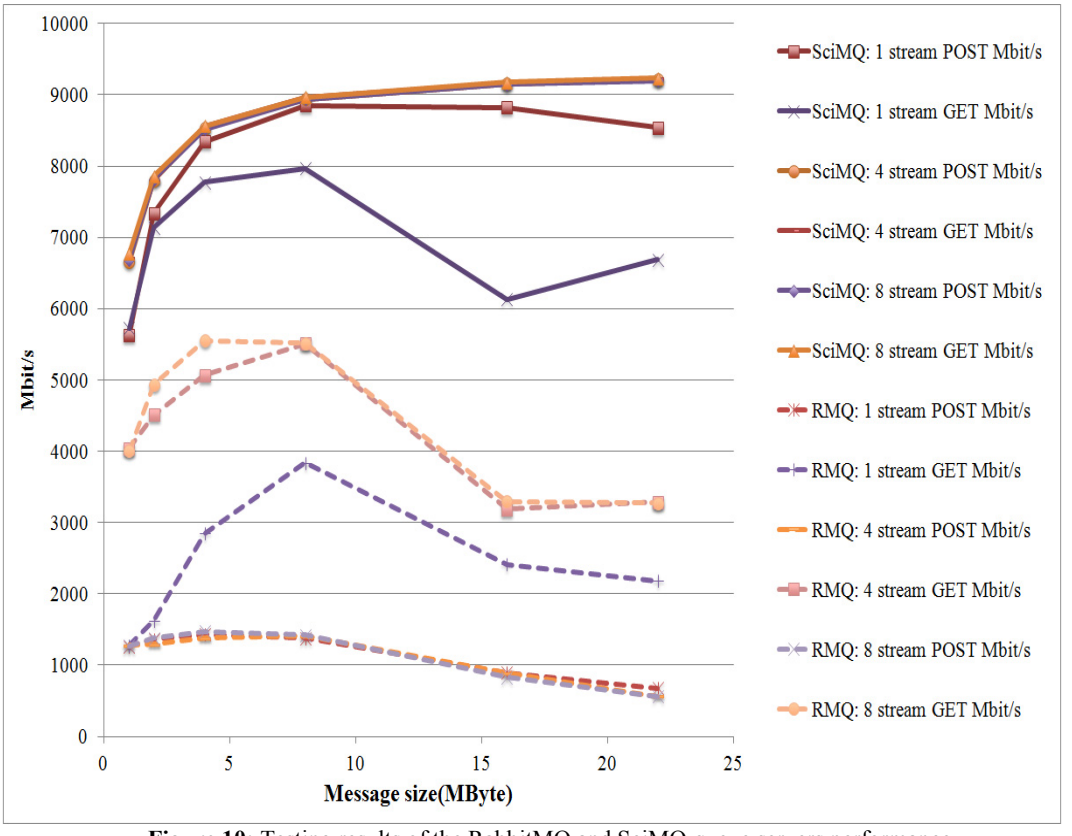

Figure 10: Testing results of the RabbitMQ and SciMQ queue servers performance

8 Conclusion

Proposed solution provides fundamentally new instrument for unique physical research in research labs and industry. It becomes possible to perform complicated experiments with intensive dataflow online processing. High-speed network links allows interconnecting the place of measurement with remote supercomputers, eliminating local computational resources upgrade costs. The application area of proposed solution is a new generation of distributed measuring systems, including high-tech measuring equipment in situ and high-performance computing facilities at the supercomputer centers.

References

Gu, Y., & Grossman, R. L. (2007). UDT: UDP-based data transfer for high-speed wide area networks //. *Computer Networks , 51* (7), 1777-1799.

Hildebrand, D., Eshel, M., Haskin, R., Kovatch, P., & Andr, P. (2008). Deploying pNFS across the WAN: First Steps in HPC Grid Computing. *in Proceedings of the 9th LCI International Conference on High-Performance Clustered Computing.*

Masich, G. F., & Masich, A. G. (2009). Ot «Iniciativy GIGA UrB RAS» k Kiberinfrastrukture UrO RAN. *Vestnik Permskogo nauchnogo centra UrO RAN* (4), 41-56.

Pittsburgh Supercomputing Center. (2015). *Advanced Networking*. Retrieved 2015 йил 27-07 from http://www.psc.edu/index.php/research-programs/advanced-networking

Rumble, S. M., Ongaro, D., & Stutsman, R. (2011). It's time for low latency. *Proceedings of the 13th USENIX conference on Hot topics in operating systems. HotOS'13* (pp. 11-11). Berkeley, CA, USA: USENIX Association.

Shchapov, V. A. (2013). Programmnaja arhitektura sistemy peredachi intensivnogo potoka dannyh v raspredelennyh sistemah. *Parallel'nye vychislitel'nye tehnologii (PaVT'2013): trudy mezhdunarodnoj nauchnoj konferencii (g. Cheljabinsk, 1–5 aprelja 2013 g.)* (pp. 566–576). Cheljabinsk: Izdatel'skij centr JuUrGU.

Shchapov, V. A., & Masich, A. G. (2012). Protocol of High Speed Data Transfer from Particle Image Velocimetry System to Supercomputer. *Proc. of The 7th International Forum on Strategic Technology (IFOST 2012) September 18-21. 2*, pp. 653-657. Tomsk: Tomsk Polytechnic University.

Shchapov, V. A., Masich, A. G., & Masich, G. F. (2012). A model of stream processing of experimental data in distributed systems. *Vychisl. Metody Programm , 13*, 139-145.

Stepanov, R. A., Masich, A. G., & Masich, G. F. (2009). Iniciativnyj proekt «Raspredelennyj PIV». *Nauchnyj servis v seti Internet: masshtabiruemost', parallel'nost', jeffektivnost': trudy Vserossijskoj superkomp'juternoj konferencii* (pp. 360-363). M.: Izd-vo MGU.

Weikuan Yu, R. N. (2008). Performance of RDMA-capable storage protocols on wide-area network. *Petascale Data Storage Workshop, 2008. PDSW '08.*, (pp. 1-5).

Yildirim, E., Arslan, E., Kim, J., & Kosar, T. (2015). Application-Level Optimization of Big Data Transfers Through Pipelining, Parallelism and Concurrency. *Cloud Computing, IEEE Transactions on , PP* (99).