- partitioning $\widehat{H}$ into binary cube like vertical extensions of its upper homogeneous elements and applying Hansel's method for them,
- identifying monotone functions defined on $\widehat{H}$ with use of additional information on location of their upper units through L_{min} and L_{max}.

The choice of concrete resource set depends on requirements of certain applications.

Conclusion

Algorithmic resources are considered for elaboration and identification of monotone functions. Current research proposes two new components - partitioning the multi-valued cube through binary cube like vertical extensions of its upper homogeneous elements; and learning upper homogeneous area through the analogous partitioning. The choice of concrete resource depends on requirements of certain application.

Bibliography

[M, 1959] V. Mickeev. On sets, containing maximal number of pair wise incomparable Boolean vectors, Prob.Cyb, 2, 1959.

[K, 1965] V. Korobkov. On monotone functions of algebra of logic, Prob.Cyb, 13, 1965.

[K, 1981] A. Korshunov. On the number of monotone Boolean functions, Prob.Cyb, 38, 1981.

[H, 1966] G. Hansel. Sur le nombre des fonctiones booleennes monotones de n variables, C.R.Acad.Sci., Paris, 262, N.20, 1966.

[T, 1979] G. Tonoyan. Cain partitioning of n-cube vertces and deciphering of monotone Boolean functions, Journal of Computational Mathematics and Math. Physics, 1979, vol. 19, N.6.

[A, 1976] V. Alexeev. On deciphering of some classes of monotone many valued functions, Journal of Computational Mathematics and Math. Physics, 1976, vol. 16, N.1.

[K, 1978] Katerinochkina N., On sets, containing maximal number of pair wise incomparable n-dimensional k-valued vectors, Mathematical notes, v. 24, no, 3, Sept. 1978.

[AS, 2001] L. Aslanyan, H. Sahakyan. On the boundary cases of partitioning of subsets of the n-dimensional unit cube, "Computer Science & Information Technologies" Conference, Yerevan, September 17-20, 2001

[S, 2006] H. Sahakyan. Numerical characterization on n-cube subset partitioning, submitted to Discrete Applied Mathematics.

Authors' Information

Hasmik Sahakyan – *e-mail:* hasmik@ipia.sci.am

Levon Aslanyan – *e-mail:* lasl@sci.am

Institute for Informatics and Automation Problems, NAS Armenia, P.Sevak St. 1, Yerevan-14, Armenia

DYNAMIC DISTRIBUTION SIMULATION MODEL OBJECTS BASED ON KNOWLEDGE

Alexander Mikov, Elena Zamyatina, Konstantin Osmehin

Abstract: This paper presents the process of load balancing in simulation system Triad.Net, the architecture of load balancing subsystem. The main features of static and dynamic load balancing are discussed and new approach, controlled dynamic load balancing, needed for regular mapping of simulation model on the network of computers is proposed. The paper considers linguistic constructions of Triad language for different load balancing algorithms description.

Keywords: Distributed calculations, distributed simulation, static load balancing, dynamic load balancing, expert systems

ACM Classification Keywords: I.6 Simulation and Modeling I.6.8 Types of Simulation - Distributed: I.2 Artificial Intelligence I.2.5 Programming Languages and Software - Expert system tools and techniques

Introduction

The complicate problem solutions with employment of high performance computers are very vital now. This is an actual problem for the simulation too. It is imposed by inevitability of increasing computation capacity and simulation running optimization on the one hand [1,2,3]. The simulation model objects are distributed across the calculation nodes in cluster or in local area network (LAN) or in wide area network (WAN) and interact by passing messages from one to another. From the other hand, it is necessary to use distributed simulation in order to combine already designed simulation models or because of inevitability to organize the conjoint work of some investigators participating in a common simulation experiment.

The computing environment heterogeneity and simulation model heterogeneity can be the reason of load balancing violation during simulation run. The heterogeneity of calculating environment is associated with varied capacity of calculating nodes and diverse capacity of communication lines. The simulation model heterogeneity is based on the fact that some of objects wait some event occasion but the other are in the state of calculating almost all the time initiating appropriate events. Besides, some objects may communicate with high intensity whereas the communication between the other of them is seldom the case. So unbalance would negate the benefit of distributed calculation.

That is why a demand arose in algorithms designing and programming tools development needed for load balancing preservation. It was correctly reasoned for distributed simulation and for distributed calculation as a whole [2,3].

The load balancing algorithm must be optimal for any model with any structure and any mechanism of time advancement. But it is not the easy problem. A considerable number of investigators attempted to design load balancing algorithms [2, 3]. But these algorithms were designed only for specific problems or they changed the original program code [2]. From the above discussion it appears that the new approach to load balancing problem is needed. Authors [5] suggest this approach: controlled load balancing algorithm based on knowledge. The paper considers the target setting, the summary review load balancing algorithms (and the load balancing algorithms used in distributed simulation) and the architecture of load balancing programming tools. Furthermore the authors propose the architecture of a load balancing subsystem for distributed simulation system Triad designed by them and suggest linguistic constructions for load balancing algorithms description.

Load balancing of computing nodes during simulation run

Most often the components of system being simulated are presented in PDES (PDES-Parallel Discreet Event Simulation) as logical concurrent processes ($LP_i, i=1 \div n$). Logical processes are allocated on calculating nodes (nodes of cluster or network) and interact by passing messages from one to another. One can find out the conflict of good balanced logical processes allocation and low speed of message interchange (communication lines have low capacity or these lines are overloaded) during simulation run. One can observe another situation: the time which is needed for simulation objects communication almost equal to zero, and in the same moment some computers (processors) are waiting the work and the other ones are overloaded. On the other hand good balanced system may demand a great amount of time for communication. So we can conclude that the strategy of load balancing system supposes regular workload of processors and not overloaded communication.

One must differ static and dynamic load balancing. Static load balancing must be fulfilled before the simulation experiment using the prior runtime data (in SPEEDES[2], for example) or structural characteristics of simulation model (Triad.Net). Triad.Net simulation system proposes to allocate a simulation object and substructure of this object (the structure of Triad simulation system is hierarchical) in one calculating node. Moreover Triad.Net load balancing subsystem finds out cliques and allocates them in one computing node too in order to reduce communication time.

But preliminary allocation of simulation objects is not effective always.

This is explicable on the basis of follow facts:

- A simulation model can be changed in the course of simulation run because of scheduling new events, new processes appearance, terminating some processes;
- A calculating environment can be changed because one or several processors (or computers) are failed;

– A processor (or computer) is used to calculate not only a simulation model but it may carry out another calculating works and portion of these works may increase with the time.

In any case, the benefit of distributing logical processes between calculating nodes before the simulation experiment very often cannot be found out. A dynamic load balancing supposes distribution of logical processes during simulation experiment. One can single out some stages in a dynamic load balancing process: an assessment of workloads, an initiation of load balancing, decision making of expediency of load balancing, migration of workloads from one node to another.

Automatic load balancing process makes a decision on workload (object of simulation model) migration from one node to another relying on data collected during simulation experiment. These data are saved in the data base and include data of two types:

– a data about simulation model (the frequency of exchange between simulation model objects, the number of objects allocated in the same calculating node, duration of some logical processes);
– a data about computing environment (computing load of calculating node, the load of communicating lines, calculating nodes lay-up and so on).

Besides, it is very important to have information about communication of processes (the topology of exchanges). One can assess the workload of a processor analytically basing on knowledge about simulation model behavior. Another way to assess workload is to make metrology during the simulation run. The most of modern computers contain time counters needed to evaluate the execution time for every application. There are special programming tools for data collection.

After that it is necessary to find out the load balancing violation and to make a decision about migration of objects from one calculating node to another because the frequent migration may reduce a benefit from load balancing.

User can find out unbalance using one of two ways:

– *Synchronously:* all the calculating nodes interrupt their run in some definite moments of synchronization, and load unbalance can be defined by comparison of workload of diverse computing nodes.
– *Asynchronous:* every calculating node has the history of its workload. So there is no single moment of synchronization needed for unbalance determination. Background process functioning across simulation model during simulation experiment evaluates workload.

Load balancing subsystem makes a decision about simulation nodes migration:

– In centralized manner which means that some special computer carries out data collection (data includes information about computing environment state) and makes a decision about the migration of the simulation objects.
– In distributed manner which means that each calculating node carries out its own load balancing algorithm. According to this algorithm simulation model objects could be transferred only to adjoining nodes.

The last is the stage of object migration and it is necessary to provide object integrity.

So a great amount of algorithms and strategies of load balancing exists. Nevertheless there are common features in architectures of various subsystems and common stages in various load balancing algorithms. Let us list load balancing subsystem components: first of all it is a component providing the distributed simulation model state assessment and estimation of computing environment characteristics, the control program which chooses logical processes and makes a decision about the moment of migration, the program tools providing the migration of the objects from one calculating node to another, data base with information about simulation model objects and computing environment and at last subsystem of visualization. This subsystem must show the simulation model objects mapping on computing environment, the scheme of communication, simulation model changes and computing environment transformation.

And now let us consider simulation model and load balancing subsystem in Triad.Net more precisely (Triad.Net is new distributed simulation system, it is advanced version of CAD and simulation system Triad).

Simulation Model in Triad

Simulation model in Triad.Net is represented by several objects functioning according to some scenario and interacting with one another by sending messages. Simulation model [4] is μ={STR,ROUT,MES} and it consists of three layers, where STR is a layer of structures, ROUT – a layer of routines and MES – a layer of messages appropriately. The layer of structure is dedicated to describe the physical units and their interconnections, but the layer of routines presents their behavior. Each physical unit can send a signal (or a message) to another object. So, each object has the input and output poles (P_{in} – input poles are used to send the messages, P_{out} – output poles serve to receive the messages). A message of simple structure can be described in the layer of routines, but a message of the complex one – only in the layer of messages. Many objects to be simulated have a hierarchical structure. So their description has a hierarchical structure too. One level of the structure is presented by graph P = {U, V, W}. P-graph is named as graph with poles. V is a set of nodes presenting the physical units of object to be designed, W – a set of connections between them, U – a set of external poles. The internal poles are used for information exchange within the same structure level; in contrast, the set of external poles serves to send signals (or more complex information) to the objects situated on higher or underlying levels of description. Special statement *out* (out <message> through <name of pole>) is used for message sending. A set of routines is named ROUT- routine layer.

Special algorithms – routines – define the behavior of a physical unit and are associated with particular node of graph P = {U, V, W}. Each routine is specified by the set of events (E-set), the linearly ordered set of time moments (T-set), the set of states {Q-set}. State is specified by the local variable values. Local variables are defined in routine. The state is changed if only an event occurs. One event schedules another event. So simulation system Triad.Net is a discrete-event one. Routine (as an object) has input and output poles (Pr_{in} and Pr_{out}). An input pole serves to receive messages, output – to send them. One can pick out input event e_{in}. All the input poles are processed by an input event, an output poles – by the other (usual) event.

Simulation system Triad.Net is advanced Triad system, but it is the distributed/parallel one. Conservative and optimistic algorithms were designed in Triad.Net.

Special subsystem called analysis subsystem includes special objects of two types. Some of them are named "information procedures". Information procedures examine simulation results and inspect the simulation run. Information procedures are "connected" to nodes or, more precisely, to routines, which describe the behavior of particular nodes during simulation run. Information procedures act as monitors of a simulation model. Each information procedure can watch several simulation model objects in same time and in any time of simulation run. The second type of analysis subsystem objects is "*conditions of simulation*". "*Conditions of simulation"* is a special linguistic construction defining the algorithm of investigation (it includes a list of information procedures dedicated to examine a simulation model during concrete simulation run). In order to examine the simulation model component from another point of view one can use other conditions of simulations with the new list of information procedure names. Besides, conditions of simulation define the condition of simulation run termination.

It must be remembered that the simulation model in Triad.Net is not static. There is a special type of variable – type "model" in Triad language and several operations with the variable type "model". These operations are defined for the model in general and for each layer. For example, one may add or delete a node, add or delete an edge (arc), poles, union or intersection of graphs. Routine layer permits to add or delete any event, layer of messages – to add or delete types or selectors. Besides, one or another routine (routine layer) using some rules can be assigned to the node (structure layer). The behavior of the object associated with this node would be changed. Besides, one does not need to retranslate the model. But we see that the structure of simulation model may be changed, and so we have an additional reason of dynamic load balancing procedure existence.

Load balancing subsystem in Triad.Net

Load balancing subsystem is dedicated to optimal distribution simulation model across calculating nodes (in multiprocessor computer or in network) and consequently to enhance the performance of these computers. Load balancing it is a problem of non isomorphic vertex-connected graphs mapping B: TM $\rightarrow$ NG, where TM – a set of graphs of simulation models, NG – a set of graphs – computer network configurations. Graph G $\in$ NG, G = {C, Ed}, can be defined by a set of calculating nodes C and a set of edges Ed (edges ED are associated `with

communication lines). One can consider NG as a super graph, containing all eventual (admissible) graphs G as sub graphs. Graph $M \in TM$, $M = \{U, V, W\}$ represents simulation model.

Let us consider three kinds of load balancing [5]: static B_s, dynamic (automatic) B_a and dynamic (controlled) B_c.

This is a generally recognized result – the existence of sub graph $G \subset NG$ which is isomorphic to simulation model M. But usually this graph does not exist, so it is valuable to find closely corresponding graph.

In regards to dynamic balancing B_a graphs G and M are considered to be loaded. The nodes of the first graph have a parameter – performance, edges – data rate. The characteristics of nodes in the second graph – time complexity, the characteristics of edges – the intensity of a traffic flow.

The weights of nodes and edges in graph NG (and consequently, any subgraph of this graph) are considered to be known. The corresponding graph M parameters must be defined during the simulation run. The "bottle neck" of simulation model and computer system is determined in accordance with some algorithm, and migration of the simulation model objects without interruption of simulation run is fulfilled. The automatic load balancing algorithm may be described in Triad language.

One can define better distribution simulation model objects across the nodes of graph G for the following simulation experiments using data collected by information procedures during previous simulation runs and genetic algorithms. Automatic dynamic balancing algorithm uses antecedents of the simulation run in order to plan the future simulation run. But the actual simulation model behavior may not correspond to this prediction.

Really it is well known that there is no a considerable gain in performance when employing automatic load balancing algorithm. Only the designer of simulation model knows the behavior of simulation model in specific situation: for example, the simulation model designer knows that the intensity of a flow of requests after 600 units of simulation time increases. Another example: the exchange intensity between two nodes becomes very high after 300 units of simulation time. Load balancing subsystem must map these two nodes of graph M in the same calculating node (graph G) or in two adjoining nodes of graph G with powerful communication line. So it becomes clear, that the efficiency of load balancing may increase if we develop some special tools to control the load balancing process.

There is a new approach of implementation of controlled dynamic load balancing based on knowledge in Triad.Net. Controlled dynamic load balancing subsystem includes expert component and information procedures developed by a model designer (non standard information procedures in other words). Expert component consists of optimization rules defined by the author of the given model (or of class of models). Non standard information procedures are intended to estimate the events (or conditions) of rule applications.

Load balancing subsystem architecture

So, knowledge-based load balancing subsystem includes:

- Expert system with knowledge base, rules editor, inference engine and module of explanation. Knowledge base consists of rules for optimal distribution of simulation model objects across the calculating nodes.
- Simulation model and computation environment analysis subsystem. Analysis subsystem consists of information procedures to collect data on simulation model objects behavior (the frequency of interchanges among the simulation model objects, the frequency of event occurrence and so on) and to collect data on computational environment (flow capacity of communication lines, workload of computers).
- Subsystem for simulation model and calculating environment visualization. Subsystem of visualization present diagrams and plots. User has an opportunity to choose information representation enjoying these or other information procedures. Besides subsystem must to represent simulation model mapping on calculating environment.
- Migration subsystem which carries out simulation model migration from one computing node to another.

As it was mentioned above special objects – information procedures observe simulation model objects, more precise, they observe local variables changing, events occurrences and fix the facts of messages sending and receiving. Information procedures can analyze and compare local variables of different routines in the same moment during simulation run. Subsystem of analysis includes standard information procedures. Besides user can design his own original one using Triad language.

The information procedures visualize characteristics of simulation model objects during simulation run. If the values of some characteristics are equal to some limit values the information procedures send a message to expert component. Expert component carries out some operations on graph G (this graph represents the structure of simulation model) mapped on graph M – graph of calculating environment.

The rules could be formulated as so:

- If there are more than 2 active objects in i-th computing node but j-th node is free then migrate one of simulation model object to j-th free;
- if there are two objects V_i and Vj in simulation model and they exchanges by messages and if there is communication line E_{dk} which communication capacity is rather high then it is advisable to map arc w_{ij} on E_{dk};
- if one can observe high intensity of two simulation model objects (V_i and V_j) exchanging by messages and there is free computing node D_i then it is advisable to migrate (V_i and V_j) in D_i

So rules imply operations on graph G. Rules are productions such as «if then else…» and could be described by Triad language. One can describe the automatic load balancing algorithm by Triad.Net too.

Language constructions in Triad for load balancing description

Let us show a little fragment of Triad program describing automatic load balancing algorithm. It is an algorithm from distributed simulation system SPPEEDES [3]. Following this algorithm one must choose the most loaded computation node in the case of unbalance occurrence. Further it is necessary to choose one or more simulation model objects which are situated on this overloaded node and to carry out the migration of chosen objects. The choice of objects is random.

Simulation model description begins with key word model (Triad language) and the end of Triad-program is marked with key word *endmod*. The model description includes a description of the layer of structure (*structure …endstr*), a description of layer of routines (*routine … endrout* (the behavior of simulation model objects)) and a layer of messages *message…endmsg* (messages with rather complicate structure). Let us suppose that the structure of simulation model G is represented a graph with nodes A,B,C,D,E,F (a complete graph with nodes A,B,C,D and two nodes E and F adjoining node D. This nodes a connected with edge.

model Mod1;

var graph G; *structure* S *def*…G1:=compl(A,B,C,D)+node(E)+Node(F)+edge(E<>F)+edge(D<>E)+ edge(D<>F) …endstr

…G:=S; (* we described the structure of simulation model, compl – graph constant «a complete graph»*)

It is necessary to use structure layer of Triad language to describe calculating environment.

….*Var graph* M; (* Calculating environment description, calculating environment has a star topology, network consists of three computers with computer named O in the center of star*)

structure S1 def … G2:=*star*(3)(O,P,R,S) …*endstr*; M :=S1: …(*структура ВС*).

Now we shall consider the description of a static load balancing. Let us use a procedure which carries out the preliminary mapping of simulation model objects. This procedure must be executed before a simulation run. So it is advisable to allocate the call statement of this procedure in language construction "conditions of simulation". This part of program code executes before other one. So load balancing procedure may be represented as follow:

procedure Static_Load_Balancing (*in ref node* G1, *ref node* V,W,X,Y,Z) *def* X:=G1; Y:=V; Z:=W;*endproc*

The algorithm of automated load balancing is presented as procedure *Automated_Load_Balancing.* Following this algorithm a simulation model object in one of computing nodes is selected by accident. The name of computing nodes is one of a parameters of procedure mentioned above. Selected simulation object is migrated to another computing node. The name of this node is an input parameter of procedure too. The names of nodes are defined as a most loaded and least loaded by standard information procedure. Let it be M.P and M.R.

procedure Automated_Load_Balancing (in ref node X,Y) def

set U *of node* (G.A,G.B,G.C,G.D); *ref node* Q; Q:= *random*(U); Migrate(Q) *from* (X) *to* (Y)

endproc

The linguistic construction "conditions of simulation" may include call statement of procedure Automated_Load_Balancing too (in main part after the call statements of information procedures).

Conditions of simulation Running …

Initial … Static_Load_Balancing(compl(G.A,G.B,G.C,G.D,G),G.E,G.F,M.P,M.R,M.S) …*endi*

….(*call statements of standard information procedures which are used for data collection*)…

(*call statement of automated load balancing procedure*)

Automated_Load_Balancing (M.P,M.R); *processing* ….*endproc*;

Endcond

Conclusions

So this paper describes the load balancing subsystem architecture and linguistic construction for their description in Triad language. These linguistic constructions are rather convenient for original load balancing algorithms description. This algorithms may be designed by investigators themselves and it may be more adequate for the specific simulation model behavior during simulation run. Load balancing subsystem in Triad.Net includes not only programming tools supporting static and automatic dynamic load balancing but programming tools for implementation of controlled dynamic load balancing based on knowledge too. The rules must be defined by investigator knowing the behavior of specific simulation model (or already known rules for similar simulation model may be modified).

Aknowledgements

This paper is supported by RFFI (RF) under project 07-07-412-a

Bybliography

1. [Fujimoto, 2003] Fujimoto R.M. Distributed Simulation Systems. In Proceedings of the 2003 Winter Simulation Conference S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds. pp. 124-134
2. [Wilson, 1998] Wilson L.F. and Wei Shen. Experiments In Load Migration And Dynamic Load Balancing In Speedes. Proceedings of the 1998 Winter Simulation Conference.D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds, pp.590-596
3. [Zheng, 2005] Zheng G. Achieving High Performance on Extremely Large Parallel Machines: Performance Prediction and Load Balancing; in Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 2005,165p. Доступно на сайте: http://charm.cs.uiuc.edu/
4. [Mikov, 1995] Mikov A.I. Simulation and Design of Hardware and Software with Triad// Proc.2nd Intl.Conf. on Electronic Hardware Description Languages, Las Vegas, USA, 1995. pp. 15-20.
5. [Миков, 2005] Миков А.И.., Замятина Е.Б., Осмехин К.А. Метод динамической балансировки процессов имитационного моделирования. В кн. «Материалы Всероссийской научно-технической конференции «Методы и средства обработки информации МСО-2005». М.: Изд-во МГУ, 2005, стр.472-478.

Authors Information

Alexander Mikov – The Institute of Computing, the director, Full professor, RF, Krasnodar, Aksayskaya, 40/1-28; e-mail: alexander_mikov@mail.ru

Elena Zamyatina – Perm State University, associate professor, Computer Science Department, RF, Perm, 614017, Turgeneva, 33,. 40; e-mail: e_zamyatina@mail.ru

Konstantin Osmehin – Perm State University, post graduator, Computer Science Department, RF, Perm, 614017, ул. Gashkova, 28-12; e-mail: kosmehin@lukoilperm.ru