Load-balancing Algorithm Supporting Real Time Mode in Distributed System

V. Kryukov, C. Shakhgeldyan, V.Mayorov
Vladivostok State University
41, Gogolya, Vladivostok, Russia 690600
E-mail: kryukov@vvsu.ru

Abstract

Heterogeneous workstations connected by local area networks are an attractive solution to design data acquisition and processing systems intended for analysis of the data of physical experiments. One of the most important problems arising during of the designing such systems is support of real time mode. In our work we will consider possibility of the use of several computers connected by local networks for support of the soft real time mode to acquire, process and accumulation of the data without information loss. Load balancing algorithm was developed for maintenance of system operation in real time mode. We discuss in the paper the following problems: detecting when the data needs to be repartitioned to provide real-time; technique of the choice of the most suitable workstation for load distribution; the mechanism to perform the repartitioning, and the results of the algorithm operation.

1 Introduction

Heterogeneous workstations connected by local area networks are an attractive solution to design data acquisition and processing systems intended for analysis of the data of physical experiments. The computational power of workstations, and bandwidth of networks are increasing rapidly [1]. As a result, the use of distributed model allows designing high-performance data acquisition and processing systems where data are processed by complicated long-duration procedures.

One of the most important problems arising during the designing such systems is support of real time mode. In our work we will consider the possibility of the use of several computers connected by local networks for support of the soft real time mode to acquire, process and accumulation of the data without information loss.

Load-balancing algorithms should be applied to use several networked computers with benefits. The most of load balancing algorithms are aimed to decrease calculating time [2-10]. In accordance with these algorithms workloads are redistributed if there are idle workstations. In our case we should not redistribute workloads if some nodes are idle. The main goal of the presented approach is to schedule workloads *to provide real-time mode*, i.e. to process without data loss.

In the second part of the paper we present the architecture of the Distributed Data Acquisition and Processing System (DDAPS) for which the load-balancing algorithm was implemented. The third part of the paper is devoted to the load-balancing algorithm. We discuss when the data needs to be repartitioned to provide real-time; the technique of the choice of the most suitable workstation; the mechanism to perform the repartitioning. The results of the algorithm operation are discussed in the fourth part of the paper.

2 Architecture of the system

DDAPS is intended for acquisition, processing, visualization, and management of physical experimental data. DDAPS is used in local area networks Fast Ethernet with network computers under Windows/Solaris. MS SQL Server DBMS is used to store DDAPS’s data (Fig.1). DDAPS was developed on the basis of CORBA technology, which provides object-oriented model of designing of the distributed systems. DDAPS represents a set of the CORBA-objects operating in the heterogeneous environment (Fig.2).

CORBA-objects of DDAPS belong to the following types: the control objects, processing objects, generating objects, objects of accumulation, and objects of visualization.

Three control objects are implemented in the current version of the system. The first object TaskManager (Fig.2) provides the system’s control and implements the load-balancing algorithm. The second object provides interaction with DDAPS’s database. The gathering statistics objects accumulate the information about CPU’s usage and belong to control objects of DDAPS also.

The class library providing coding new user’s objects of different types and theirs embedding into the DDAPS without change of the system is developed. DDAPS’s user has to create an experiment and to start it to operate in DDAPS. An experiment includes the data sources (objects of data acquisition, simulation of a signal, or read of signal from database), various processing methods, accumulation and visualization.

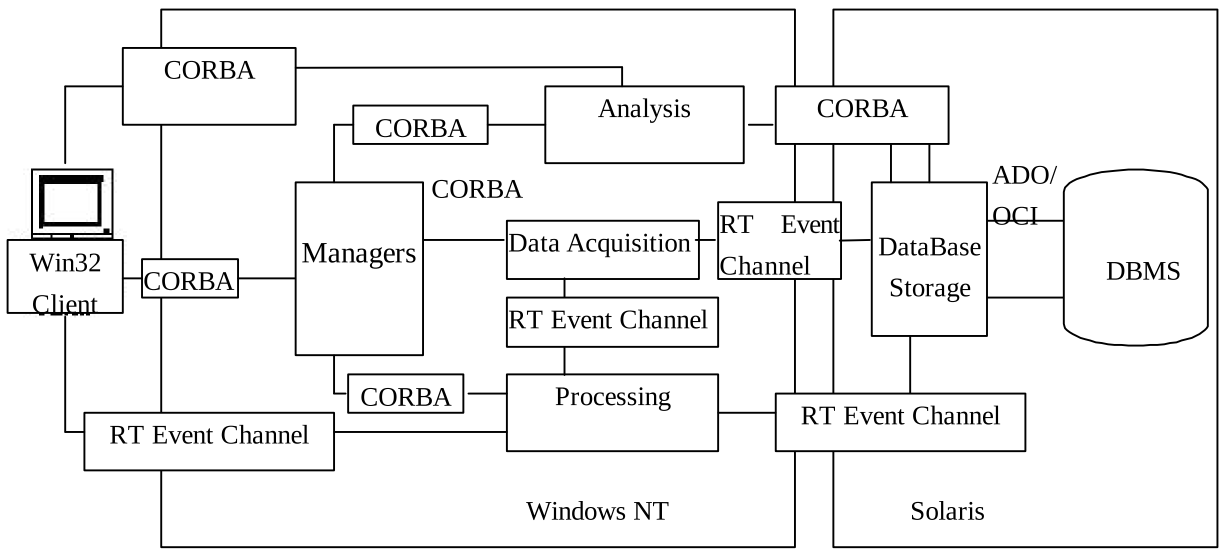

Figure 1: Architecture of DDAPS

Data of DDAPS are transferred through CORBA Service – Event Service (Fig.2). The service allows to transmit data according to "one-to-many" rule generating an event. All CORBA-objects wishing to receive data have to subscribe to the event in the Event Channel (Fig.2). As soon as data come into the Event Channel, Even Service notifies all subscribed objects about the event.

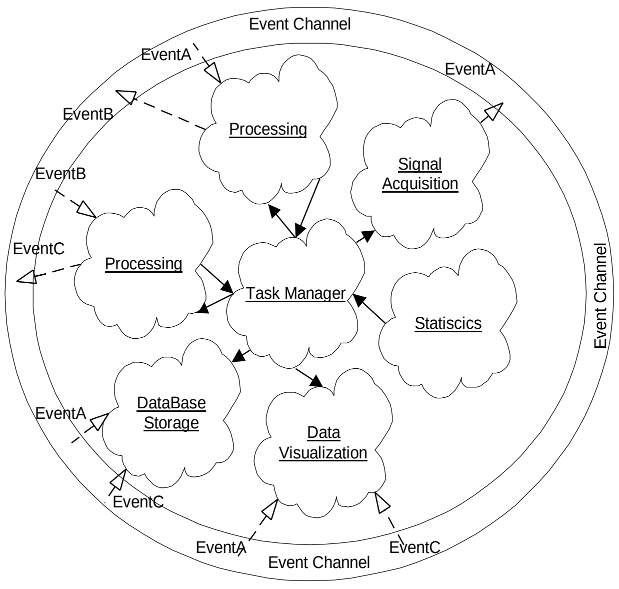

Figure 2: Logical model of DDAPS

The objects may receive data from Event Channel. But Event Service uses TCP/IP protocol to transmit the data. It results in dependence between number of the nodes and time of data transmission. It opposes to provide real-time mode. We will use UDP to transmit the data between the DDAPS's objects.

3 Load-balancing algorithm

The load-balancing algorithm allows to answer the following questions.

1. When the data needs to be repartitioned.
2. What computer should be selected for repartition of a part of workload on it.
3. What scheme to perform the repartitioning to provide real-time mode for data acquisition, processing, and accumulation.

3.1. Criterion of repartition

The main goal of load balancing algorithm in our task is to provide data acquisition, processing, and accumulation *without information loss.* The data have to be redistributed when a threat of data loss is arisen. Let's describe the scheme of data acquisition and processing in DDAPS.

Buffers organized in a queue called "ready queue" are used for filling by the digitized data on the data acquisition node (fig.3). After the digitized data have been written in the buffer they are transferred to the Event Channel. After data transmission to the Event Channel, the buffer is located in the "ready queue" again. If the data are dispatched to the Event Channel too slowly the "ready queue" of buffers will be empty and there will be data loss. The processing objects located on various network nodes accept the data from the Event Channel and write them in the raw data buffers organized in queues liked "raw data queue". Then the buffer is extracted from the raw data queues to be processed, and after processing it is dispatched in the Event Channel for the further accumulation or visualization.

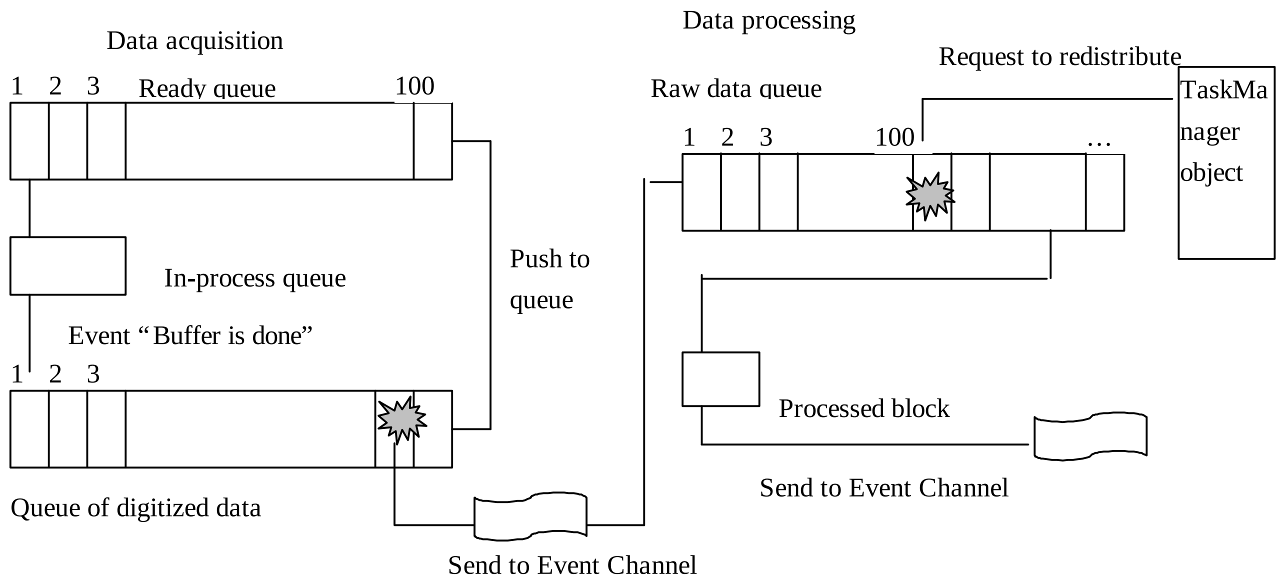

Figure.3: Criterion of repartition

The criterion of repartition is the length of raw data queue located on the processing node. If the length of a raw data queue exceeds length of the "ready queue" then it is necessity to redistribute workload, as the node is not able to process data in time.

3.2. Choice of the node

DDAPS works in the heterogeneous environment where the nodes have various performances and fulfill different tasks. It is necessary to fix the parameters determining "the most suitable" node to redistribute workload on it.

Processor's performance should be one of the parameters having an influence on a choice of the most suitable node. Higher performance node has to be used more often. Length of a raw data queue is the second parameter. If the length of raw data queue is great, the node cannot be considered to add workload to it independently of processors' performance.

The important parameter is the percent of CPU's usage. Heavily loaded workstation is not able to perform additional work. The percent of CPU's usage is sent to TaskManager node from each node in a constant interval.

Thus it is required to fix such node that the choice of the node will result in the fastest decrease of length of raw data queue.

In the current version of DDAPS the most suitable node is selected from a relation

$$G(l,p,u)=100(1-l)+50(1-u)+50p, \qquad (1)$$

where l-relative length of a raw data queue, u - percent of CPU's usage, p – throughput of the node for a specific target. All parameters vary from 0 up to 1. The most suitable node has a maximum of $G(l,p,u)$. We are going to make more exact the formula in future.

3.3. Decimation algorithm

We enumerate some preconditions, which we will use for our algorithm.

1. Data are acquired and processed by blocks. (It is true for science field where DDAPS is used: acoustic and hydrophysics experiments)
2. UDP may be used for data transmission. (The item allows not to take into account time of transmission because time is independent on the number of nodes)
3. Processing algorithm is considered as indivisible (Algorithm may include many parts but all of parts operate on one node).

Initially the only workstation processes the data received from the data acquisition node. When the length of the raw data queue on the processing node becomes more than "ready queue" then the processing object send the message to the control object to request about repartition of workload.

If load balancing is required the most suitable node is selected accordingly the formula (1) and the objects, which are the same as ones on the processing node, are invoked. Then two processing nodes participate in the experiment, each of them processes each second block. The nodes have twice as much time for processing of a block.

If one of the processing nodes is not able to fulfill workload then the supplementary node is enlisted. Half of workload from the heavily loaded node is removed to the supplementary node. Repartition is carried out on 2 always. Each second data block is moved to the supplementary node. Data space is reduced by half and allowed time for processing is increased in 2 times on the heavily loaded node.

In the general case the load-balancing scheme can be described as following.

Let *n* is decimation coefficient used to pick out the blocks and to process them on a node. For example, if *n*=4 the node processes each fourth block. Let *k* is number of the block modulo *n*, i.e. the node processes each *n*-th block with number *M* where *k* is residue of division *M* on *n* (*M*%*n* =*k*). Data repartition is fulfilled on two nodes always. *k* for the first node remains the same as earlier, and *k* for the second node is increased on *n* then *n* is increased in 2 times on the both nodes.

Decimation algorithm may be applied not only to independent block. Sometimes the results of processing of previous block are needed for processing of next block. Example of the task is data filtering by sectioned convolution. In the case processing CORBA-object receives not only data from data acquisition but results of processing of previous block too.

4. Testing results

The load-balancing algorithm is implemented in DDAPS and tested on networked workstations with the Windows 2000 operating system. The computers are networked Fast Ethernet 100 Mb.

The computers described in the table 1 had taken part into the experiments.

Table 1. Description of the computers

#	Description	Purpose	Sampling frequence	
			10 loops	25 loops
A	PII 300 MHz	Data acquisition		
B	PIII 450 MHz	Management objects, Name Service, EventService, Processing	43 KHz	23 KHz
C	PIII 700 MHz	Processing	72 KHz	41 KHz
D	PII 400 MHz	Processing	50 KHz	23 KHz
E	PIII 650 MHz	Processing		36 KHz
F	PIII 650 MHz	Processing		36 KHz

The data acuisition node (A) samples signal with a sampling frequency. Then the digital sinal is send to Event Channel. If the data processing computers are not able to receive the data then data is lost. The maximum sampling frequencies when the data are not lost, are shown in the table 1.

The data processing is Fast Fourier Transform (FFT) several times, in a loop. We had used both direct and inverse FFT. The number of loops allows to vary the processing duration.

The node B has additional responsibilities to manage DDAPS (including the load balancing algorithm). Therefore the maximim sampling frequency is a few smaller than the frequency on the other nodes.

The task of the first experiment is to provide real-time processing for maximum possible sampling frequency (100 KHz). The results are shown on Fig.4.

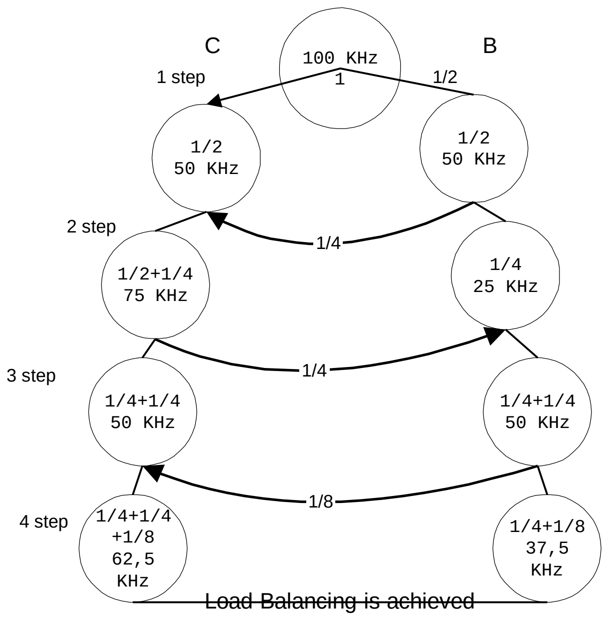

Figure 4: Experiment with 2 processing nodes (10 loops of FFT).

At the beginning of the experiment the node B performed all operation. But when it was not able to process more than 43 KHz (10 loops) then half of workload (each second block) was moved on the node C. Then both the node B and the node C tried to process 50 KHz. When the node B is not able to perform the workload then half of it was moved to the node C. Then the node B worked with 25 KHz and the node C worked with 75 KHz. But the node C may process no more than 72 KHz. Half of the most part of workload (25 KHz) was moved from the node C to the node B. Then each node has to process 50 KHz (25+25). The part of workload (12,5 KHz) was moved from the node B to the node C again. As a result the node C processed 62,5 KHz and the node B processed 37,5 KHz. Load balancing was achieved for 4 steps.

We may suppose a priori the nodes, which provide the processing with sampling frequencies 72 and 43 KHz, are able to process jointly with sampling frequency 100 KHz. But if the overhead charges on the use of the additional node are too great then the nodes are not able to process with the required frequency.

As Fig.3. shows workload after the first step is similar to workload after the third step. It seems that the second and third steps are unnecessary. But it is not true. Always the most part of workload is divided.

Five processing nodes had taken part in the next experiment under 25 loops of FFT(Fig.5).

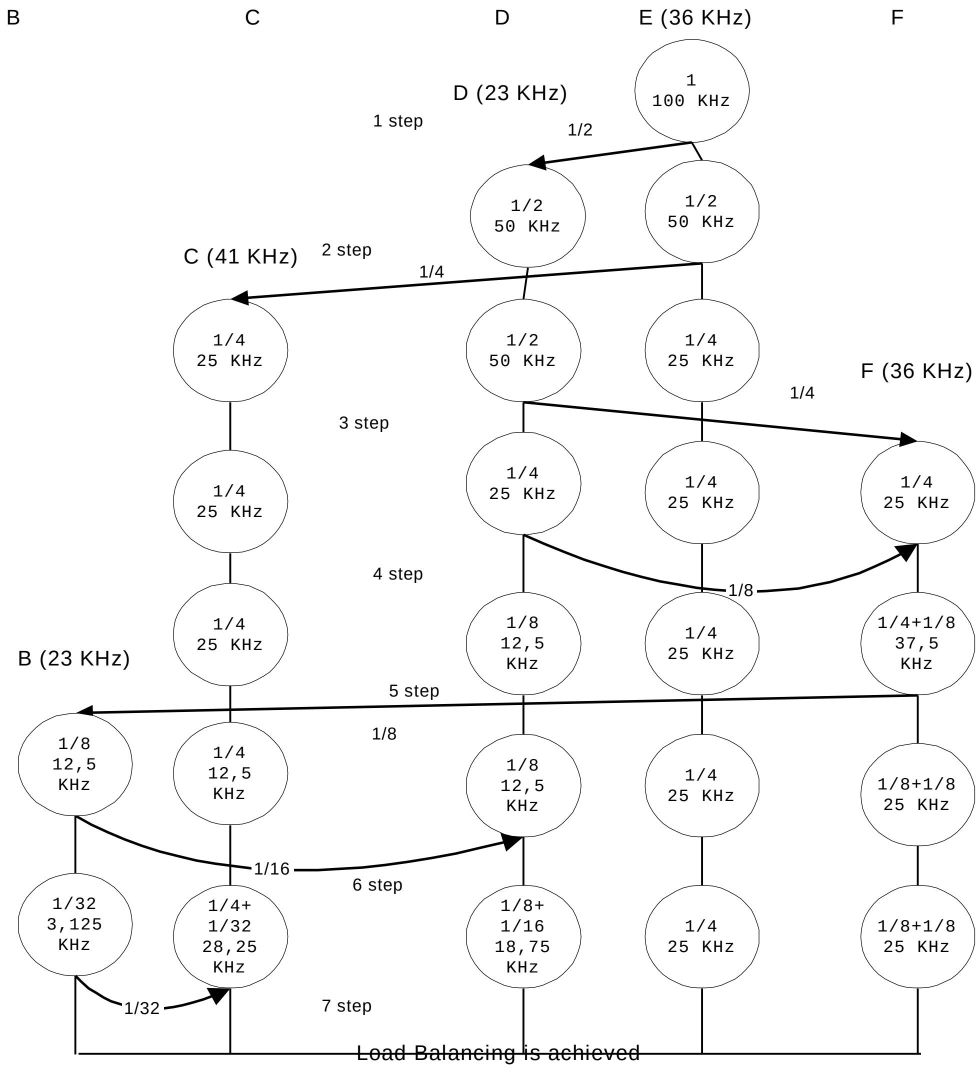

Figure 5: Experiment with 5 processing nodes (25 loops of FFT).

In the experiment (Fig.5) load balancing is achieved at the moment when workload is distributed uniformly in accordance with performance of the nodes. The same results were taken in each experiment with any number of the nodes.

When load balancing is achieved then workloads on the nodes are linearly dependent on performance of the nodes. There are no dead-ends in the algorithm. Therefore either the balance is achieved or the message about impossibility to achieve real-time is generated. One or more nodes may be added to DDASP. It can result in possibility to achieve real-time mode.

The results of experiments have confirmed efficiency of load balancing algorithm to provide real-time mode at least for 10 nodes. The greater number of nodes results in hard load of the node with EventService. Data should be transferred by other methods discussed in [11]. Load balancing algorithm is not changed at the same time.

As experiments show total performance of nodes has to be in 1,3-1,5 times more than required performance to provide real-time. It may be explained by the one of the nodes (TaskManager node) has additional workload (CORBA services: Name Service, Event Service, and objects of management of DDAPS including load-balancing algorithm and Task Manager). Other reasons are that during a predefined interval, all nodes send their load information (the usage of CPU) to the TaskManager node. And overhead on data transmission is considerable.

5. Conclusions

Load balancing algorithm to provide real time mode is used in DDAPS developed for Pacific Ocean Institute of Russian Academy of Science [11-12]. DDAPS allows to acquire, process, display, and accumulate data of different sources. DDAPS is used for processing of acoustic, temperature, seismic, and other data. Algorithm of load balancing is especially important for processing of acoustic data to avoid data loss. It is explained by high sampling frequencies.

References

[1] Ka-Yeung Kwok, Fu-Man Lam, Mounir Hamdi, Yi Pan , and Chi-Chung Hui. Application-Specofoc Load Balancing in Heterogeneous Systems. //High Performance Cluster Somuting. Vol.2., pp. 350-374. 1999

[2] V. Velusamy, I.Banicescu. “Efficient Data Management for Load Balancing Scientific Applications in Distributed Computing Environment with Factoring Methods”.. *Proceedings of Parallel and Distributed Techniques and Applications. Las Vegas, USA. 2000.*

[3] Y. F. Hu and R. J. Blake. Load Balancing for Unstructured Mesh Applications. 2000. http://www.dl.ac.uk/TCSC/Staff/Hu_Y_F/PROJECT/pdcp_siam/

[4] Peiyi Tang and Pen-Chung Yew. Processor Self-Scheduling for Multiple-Nested Parallel Loops. In Kai. Hwang, Steven M. Jacobs, and Earl E. Swartzlander, editors, *Proceedings of the 1986 International Conference on Parallel Processing*, pages 528–535, University Park, Pennsylvania, August, 1986. IEEE Computer Society Press.

[5] Constantine D. Polychronopoulos. Toward Auto-scheduling Compilers. *The Journal of Supercomput-ing*, 2(3):297–330, 1988.

[6] Constantine D. Polychronopoulos and David J. Kuck. Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers. *IEEE Trans. on Computers*, C-36(12):1425–1439, December, 1987.

[7] Marc Willebeek-LeMair and Anthony P. Reeves. Dynamic Load Balancing Strategies for Highly Parallel Multicomputer Systems. Technical Report EE-CEG-89-14, Cornell Univ. Computer Engineering Group, December, 1989.

[8] Susan Flynn Hummel, Edith Schonberg, and Lawrence E. Flynn. Factoring: A Practical and Ro-bust Method for Scheduling Parallel Loops. In *Supercomputing ’91 Proceedings*, pages 610–619, Albuquerque, NM, November 18–22, 1991. IEEE Computer Society Press.

[9] S. Flynn Hummel, E. Schonberg, and L. E. Flynn. Factoring: A Method for Scheduling Parallel Loops. Communi- cations of the ACM, 35(8):90{101, Aug. 1992.

[10]Bruce S. Siegell. Automatic Generation of Parallel Programs with Dynamic Load Balancing for a Network of Workstations. May 5, 1995, CMU-CS-95-168.

[11]Kryukov V.V, Shakheldyan C.J. Real-time in Ethernet network. *Telematika 2001, Peterburg, 2001.*

[12]Kryukov V.V, Shakheldyan C.J. Implementation of the Distributed System for Data Acquisition, Processing and Analysis. . *Proceedings PDPTA'2000 Las-Vegas* . *USA, Vol. 3, pp.1721-1727.*