Multiagent Learning Model in Grid

QingKui Chen 1, Lichun Na2
[1] School of Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China

[2] Dept. of Information Science, Shanghai LIXIN University of Commerce,
Shanghai 201620, China

Summary
For improving the efficiency of resource use in dynamic network environment, the computational model base on multiagent is adopted, and an effective learning model based on the reinforcement learning and the multi-level organization learning is proposed in this paper. A series of formal definitions, such as the dynamic network grid (*DNG*), the computing agent, the cooperation computing team and the relations among them, were given. The rules are classified into the basic rules, the static rules and the dynamic rues. Using the generation technique of dynamic knowledge, the knowledge revision technique based on the reinforcement learning and the learning framework based on multi-level organizations of agents, the learning model was studied. The migration learning process was described in *DNG*. The experiment results show that this model resolves effectively the problems of optimization use of resources in *DNG*. It can be fit for grid computing and pervasive computing.
Key words:
multiagent; learning model; migration learning; grid.

1. Introduction

With the rapid development of the information techniques and their popular applications, and facing the problems of complex computations and massive data process, the demand for the high performance processing devices is becoming more and more vehement. Nowadays, the numbers of Intranet composed of many computer clusters are quickly increasing, and a great deal of cheap personal computers are distributed everywhere, but the using rate of their resources is very low[1-3]. The grid [4] techniques and pervasive computation can become the main approach to use effectively these resources. Mining and adopting these idle resources, we can get a lot of large-scale high performance computation, storage and communication resources which are not special. However, the dynamic and unstable characteristic of these resources brings the huge obstacle for us. The multiagent [5, 6] techniques have already become the feasible solutions for supporting the grid computing and the pervasive computing in dynamic network environment. There are many successful examples [7] [8] of researches and applications which are in conjunction with the automatic multiagent system. The one of key techniques about multiagent is the improving intelligence by self-learning. There are a lot of researches about agent learning [9-13]. It includes two aspects: passive learning and active learning. The main theory of active learning is the reinforcement learning. However, these techniques can't be fit the dynamic and unstable network computation. The problem of distributed and cooperative multiagent learning is studied through complex fuzzy theory in paper [14], but it can't satisfy the need of dynamic network. On the other hand, the more effective solutions are the learning during the migration process and the agent organization learning [15, 16].
This paper proposed a multiagent learning model, which support the grid computing and pervasive computing in dynamic network environment that is composed of many computer-clusters connected by Intranet. Using of the multi-level cooperation agent organizations and adopting the dynamic knowledge revising based on reinforcement learning, we studied and implemented this model. This model can support the grid computing and pervasive computing based on task-migratory mechanism, and it can fit the heterogeneous and dynamic network environment.

2. Definitions of architecture

Definition1. Computing Node (CN). *CN* is defined as *CN* (*id*, *CT*, *Am*, *AS*), where *id* denotes the identifier of *CN*; *CT* denotes the type of computing node (defination7); *Am* denotes the main control agent of *CN*; *AS* is the set of agents running on *CN*.
Definition2.computer cluster (CC). *CC* is defined as *CC* (*Ma*, *CS*), where *Ma* denotes the main computer of *CC*; $CS = \{CN_1, CN_2 \ldots CN_p\}$ denotes the set of all computing nodes which *CC* includes;
Definition3.dynamic network grid (DNG). *DNG* is defined as *DNG* (*Ma*, *CCS*, *N*, *R*), where *Ma* denotes the main computer of *DNG*; *CCS* denotes the set of all computer clusters which *DNG* includes; *N* is the

Manuscript received August 25, 2006.
Manuscript revised August 30 , 2006.

connection network set of *DNG*. *R* is the rules of connections.

Definition4. Computing Agent (CA) . *CA* is defined as *CA* (*id, PRG, BDI, KS, CE*), where *id* denotes the identifier of *CA*; *PRG* denotes the executable program set of *CA*; *BDI* is the description of its BDI; *KS* is its knowledge set; *CE* is its configuration environment.

A *CA* is the basic element to execute computation task. If a *CA* could complete independently the task, we call it as the **independent computing agent** (*ICA*). If a *CA* couldn't complete independently the task, and it must cooperate with others, we call it as the **cooperative computing agent** (*CCA*).

Definition5. Cooperation Computing Team (CCT) . *CCT* is defined as *CCT* (*id, Am, CAS, BDI, CKS, CCE*), where *id* denotes the identifier of *CCT*; *Am* denotes the main control agent of *CCT*; *CAS* denotes the set of all cooperative computing agents which *CCT* includes; *BDI* is the description of its BDI; *CKS* is its knowledge set. *CCE* is its configuration environment.

Definition6. Global Computing Group (GCG) . *GCG* is defined as *GCG* (*id, Am, ICAS, CCTS, GKS, GCE*), where *id* denotes the identifier of *GCG*; *Am* denotes the main control agent of *GCG*; *ICAS* denotes the set of *ICA* which *GCG* includes; *CCTS* denotes the set of *CCT* which *GCG* includes; *GKS* is its knowledge set. *GCE* is its configuration environment.

Many tasks are executed together in *GCG* during the same time, and the tasks are calculated by a lot of *ICA*s or *CCT*s. A *DNG* can support many units of *GCG*. Because of the dynamic variety of *DNG*, the computing agents (*ICA*, *CCT*) are often migrated in *DNG*. Owing to migration of the agents, the environment of the computing agents (*ICA*, *CCT*) is changed in dynamic. So, their behavior and knowledge should be adjusted automatically in order to keep with dynamic demand. The relations between *DNG* and *GCG* are presented in Figure 1.

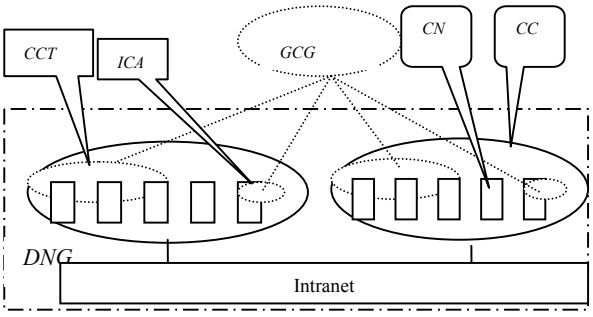

Figure1. The relations between *DNG* and *GCG*

Definition7. Computing Node Type (CNT). *CNT* is defined as *CNT* (*cpu, mem, disk, net*), where *cpu* denotes the power of processor of *CN*; *mem* denotes the power of storage of *CN*; *disk* denotes the power of the input/output of *CN*; *net* denotes the power of communication of *CN*.

According to the real conditions of each *CN* in *DNG*, the *CN* types can be formed a type set *CTS*= $\{CT_1, CT_2,...,CT_{ct}\}$, and the *CTS* is called as the set of computing node type

Definition8. Network Type (NT). *NT* is defined as *NT* (*B, PRT*), where *B* denotes the network bandwidth of *CC*; *PRT* denotes the network protocols of *CC*.

According to the real condition of networks in *DNG*, the network types can be formed into the type set *NTS*= $\{NT_1, NT_2,...,NT_{nt}\}$.

Definition9. Basic Rule (br). *br* is defined as *br* (*id, rul, MRS*), where *id* denotes its identifier; *rul* denotes the formalization description of *br*; *MRS* denotes the meta-rule set for revising *br*.

Definition10. Basic Rule Set (BRS) . *BRS* is the set of all the basic rules which *GCG* includes.

Definition11. Dynamic Rule (dr). *dr* is defined as *dr* (*ct, nt, br, rul, w, sta, life*), where $ct \in CTS$, $nt \in NTS$, $br \in BRS$; *rul* is the formalization description of rule; *w* is the value of its weight; and *sta* is its state, and $sta \in$ {*"Naive", "Trainable", "Stable"*}; *"Naive"* denotes that the *dr* is a new rule; *"Trainable"* denotes that the *dr* is revising; "*Stable"* denotes that the *dr* is a mature rule; *life* denotes the value of its life.

Definition12. Static Rule (sr). If *dr* is a dynamic rule and *dr.w>MaxWeight*, which *MaxWeight* is a constant in *GCG,* we can call *dr* as a static rule (*sr*). Its state is *"Static"*.

Definition13. Castoff Rule (cr). If *dr* is a dynamic rule and *dr.w<MinWeight*, which *MinWeight* is a constant in *GCG,* we can call *dr* as a castoff rule (*cr*). Its state is *"Castoff"*.

The state graph of rules is presented in Figure 2.

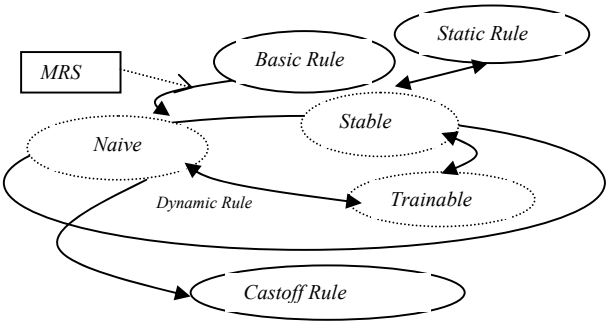

Figure2. State Graph of Rules

The dynamic knowledge is the set of all the dynamic rules in *GCG*. The static knowledge is the set of all static rules in *GCG*. The basic knowledge can be formed by passive learning. For adjusting the dynamic knowledge established according to basic knowledge, we can revise

them through reinforcement learning and multiagent cooperation during the computing process in *DNG*.

Definition14. Task (TSK). *TSK* is defined as *TSK* (*id, DAT, AS*), where *id* denotes the identifier of *TSK*; *DAT* denotes the data set; *AS* is the set of agents which execute *TSK*.

3 Description of learning model

3.1 Initialization of knowledge

Suppose that *TSK* is a new computing task, and *GCG* is the global computing group in *DNG*. The rule initialization works about *TSK* are as follows:

Algorithm1. Initialization of basic rules

① Decide *AS* for *TSK*;
② Establish the static rule set for each computing agent *CA* of *AS* ,and form its *KS*(or *CKS*);
③ While $AS \neq \phi$ do
  Get a computing agent *CA* from *AS*;
  While $KS \neq \phi$ do
    Get a basic rule *br* from *KS*;
    Construct the meta-rule set *MRS* for *br*;
    KS=*KS*-{*br*};
  End do;
  AS=*AS*-{*CA*};
 End do;
④ Commit *TSK* into *GCG*, and *GCG* dispatch a computing device(*CN* or *CC*) for *TSK*, and the computing device start to calculate *TSK*;
⑤ End.

After it is dispatched in *DNG*, *TSK* is executed. For fitting the variety of computing device resources, the dynamic rules must be generated. The generation process is as follows:

Algorithm2. Generation of dynamic rules
Input: *TSK* **(***id, DAT, AS***)**
Output: the *TSK* having dynamic *KS*

① *TmpAS*=*TSK.AS*;
  /* *TmpAS* is the control variable */
② When $TmpAS \neq \phi$ do ③~⑧:
③ Get a *CA* from *TmpAS*;
  $TmpDr = \phi$;
  /* *TmpDr* is a temporary set for new *dr**/
  TmpKS=*CA.KS*; /* *TmpKS* is the control variable */
④ When $TmpKS \neq \phi$ do ⑤~⑥:
⑤ Get a *br* from *TmpKS*;
  Set *dr*=*br*;/* construct a new dynamic rule for *br**/
  Set *dr.w*=*MinWeight;*
  Set *dr.life*=0;/*the value of life is 0*/
  Set *dr.ct* and *dr.nt* by the information of the computing device and *DNG*;
  Revise *dr.rul* through *br.MRS*, *dr.ct* and *dr.nt*;
  Set *dr.sta*= "*Naive*";
  Set *TmpKS*= *TmpKS*-{*br*}; /*Remove *br* */
  Set *TmpDr*= *TmpDr*+ {*dr*}; /*save new *dr**/
⑥ End do;
⑦ *CA.KS*=*CA.KS* + *TmpDr*;/*Add the new *dr* into *CA.KS* */
  TmpAS= *TmpAS*-{*CA*};
⑧ End do;
⑨ Output the new *TSK*.

3.2 Revising for dynamic knowledge

The dynamic knowledge constructed through algorithm2 must be revised during the *TSK* be calculated. The revising mechanism can adopt the reinforcement learning, and the use rate of resource for *TSK* can be as the reinforcement function.

Algorithm3. Revising for Dynamic knowledge based on reinforcement learning.

Suppose that Y_1 is the ***castoff threshold***, and Y_2 is the ***mature threshold*** ; *Q (urt)* is the *reinforcement function* , and *Q (urt)* >0, and *urt* is the use rate of resources; *MaxWeight* is the maximum of the rule weight, and *MinWeight* is the minimum of the rule weight, and let $MinWeight<Y_1<Y_2<MaxWeight$; *MaxLife* is the maximum of life value.

① Suppose that a computing agent *CA* adopted a dynamic rule *dr* of *CA.KS*;
  dr.life + +; /* increase the value of life */
  Wait for the *urt* from the computing system;
② If *urt* >0 then *dr.w*=*dr.w* + *Q*(*urt*);/*increase weight*/
  If *urt* <0 then *dr.w*=*dr.w*—*Q* (*urt*); /*decrease weight*/
③ If *dr.w*>*MaxWeight* then *dr.w*=*MaxWeight*;
  If *dr.w*<*MinWeight* then *dr.w*=*MinWeight*;
④ If *dr.w*<Y_1 and *dr.life*>*MaxLife* then *dr.sta*= "*Castoff*";/*Castoff rule */
⑤ If Y_2< *dr.w* < *MaxWeight* then *dr.sta*= "*Stable*"; /*Stable rule */
⑥ If *dr.w* >= *MaxWeight* then *dr.sta*= "*Static*"; /*Static rule */
⑦ If Y_1<*dr.w*<Y_2 then *dr.sta*= "*Trainable*"; /*Trainable rule */
⑧ If *MinWeight* <*dr.w*<Y_1 then *dr.sta*= *"Naive"* ; /*Naive rule */

⑨ End.

Algorithm3.1. Revising for CCT.

The dynamic cooperation Knowledge of *CCT* usually involves the using of share resources, such as the network bandwidth, so the *urt* value is the average using rate of *CCT* resources. The revising learning process is similar to *Algorithm3*.

When the learning and executing process continues for a long time, the dynamic knowledge is excessive and it will lower the searching efficiency. So, it needs the artificial adjustment for dynamic knowledge to reduce the numbers of the rules. The process is presented as follows:

Algorithm4. Artificial adjustment

Y_3 is the ***adjustment factor***. *CA* is a computing agent.

① *Num* is the numbers of dynamic rules which state is "*Trainable*";

② If $num > Y_3$ then

{Show an interface to adjust;

Let *DelSet* is the rule set that will be deleted;

Show the rules that state are "*Trainable*";

Decide *DelSet* by users;

Delete *DelSet* from *KS* ;}

③ Delete the rules which state are *"Naive"* from *KS*.

3.3 Learning in migration process

In order to persist and improve the global knowledge in *DNG* , the main control agents *Am* in *CN*, *CCT* and *GCG* get the share organization knowledge through the organization learning [15, 16]. Owing to the agent migration in *DNG*, the running environments of *ICA* and *CCT* are in the dynamic change. For fitting the change, the migration organization learning is very important.

Algorithm5. Migration learning of ICA

An *ICA* running on the computing node CN_i must be migrated into the computing node CN_j, so the migration learning process is presented as follows:

① *ICA* commits its *KS* to *Am* in CN_i, and *Am* in CN_i saves the *KS*;

② *ICA* asks for a new computing node CN_j from *GCG*, and it consults with *Am* of CN_j, and migrates into CN_j;

③ *ICA* gets the current knowledge about CN_j and the other agents from *Am* of CN_j;

④ *ICA* and *Am* of CN_j learn each other, and they resolve the conflicts, and they refresh their *KS* and *CE*;

⑤ CN_i and CN_j report their *KS* and CE into *DNG* and *GCG*;

⑥ *ICA* continues for executing and Learning on CN_j;

Algorithm6. Migration learning of CCT

Suppose that *CCT*(*id, Am, CAS, BDI, CKS, CCE*) is a cooperation computing team running on the computer cluster CC_i, and $CAS=MIG \cup NMIG$, where *MIG* is the set of the computing agents that will be migrated; *NIMG* is the set of the non-migration computing agents. The migration process includes three sub-algorithms.

Algorithm.6.1

While $NMIG \neq \phi$ *and* $Am \in NMIG$, the migration learning process is as follows:

① All the computing agents of *MIG* migrate into their new computing nodes severally, and learn the new knowledge according to the algorithm5;

② for all $CA \in MIG$ do ③④⑤:

③ *CA* learns the cooperation knowledge from *Am* of *NMIG*, and consults to resolve the conflicts;

④If the conflict consultation was failure, *CCT* will retract the task *t* from *CA* and expel *CA;*/**MIG=MIG-*{*CA*}*/

⑤If the conflict consultation is successful, then {*CA* refreshes its *KS*; *NMIG=MING+* {*CA*}; *MIG=MIG-*{*CA*}};

⑦ End.

Algorithm.6.2

While $NMIG \neq \phi$ *and* $Am \notin NMIG$, the migration learning process is as follows:

① All the computing agents in *NMIG* cooperate to elect a new main control agent *Anm* from *NIMG*, *Am* submits the cooperation knowledge about *CCT* to *Anm*;

② Do the algorithms.6.1;

Algorithm.6.3

While $NMIG = \phi$, the migration learning process is as follows:

① *Am* migrates into the new computing node according to algorithm5, and notices *GCG*;

② *Am* broadcasts its new conditions to all members of *CAS* of *CCT*; The other members of *CAS* receive the messages from *Am* ,and they do the migration learning by algorithm 5, and they submits the learning results to *Am*;

③ After all the cooperation computing agents finish the migration learning ,*CAS* cooperate to produce the new *CKS* by *Am* control ;

The process of knowledge consultation and organization learning is described in paper [15] [16].

3.4 Learning process of GCG

Algorithm7. GCG learning process

①*GCG* receives a task *TSK*;
②*GCG* calls the *algorithm.1* to do initialization work for *TSK*;
③*CGG* allots a computing device *PE* (that is a *CN* or a *CC*) for *TSK*;
④The computing agents (*ICA* or *CCT*) for *TSK* construct the dynamic knowledge according to the *algorithm2*;
⑤Repeat do ⑥⑦:
⑥All the computing Agents calculate the *TSK*, and they start the *algorithm3* and *algorithm4* to revise the dynamic knowledge;
⑦Until *finished*(*TSK*) or *migrating*(*TSK*);
/* *finished* () and *migrating* () is two boolean functions*/
⑧ If *migrating (TSK)=true* then
{*GCG* searches a new computing device *NPE*;
The computing agents do the migration learning process according to the *algorithm5* and the *algorithm6*;
Goto ④;}
⑨ End.

4 Experiments

We built a *DNG* that is composed of 24 computers and 4 computer-clusters that connected by Intranet. All the computers are classified into 6 types according to their types of cpu, memory, disk, and net adapter. The operating systems of the computers are the Windows series or LINUX. The computing tasks provided by *DNG* are the matrix operations and the linear programming. The *CCT* algorithms (Parallel algorithms based on computer cluster) for the matrix operations and the linear programming are given. The Development tools are the JAVA. The intranet clock is synchronous by GTS protocol.The initialization basic rules include 24 rules for *CA* and 7 rules for *CCT*, and the parameter values are as follows: *MaxLife*=43200(s), Y_1=15, Y_2=80, Y_3=0.3 *MaxWeight*=100 and *MinWeight*=0.The experimentation includes seven times, and each time has 12 hours, and the total amount is 84 hours. The tests adopt a random function to choose some tasks (the matrix operation, the linear programming and their parallel edition) in each time. In order to make the tasks to migrate as far as possible in the *DNG*, We make use of the random migration function *RandMigration()* and form the migration strategy during the test processes. Through the average values of the test information, we observe the learning results by this model. The experiment results are as follows:

Experiment1. We tested the variety of the average use rates of *DNG* resources along with the learning process. The thick solid line means the use rate distribution in the figure 3(a). This test result shows that this model can raise the use rates of *DNG* resources consumedly.

Experiment2. We tested the variety of the numbers of new dynamic rules which are generated during the learning process. The solid line means the distribution of the dynamic rules that their state is always in the *"Naive"* state during their life period in the figure 3(a). The dotted line means the distribution of the *"Trainable"* dynamic rules that their sate has became the "*Stable"* during their life period in the figure 3(a). This test results show that the learning efficiency of this model increases gradualy along with learning process .

Experiment3. We counted the numbers of the artificial adjustments. The solid line means the distribution of the numbers during the tests in the figure 3(b). This test result shows that the numbers of artificial adjustments decreases gradualy along with learning process.

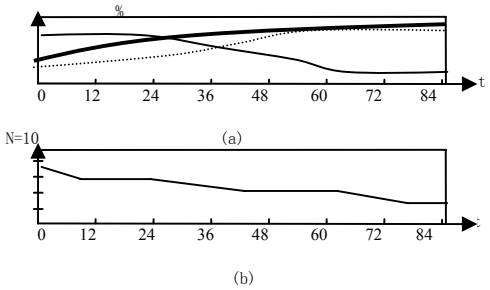

Figure3. The results of tests

5 Conclusions

Because of the heterogeneous resources, the difference of the computing ability of computers and the migration of computing agent, the effective use of resources is very difficult for the grid computing in the dynamic network environment. A good learning model can improve the intelligence of multiagent, and it can raise the use rate of resources. Our learning model and organization learning frame solve these problems.

Acknowledgement

We would like to thank the support of the National Nature Science Foundation of China (No.60573108); the Nature Science Foundation of Shanghai of China (No.04ZR14100).

References

1. E. W. Felten and J. Zahorjan. Issues In the Implementation of a Remote Memory Paging System. Technical Report 91-03-09, University of Washington,1991
2. E. P. Markatos and G. Dramitions. Implementation of Reliable Remote Memory Pager. In proceedings of the 1996 Usenix technical Conference[C], pages 177-190, 1996
3. Anurag Acharya and Sanjeev Setia. Using Idle Memory for Data-Intensive Computations. In proceedings of the 1998 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems[C], Pages 278-279, June 1998.
4. I. Foster, C. Kesselman. The Grid: Blueprint for Future Computing Infrastructure [M]. San Francisco, USA: Morgan Kaufmann Publishers, 1999
5. E. Osawa. A Scheme for Agent Collaboration in Open MultiAgent Environment .Proceeding of IJCAI'93, August 1993, 352~358
6. Wooldridge, M.: An Introduction to Multivalent System, John Wiley & Sons (Chichester, England). ISBN 0 47149691X, February 2002
7. O.F. Rana and D.W. Walker: The Agent Grid: Agent-based resource integration in PSEs, In proceedings of the 16th IMACS World congress on Scientific Computing, Applied mathematics and Simulation, Lausanne, Switzerland, August 2000.
8. Overeinder, B.J., Wijngaards, N.J.E., Steen, M. van, and Brazier, F.M.T. Multi-Agent Support for internet-scale Grid management,http://www.cs.vu.nl/pub/papers/globe/asib-grid.02.pdf.
9. Kaelbling L P, Littman M L, Moore. Reinforcement learning: A survey. Journal of Artificial Intelligence Research, 1996, 4(2):237~285.
10. Sutton R S. Learning to predict by the methods of temporal differences. Machine Learning, 1988,3:9~44.
11. Watkins P. Dayan. Q-learning. Machine Learning, 1992,8(3):279~292.
12. Rummery G. Niranjan M. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.
13. Horiuchi T, Katai O. Q-PSP learning: An exploitation-oriented Q-learning algorithm and its applications. Transactions of the Society of Instrument and Control Engineer, 1999, 39(5):645~653.
14. Berenji H R, Vengerov D. Advantages of cooperation between reinforcement leaning agents in difficult stochastic problems. Proc of the Ninth IEEE Int Conf on Fuzzy System, 2000, 2:871~876.
15. F Zambonelli. N R Jennings, M Wooldridge. Organizational abstractions for the analysis and design of multi-agent systems. Proc of 1st Int'1 Workshop on Agent-Oriented Software Engineering, Lectures Notes in AI, 1957. Limerick, Ireland: Springer-Verlag, 2000:127~141.
16. F Zambonelli, N R Jennings, M Wooldridge. Organizational rules as abstractions for the analysis and design of multi-agent systems. International Journal of Software Engineering and Knowledge Engineering, 2001, 11(3):303~328.