A Communication Kernel of a Cluster System for Distributed Simulation

H.Valchanov, I.Ruskov, N.Ruskova

***Abstract**: Parallel discrete event simulation (PDES) is a basic approach for evaluation of the complex systems. A PDES attempts to speed up the execution of a simulation by distributing the simulation's workload between multiple processors (DS). A network of workstations is a widely available platform for DS. This paper explores an implementation's details of a communication subsystem for DS. The techniques for reducing of interprocessor communication overload are presented.*

***Key words**: Distributed simulation, communication kernels, cluster of workstations*

INTRODUCTION

Recently the distributed simulation (DS) is becoming a main approach for investigation and evaluation of complex systems behavior. The DS model consists of logical processes (LP). Each of the LPs simulates one or several components of the investigated system. A particular LP can run on a separate processor increasing in such a way the simulation performance. The system components interactions are represented by events, simulated through a message passing between the particular LPs [1].

The communication between LPs in distributed environment has two aspects: one aspect concerns the delivery of event messages, and the second aspect is the LPs synchronization during the simulation [2].

The paper focuses on the implementation features of the communications in the distributed simulation system (DSS), developed by the authors. It is running on LAN of Linux workstations. The communications between LPs on different workstations are based on the transport protocol SCTP (Stream Control Transport Protocol) [4].

The DDS structure consists of three hierarchical layers – *application*, *synchronization* and *communication* layers. The application layer is an abstraction layer. Its goal is to hide the particular implementation details from the LPs. It is developed as library primitives and gives services for logical processes dynamic creation and communication. The underlying synchronization layer is based on the distributed simulation concepts and it executes both the conservative and the optimistic synchronization protocols [1], which are typical for DS. The communication layer provides an interface to implement the message passing in a distributed computing environment.

In order to increase the simulation efficiency the logical processes in DSS are organized into *clusters*. Several LPs are grouped together into one Linux process (cluster) running on a separate workstation. This cluster performs the dispatching of its LPs for execution and their communication with other clusters on the same workstation or on the other workstations in the network.

During the design of the DSS we pursue the following goals:

- Providing an execution environment for the LPs with low dependency on the underlying operating system and a high level of abstraction, hiding the details of its implementation from the user;
- Implementing a reliable and fast communication environment, invisible for the LPs;
- Providing a possibility for parallel execution of separate functions to achieve maximum optimization and acceleration of the simulation processing.

COMMUNICATION SUBSYSTEM OF DSS

The communication subsystem of DSS consists of multiple communication kernels (CK) running on each workstation involved in simulation process. The communication kernels interact with each other by messages using the SCTP protocol. Two kinds of messages with different functionality are introduced:

- Event messages, passed between the LPs for the simulation goals;
- Control messages, passed between the distinguished CK for the cluster synchronization and the simulation control.

For simulation efficiency different delivery schemes are used for the above two kinds of messages. A distributed scheme is used for the event messages delivery, which allows direct communication between any two clusters. The control messages are delivered according to a centralized scheme using *master-slave* approach. According to this approach, one of the workstations is elected as a master cluster. Besides the functions, executed by all of the CKs, the master cluster is responsible for the coordination of control messages and for the handling of the simulation system status information as whole. The application of the centralized scheme eliminates the need to store the other clusters full status information in each cluster. Furthermore, the master cluster supports the information for the correspondence of all clusters' logical and physical addresses.

COMMUNICATION KERNELS STRUCTURE

In order to achieve a layered structure and separation of functional units, the CKs design is based on a modular approach (fig.1). The specific for a given algorithm or simulation concept parts are implemented as separate modules. This approach allows changing in the system functionality through a simple loading of a particular new module and corresponds to the dynamic library (module) loading in Linux. Moreover, the modular approach provides easy portability of the DSS on various operating systems and network protocols.

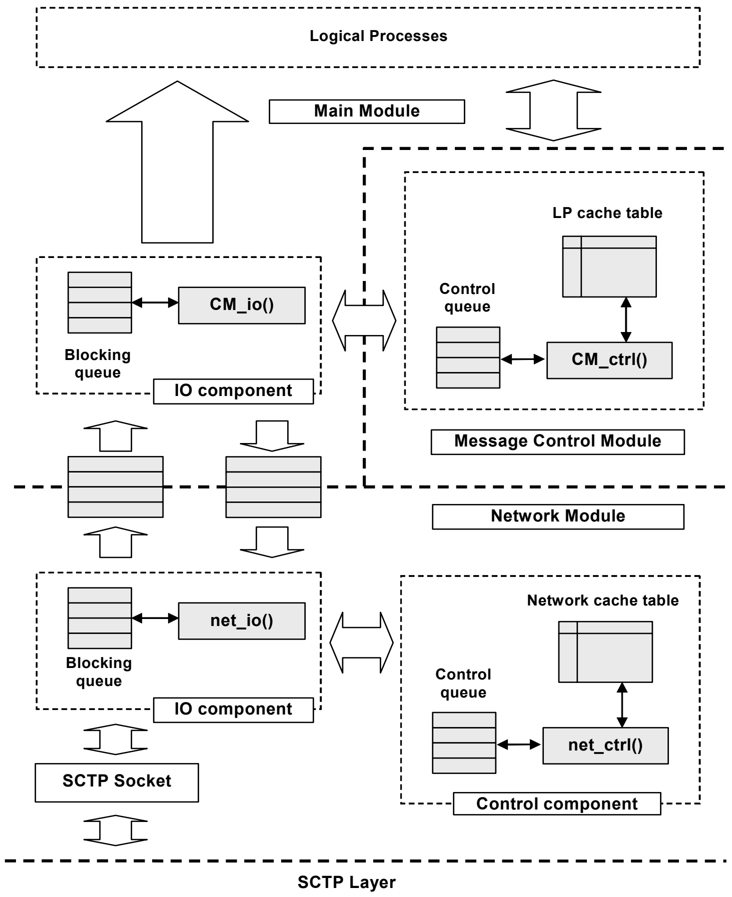

Fig.1 Communication kernel structure

Each communication kernel consists of three independent modules:

- *Network module*. It implements the interface to the underlying transport layer SCTP;
- *Control message module*. It interprets control messages. The functional features of the DSS can be increased through this module;
- *Main module*. It executes the main functions of the CK algorithm.

Each CK provides reliable and fast communications between LPs belonging as to the local cluster, as well as to remote clusters. Communication kernels implement the following main functions:

- Distributed clusters coordination;
- Unified message sending and receiving interface, and reliable message delivery;
- Interpretation of user control messages for simulation monitoring;
- Modular interface to the code, which is specific to a particular implementation;

Each CK supports a number of buffers, implemented as linked lists of messages. These buffers allow the message sending and receiving to be carried out asynchronously (without blocking) as by the LPs, as well as by the other system components.

NETWORK MODULE

The network module consists of three program components – SCTP socket, IO component and control component (fig.1). The SCTP socket is a communication point to SCTP environment. The interaction with the SCTP socket is realized through the user library *sctplib*, which implements the SCTP standard, defined by IETF, in a best way. The IO component performs the algorithm of message interchanging with the network medium. The control component interprets incoming and outdoing control messages.

The interface between the network and the main modules is built on two message queues, one for each direction – from the network module to the main module and vice versa (fig. 1). These queues are implemented as linked lists of messages. The messages from the queues are processed by unified functions, included in the network module. Furthermore, there is one more queue, storing control messages, sent to the network module itself. The control queue is processed by the control component.

The IO component analyzes the type of every incoming message from the SCTP socket and dispatches it to the needed component for processing. When a control message is destined for the local network module, it is moved to the control queue. Otherwise, the message is moved to the main module, putting it into the corresponding queue. Processing an outgoing message, the IO component maps the logical number of the destination cluster to its network address and then passes the message to this address through the SCTP socket. If the mapping process fails, the IO component puts the guilty message into a special type queue, called blocking queue. Then the IO component sends a query to the master cluster for address resolution. The message is sent immediately to its recipient after receiving the response.

The control component is intended to process the incoming control messages for the local network module. It supports specialized network cache table, which maps logical numbers to network addresses of remote clusters. In order to always keep an up to time information, the cache table entries are marked with timestamp, called *time-to-live* (TTL). After the expiration of the TTL the corresponding entry is removed from the table. The only table, keeping the full information about clusters in the whole DDS, is the table in the master cluster. The cache tables in other clusters keep partial information, which has been received from the master cluster during the address resolution.

MAIN MODULE

The main module implements the message delivery to the its LPs, as well as dynamic LP creation and destroying. It supports data structures for LPs representation. Each LP is associated with an input-output message queue.

The IO component in these module deliveries the incoming data messages from the network module to the LPs. The incoming control messages are sent to the control queue of the message control module. The IO component also supports a blocking queue, which is used in the same way as is in the network module.

MESSAGE CONTROL MODULE

In contrast to the control component of the network module, which processes only the control messages for the network module itself, the message control module manipulates with the control messages, related to LPs operations. It supports a cache table, which maps LP's identifier (LPID) to its cluster number. This table holds partial information in contrast of the master cluster's table. This cache table is implemented as an array of pointers to lists of data structures (queues), representing the LPs. The table is frequently accessed, and in case of a large number of LPs, it can reach a huge size. This is a reason to introduce a searching algorithm, based on a hash-function. The hash-function returns a table index, calculated as a module of division of the LP's identifier to the hash-table size (fig. 2). For example, when a LP with LPID 0 is created, it will be cached in the queue on position 0 of the hash-table, the LP with LPID 1 – in the queue on the position 1 and so on. The LP with LPID 1023 will be cached in the queue on the last position 1023, and the next LP (LPID 1024) – in the queue on the first position 0. In this way the queues are filling uniformly during the increasing of the number of LPs and the sequential searching in them is fast.

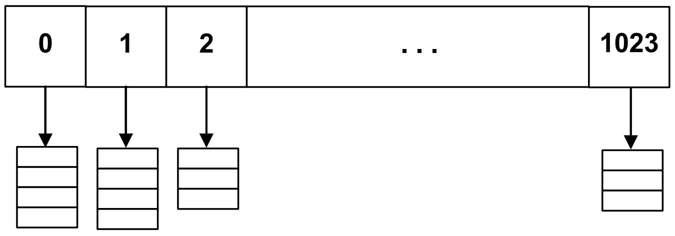

Fig.2 LP hash table

ALGORITHM OF COMMUNICATION KERNELS

One of the main features of the DSS [3] is the possibility of concurrent execution of its components. This concurrency is achieved by multithreading, implemented through the Linux *pthread* library. The communication kernel algorithm is separated from the cluster algorithm, which allows implementing it as an independent thread. This thread is executed concurrently with other threads of the cluster (the thread of the LPs scheduler, the threads of the synchronization protocols). This concurrency increases the performance of the inter-cluster communications.

```
for (;;) {
     net_io();
     net_ctrl();
     CM_io();
     CM_ctrl();
     CM_garbage();
}
```

Fig.3 Algorithm of the communication kernel

The simplified algorithm of the communication kernel operation is shown on fig.3. The communication kernel thread executes infinitely the main operations sequence, which implements the algorithms of the CK modules: the network module IO and control component (*net_io, net_ctrl*), the main module IO component (*CM_io*) and the message control module (*CM_ctrl*). The last operation (*CM_garbage*) is an operation of a garbage-collector type. This operation is used for destroying CK objects, which are no longer used, thus releasing the occupied memory.

CONCLUSION

The main goal of the developed communication system is to organize the resources of LAN of workstations, connected by Ethernet and running Linux, to work in accordance with each other for the distributed simulation purposes. The current implementation of the system is developed in C using the multithreaded programming technique. Some of the features include:

- Layered structure to enable easy and fast reconfiguration;
- Module structure. The using of the dynamic module loading allows a portability of the system to various network platforms, as well as adding of new communication protocols;
- Asynchronous event messages sending and receiving;
- Optimized distributed mechanisms for the LPs and cluster addressing;
- Optimized technique for message dispatching.

The system developed can be easy modified for other classes of distributed applications. Feature work towards the system integration into a Web based distributed simulation, is considered.

REFERENCES

[1] Fujimoto, R. Parallel Discrete Event Simulation. CACM, 1990, No.10, 30-53.

[2] Hara, V., Harju, J., Ikonen, J. Simulation of Cellular Mobile Networks in a Distributed Workstation Environment, Annual Review of Communications, 1997, 913-917.

[3] Ruskova N., Walchanov H. Run-time system for a distributed simulation environment over network of workstations. Proc. of the CompSysTech'2000, Sofia, 2000, I.8-1 - I.8-6.

[4] R. Stewart, Ch. Metz, SCTP. New Transport Protocol for TCP/IP, IEEE Internet Computing, November – December 2001.

ABOUT THE AUTHORS

Assist. Prof. Hristo Valchanov, Department of Computer Science and Engineering, Technical University of Varna, Bulgaria. Phone: +359 52 383424, E-mail: hristo@tu-varna.acad.bg

Ms.S. Ivan Ruskov, ESME Sudria, Paris, France. E-mail: ruskov_i@yahoo.fr

Assoc. Prof. Nadezhda Ruskova, PhD, Department of Computer Science and Engineering, Technical University of Varna, Bulgaria. Phone: +359 52 383424, E-mail: ruskova@tu-varna.acad.bg