DifreEngine: Distributed Forward Reasoning Engine with General Purpose

Chunyan Han
Software College, Northeastern University, Shenyang, China
College of Information Science and Engineering, Northeastern University, Shenyang, China
Email: hancy@swc.neu.edu.cn

Jianzhong Qiao
College of Information Science and Engineering, Northeastern University, Shenyang, China
Email: qiaojianzhong@ise.neu.edu.cn

Yunxiao Wang, Yixian Liu and Zhiliang Zhu
Software College, Northeastern University, Shenyang, China
Email: neuwyx@gmail.com, liuyx@swc.neu.edu.cn, zzl@mail.neu.edu.cn

Abstract—**In this paper, we propose a distributed forward reasoning engine with general purpose called DifreEngine. We present the architecture of DifreEngine with detailed description of its modules. And then, we describe the working process of DifreEngine. Also we introduce the working mechanism of three important algorithms in DifreEngine called task division, nodes management and task scheduling. The DifreEngine makes efficient general forward reasoning possible.**

Index Terms—**distributed system, reasoning engine, the logistic mapping, task scheduling**

I. INTRODUCTION

During our previous work in the research of anticipatory reasoning reacting system (ARRS for short) [1], we have applied forward reasoning engine with general purpose based on logic (FreeEnCal) [2], which is a computer program that can automatically draw new conclusions by repeatedly applying inference rules to given premises and obtained conclusions until some previously specified conditions are satisfied, as a core mechanism to reason and forecast future events.

However, there was still a problem in the efficiency of standalone reasoning engines like FreeEnCal. Currently, application reasoning engines are usually deployed on a single reasoning engine. This solution is applicable when the reasoning assignment is light weighted and the datasets are relatively small. In a complex application area, such as air traffic control area, it is unreasonable to expect there is any reasoning mechanism that process the reasoning assignment fast enough just through one single processor. Firstly, in a complex area, there will be large corpuses of data found, posing new challenges to the reasoning engines in processing techniques. Secondly, the set of reasoning data, both premises and conclusions, is growing very fast and is dynamic, since throughout the reasoning process, the sensors will automatically add sensory data into the database, meanwhile after reasoning the results will also be filled into the database. Thirdly, rules might be represented in different forms, which require a reasoning task to do preprocessing and coordination with other reasoning engines [3, 4]. Throughout the whole process of reasoning, the theorems derived become much more than those in the original input set and the time that a single engine spent on reasoning lasts longer. What's more, the reasoning capability of single reasoning engine is limited because it has to carry out reasoning linearly. In the face of these new requirements, existing reasoning methods have lost their effectiveness.

After investigating the related works of improving the efficiency of reasoning engines, we mainly summarize two ways to improve the efficiency of FreeEnCal.

On one hand, we can develop new forms of reasoning algorithm to improve the reasoning capability of a single reasoning engine like k-d trees in case-based reasoning area [5]. In this paper, authors use an algorithm based on k-d tree to improve the process of finding similar cases of the case-based reasoning system and the proposed approach was implemented on two reasoning systems for classification. Clearly it is a case to use some algorithm to improve the efficiency of reasoning engines, but it is case -based reasoning systems and it mainly focus on finding similar cases which is different from what we discuss.

On the other hand, we can use parallel reasoning processing to improve the reasoning engine. And there already are many distributed reasoning engines but they are all case-based or knowledge-based reasoning engine, not logic reasoning engine which would be used in ARRS. They always care less about logic axioms and empirical theorems which play an important part in logic reasoning engines. We eventually decide to apply distributed reasoning mechanism to FreeEnCal since the pattern of distributed reasoning enables multi-reasoning at the same time [6, 7]. In this paper we focus on the second way to apply distributed reasoning and we propose a distributed forward reasoning engine with general purpose called DifreEngine.

In this paper, we first propose the architecture of DifreEngine, and then we give a working process description about how DifreEngine works. Finally, we illustrate the working mechanism of DifreEngine with a study case to show its usability.

II. ARCHITECUTRE OF DIFREENGINE

Since DifreEngine is designed as a distributed system, we decide to adopt the classic three layers architecture, as shown in Fig. 1. In this model there are three layers called infrastructure layer, middleware layer and application layer.

The infrastructure layer is a fundamental layer that mostly concerns about the physical topology of the distributed system and network. The middleware layer is logically placed between the application layer and the infrastructure layer, and it focuses on coordinating and communicating between layers of high and low. And the application layer deals with reasoning.

The infrastructure layer is a fundamental layer that mostly concerns about the physical topology of the distributed system and network. The middleware layer is logically placed between the application layer and the infrastructure layer, and it focuses on coordinating and communicating between layers of high and low. And the application layer deals with reasoning.

A. *Infrastructure Layer*

Infrastructure layer is a fundamental physical layer that assures the security and performance of the whole reasoning process. In order to simplify the architecture of the whole reasoning engine system, we decide to build a multicomputer network based on switch like this: firstly, it is a kind of architecture with simple network topology structure, having uncomplicated executing processes; second, it has to be planar so that it's easy to build and cost less; thirdly, since DifreEngine is designed as a distributed engine, it must have good scalability which is easy to add or remove one node. Meanwhile, there need a part which is in charge of task dividing and control other computers to reason. What's more, besides the control computer, DifreEngine needs computers which are dedicated to reasoning process, which means that it must be a centralized architecture.

At present there are mainly two kinds of application architectures: Master-Slave model [8] and Peer-to-Peer model [9]. Peer-to-Peer model is not centralized, that is to say it is not suitable for DifreEngine. Master-Slave model is a model for a communication protocol in which one device or process known as the master controls one or more other devices known as slaves. The relationship of control is always from the master to slaves. It's a centralized architecture. So we adopt Master-Slave model rather than Peer-to-Peer model. The model supports heterogeneous computer network while offering a single system view.

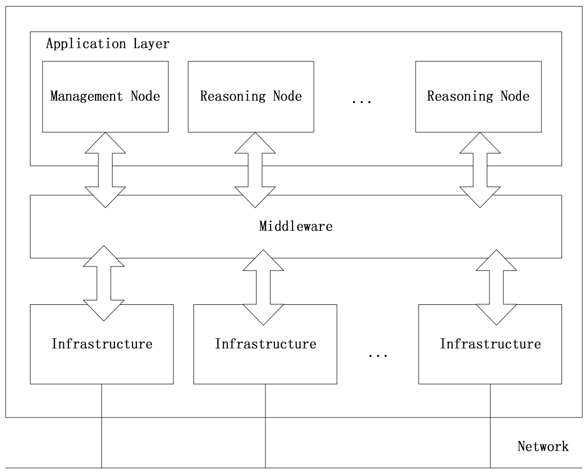

Figure 1. Logical structure of the system

B. *Middleware Layer*

Middleware layer provides a platform to connect and coordinate the application reasoning layer with the underlying layer which is logically placed between them. The function of middleware is to transmit data and to do data verification. We design the middleware layer from two aspects: offering a well-defined API in the form of document which informs the programmers all the operations the module can perform, and leveraging existing tools, such as RPC and CRC [10, 11].

We use RPC to implement distributed communication. Through RPC, master node can communicate with the remote slave nodes. RPC avoids the details of network interfaces, heterogeneous machines and precision differences, by which program can call remote procedure just as the local methods do.

Data Validation is one of the main functions of Middleware Layer. Compared with CRC, parity check can only check error, it cannot confirm that there is no mistake and it's unable to verify the location of the error so that it cannot correct the wrong code. Therefore considering the performance and the cost, CRC is far superior to parity check, since CRC is simple to implement in binary hardware, having faster computing speed, easy to analyze mathematically, and particularly good at detecting common errors caused by noise in transmission channels.

C. *Application Layer*

There are two kinds of nodes in application layer, the Management node and Reasoning nodes. Management node which is deployed in the master node is mainly in charge of choosing a logic system and inference rules, task division and results collection and integration. It has Logic Process Module, Task Division Module, Task Scheduling Module, Interaction Module, Task Integration Module, Duplication Checking Module and Nodes Management Module. While reasoning nodes which are deployed on slave nodes concentrate on reasoning according to premises and facts which they get from management node. It has Interaction Module and Reasoning Module. The detailed description of every module will be displayed in the working process.

D. The Working Process

After introducing the structure of DifreEngine, we describe the working process of DifreEngine in Fig. 2.

The whole reasoning process can be described as follows:

(1)First, in a Management node, there are some candidate logic systems like CML, RL, SRL, STDRL and so on. And there are also many empirical theorems about some fields like Air Traffic Control, Highway Traffic Control and so on. When a user wants to use DifreEngine to reason a task, the user must input facts and choose a certain logic system, and he also can input extra empirical theorems if necessary.

(2)Then the Logic Process Module would check if the logic system is suitable. If it's suitable, then go to the next step. If not, it would go back to step (1).

(3)And then the task is passed to Task Division Module. In this module, the task would be divided into several subtasks according to a task division algorithm. Then Task Scheduling Module would tell Interaction Module to assign each subtask to some certain reasoning nodes. After that Interaction Module gets subtasks and passes them to Reasoning nodes through the middleware layer.

(4)In a Reasoning node, Interaction Module gets the subtask from the middleware layer, and passes the subtask to Reasoning Module. The Reasoning Module would start to reason and generate results. In this process, the Reasoning node is being monitored by the Nodes Management Module of the Management Node. After deducing results, the Reasoning node would pass these results back to the Management node through Interaction Module and the middleware layer.

(5)After the Interaction Module pass these results to the Task Integration Module, the Task Integration Module would integrate these results to a results set. And then the Duplication Checking Module would eliminate the duplicate results.

(6)Finally, the Results Saving Module would save these results and present them to the user.

III. KEY ALGORITHMS AND STRATEGIES

A. Task Division Algorithm

In Task Division Module, task division algorithm is the most important part. We propose a task division algorithm according to the graph theory. We let every empirical theorem be a point. And then recurs all the empirical theorems, if the conclusion of one empirical theorem appears in the premise part of another empirical theorem, draw a directed line from the former to the latter. And the line is directed to the latter. After the recursion, we get a graph with points and directed lines. And then start from the point which in-degree is 0 and go through the directed line and points of an edge. We regard all the empirical theorems in an edge as a class. As is shown in Fig. 3, E1 and E2 are in a class. E1, E4 and E6 are in a class. There are 6 classes in total. We can send a class of empirical theorems to a certain reasoning node.

B. Nodes Management Strategy

Nodes management strategy describes how management node which is deployed as the master node manages and monitors the states of all the reasoning nodes in real time, including the joining, failures and register of the reasoning nodes, also the load of each reasoning node.

In DifreEngine, reasoning nodes may join or leave. When a new node joins the system, it has to register to obtain authorization, and then get the node ID. In order to deal with the node fault well, we demand that each reasoning node should send periodic message to the master node automatically. Once exceeding the period, if the master node doesn't receive the periodic message of some node, we tacitly approve that this node is out of work and mark the node as invalid.

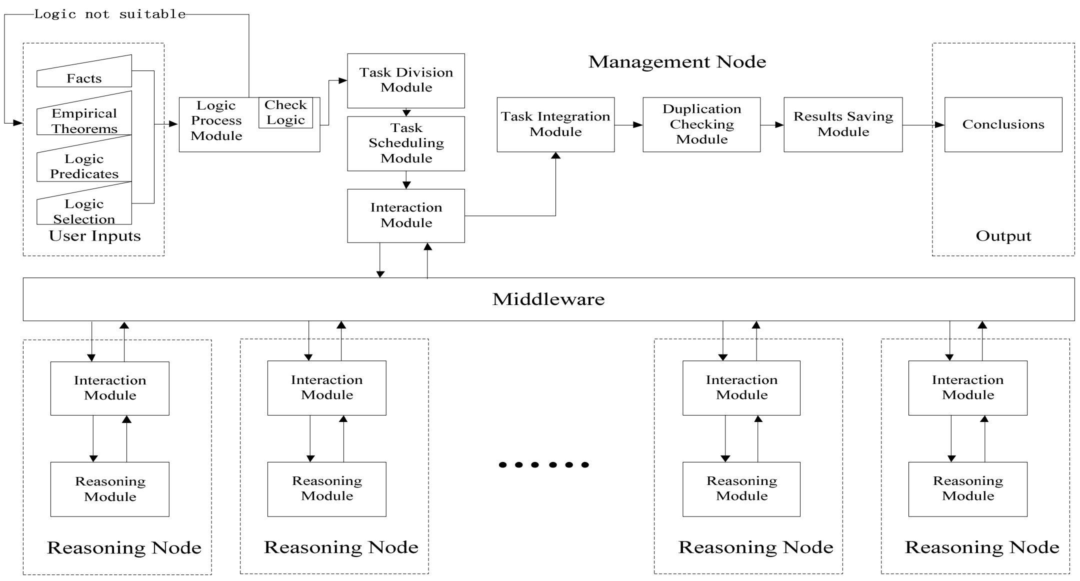

Figure 2. The reasoning process

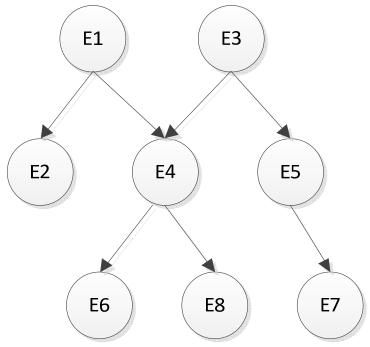

Figure 3. Example of the task dividing results

All the tasks of this node are set to be idle condition, then, these tasks can be allocated to other nodes, waiting for rescheduling by Task Scheduling Module.

The master node is the center of this architecture. Once it crashes, the whole system cannot work. Thus, we adopt a solution called dual-computer hot standby [12]. In this architecture, we have two hosts, one called primary host, i.e. the master node, another called standby host. Under normal situation, primary host is active, and standby host is in hot standby. These two hosts are synchronized in real time. They can talk to each other, and the latest information is transferred from the active host to the hot standby host in real time. Upon failure of the primary host, the standby host becomes active immediately and takes over the jobs of primary host automatically.

C. *Task Scheduling Algorithm*

At the first stage of research, we assume that every sub-task needs the same amount of reasoning time. And we decide to adopt a random scheduling method.

We chose 3 random methods as candidates. They are random method of Java, method for parallel inference tasks and an improved random method with logistic map [13, 14]. Firstly we did some experiments to compare their randomness. We use these three methods to assign 10000 subtasks to 100 nodes to reason. We calculated the variance of each node to decide which one is better. After 1000 times experiments, we concluded that from aspect of randomness, method for parallel inference tasks is way far good from the other 2 methods. And the random method of Java is a little better than logistic method. We firstly eliminated method for parallel inference tasks. As for the other 2 methods, based on the random method in Java, we combined logistic mapping with random method.

In general, we replaced the time seed of random method with sequence of numbers generated by logistic map. In this logistic map, we let X_0 be 0.7 and the coefficient r be 3.935. We make this logistic map to recurs 10000 times and store the sequence of numbers into memory. Considering that the initial part of the sequence is not so chaotic, we start from the 2000th number. The following process is the same as the random method of Java.

We use two kinds of methods to assign 10000 subtasks to 100 nodes to reason. And then the average subtasks of each node should be 100. According to the results, we calculated the variance of each node. We did 1000 times experiments in a round. As is shown in Fig. 4, after 1000 rounds experiments, we found that the situations, in which the sum of the variance of Java random method is bigger than the logistic appeared 501.4 times in average. And the opposite situations appeared 498.6 times in average. In other words, the situations in which the random method of Java performed better appeared 498.6 times in average, the improved random method performed better appeared 501.4 times in average. We can conclude that the improved method and the random method of Java had the same randomness.

As for efficiency, we did another round of experiments. But the time of generating 100000 random integer numbers by Java Random method is around 7006707.13 nanoseconds in average, while the new logistic method is around 4074179.72 nanoseconds in average. We can tell from the Fig. 5 that the efficiency of improved random method is better.

According to amount of results, we conclude that the performance of the new hybrid method is as good as random method in Java, while the time-cost is nearly the half amount. Eventually, we adopt our new hybrid method as our task scheduling principle.

From another point of view, we consider that the execution time of each sub-task differs from each other. Thus, we can measure the time complexity through degree. Degree of a logic connector is nest of the connector in a logical formula. For example, if A and B does not include logic connector, degree(A)=0; degree(A→B)=1; degree(A→(A→B))=2; degree((A→B)→(A→B))=2; degree(A→(A→(A→B))) =3. Let's just say that the higher the degree is, the more complex the sub-task is. Thus, the completion time of the sub-task is longer.

IV. Illustration

Here is a scenario about air traffic control field. We use this scenario to illustrate the working process. We assume a scene which contains three planes. In this scenario, we want to predict danger and make the planes to react according to the prediction in order to avoid danger. We will use spatio-temporal deontic epistemic relevant logic system. Here we assume that a plane is flying at a constant speed.

Scenario: As is shown in Fig. 6, we assume that in a flying area, at some certain time, there are three planes A, B and C flying in the same area. Plane A and plane B are in the same airline and flying in the opposite direction. Meanwhile, Plane C is moving in the same direction behind plane B. And plane C is no faster than plane B.

Defined predicates:

ccording to the scenario, we define predicates as follows:

SameAirline(i, j) means "plane i and plane j are flying in the same airline",

OppositeDirection(i, j) means "plane i and plane j are moving in the opposite direction",

SameDirection(i, j) means "plane i and plane j are moving in the same direction",

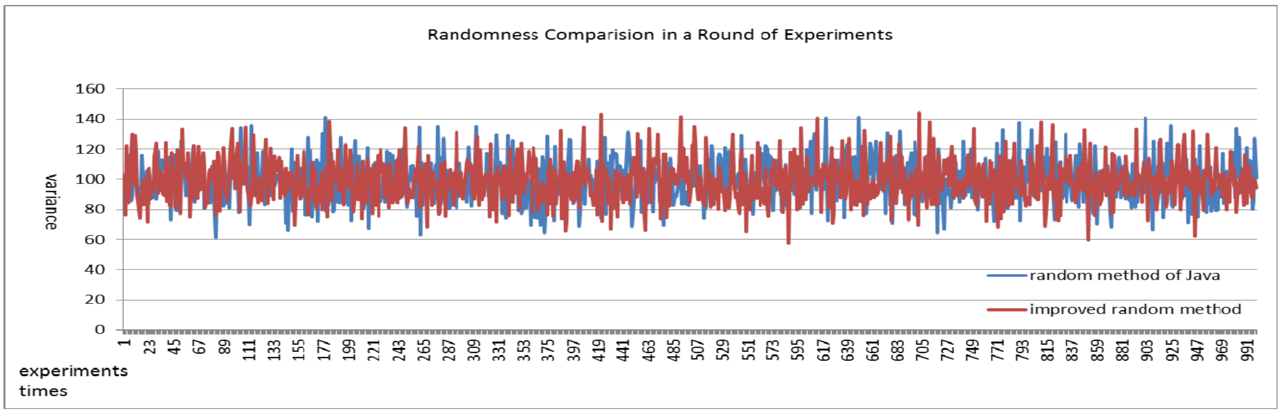

Figure 4. Randomness comparison

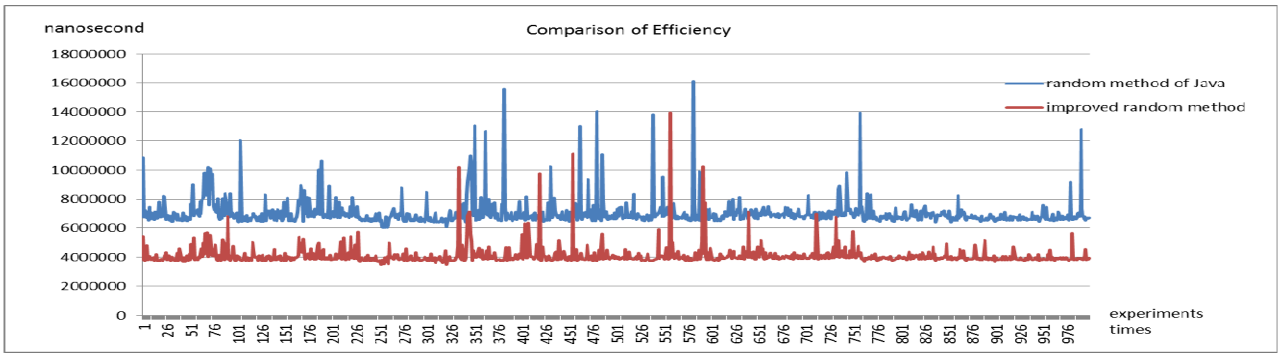

Figure 5. Efficiency comparison

Faster(i, j) means “plane i is moving faster than plane j”,

Danger(i) means “plane i is in danger”,

FlyToRight(i) means “plane i flies to its own right”.

Maintain_state(i) means “plane i maintains the original state”.

SafeDistance(i, j) means “the distance between plane i and plane j is a safe distance”

FlyBehind(i, j) means “plane i is flying behind plane j”

Empirical theorems:

According to the scenario and common sense, we work out empirical theorems as follows:

①$\forall i \forall j K_i((\mathrm{SameAirline}(i, j)) \wedge \mathrm{OppositeDirection}(i, j)) \Rightarrow K_i(F(\mathrm{Danger}(i) \wedge \mathrm{Danger}(j)))$

②$\forall i\ K_i(F(\mathrm{Danger}(i)) \Rightarrow O(\mathrm{FlyToRight}\ (i))$

③$\forall i \forall j K_i((\mathrm{SameDirection}(i, j)) \wedge (\mathrm{SameAirline}(i, j)) \wedge \mathrm{FlyBehind}(i, j) \wedge \neg \mathrm{Faster}(i, j) \wedge \mathrm{SafeDistance}(i, j)) \Rightarrow K_i(\neg F(\mathrm{Danger}(i)))$

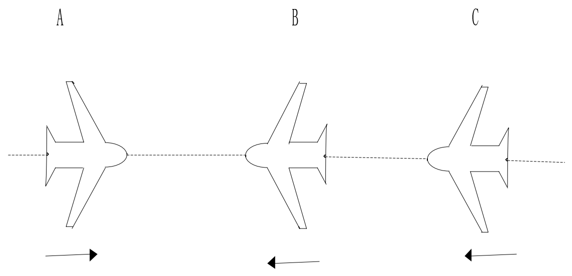

Figure 6. Current situation of three planes

④$\forall\ i\ K_i\ (\neg F(\mathrm{Danger}(i))) \Rightarrow \mathrm{Maintain_state}(i)$

On above, K is an epistemic operator, F is a temporal operator. $K_a p$ stands for “a knows that p”, and FA means “it will be the case at least once in the future from now that A”.

Facts:

From the scenario, we extract 9 facts as follows:

①SameAirline(PlaneA, PlaneB),

②OppositeDirection(PlaneA, PlaneB),

③SameDirection(PlaneB, PlaneC),

④¬Faster(PlaneB, PlaneC),

⑤SafeDistance(PlaneB, PlaneC),

⑥FlyBehind(PlaneC, PlaneB)

Detailed illustration:

(1) The user input empirical theorems ①, ②, ③, ④all the facts and chose the Spatio-Temporal RL logic system.

(2) Logic Process Module checked the input of the user is suitable to spatio-temporal deontic epistemic RL logic system. The conclusion of empirical theorem ① appears in empirical theorem ② and the premise of empirical theorem ④ contains the conclusion of empirical theorems ③ According to the task division algorithm, Task Division Module divided 4 theorems and facts into 2 classes, class1 and class 2. Class 1 contained theorem ①, ② and fact ①, ②. Class 2 contained theorem ③, ④ and fact ③, ④, ⑤, ⑥.

(3) After task division, according to the task scheduling algorithm, Task Scheduling Module assigned class 1 to Reasoning Node No.1, class 2 to Reasoning Node No.2.
(4) After receiving subtask, two nodes started to reason. In Reasoning Node No.1, it deduced two results: $K_{planeA}(F(Danger(planeA) \wedge Danger(planeB)))$ and O(FlyToRight(planeA)) with 2 unit time. While Reasoning Node No.2 deduced $K_{planeC}(\neg F(Danger(planeC)))$ and Maintain_state(planeC) with 2 unit time.
(5) Task Integration Module integrated all 4 results and Duplication Checking Module found no duplication within them.

Finally, Results Saving Module saved 4 results and presented them to the user.

V. Concluding Remarks

We presented a distributed forward reasoning engine with general purpose called DifreEngine. And then we described its architecture. After that we gave a working process of DifreEngine to show how it worked. We solved the task division problem by using a dividing method according to logic predicate. And we used many strategies to let node management manage reasoning nodes. In task scheduling module, we proposed a random number method according to logistic mapping. At last, through an illustration we described the usability of DifreEngine and concluded DifreEngine had 3 advantages: high efficiency, robustness and low cost.

In the next step, we are about to implement the whole system, especially focuses on the middleware layer and the application layer. Our main job will be: first, finish the details of the middleware layer including the applying of middleware and its modification; second, implement the modules in the application reasoning layer.

Acknowledgment

The authors would like to thank the editors and anonymous reviewers for their valuable comments and suggestions. This research work is supported by Technology Project of Liaoning Province (2011216027).

References

[1] J. Cheng. “Anticipatory Reasoning-Reacting Systems”, in Proc. International Conference on Systems, Development and Self-organization, 2002, pp.161-165.
[2] Yuichi Goto, Takahiro Koh, Jingde Cheng. “A General Forward Reasoning Algorithm for Various Logic Systems with Different Formalizations”, Knowledge-Based Intelligent Information and Engineering Systems, 12th International Conference, KES 2008, pp. 526-535, Sep 2008.
[3] Fensel, D. van Harmelen, F. “Unifying Reasoning and Search to Web Scale”, IEEE Internet Computing 11(2), 96, 94–95, 2007.
[4] Fensel, D. van Harmelen, F. Andersson, B., Brennan, P., Cunningham et al. “A Platform for Web-scale Reasoning”, in Proceedings of the International Conference on Semantic Computing, pp.524–529, 2008.
[5] Herranz Javier, Nin Jordi, Solé Marc. “Kd-trees and the Real Disclosure Risks of Large Statistical Databases”, Information Fusion, v 13, n 4, pp.260-273, Oct 2012.
[6] Li Peiqiang, Zeng Yi, Kotoulas Spyros. “The Quest for Parallel Reasoning on the Semantic Web”, 5th International Conference on Active Media Technology. Proceedings: Active media technology. Lecture Notes in Computer Science, volume 5820, pp.430-441, Oct 2009.
[7] Yuh-jen Chen, Yuh-min Chen, Yung-sheng Su, Chiun-cheng Wen. “Ontology-based Distributed Case-based Reasoning in Virtual Enterprises”, International Journal of Software Engineering and Knowledge Engineering, v 19, n 8, pp.1039-82, Dec 2009.
[8] Atila Madureira Bueno, Andre Alves Ferreira, Jose Roberto Castilho Piqueira. “Modeling and Filtering Double-frequency Jitter in One-way Master-slave Chain Networks”, IEEE Transactions on Circuits and Systems Part I: Regular Papers. Dec 2010.
[9] Adjiman, P., Chatalic, P., Goasdoue, F., Rousset, M.-C., Simon, L. “Distributed Reasoning in a Peer-to-peer Setting: Application to the Semantic Web”, Journal of Artificial Intelligence Research, v 25, p 47, 2006.
[10] Sang-Hoon Kim, Youngjae Lee, Jin-Soo Kim. “FlexRPC: A Flexible Remote Procedure Call Facility for Modern Cluster File Systems”, IEEE 2007.
[11] Yan Sun, Min Sik Kim. “A Pipelined CRC Calculation Using Lookup Tables”, Proceedings of the 7th IEEE conference on Consumer communications and networking conference, Jan 2010
[12] Chongquan Zhong, Li Zhang, Hongyu Li and Li Tian. “Research and Implementation of Dual-Server Hot-Standby of Configuration Software”, Proceedings of the 6th World Congress on Intelligent Control and Automation, June 2006.
[13] Wei Song. “Research on Logistic Mapping and Synchronization”, Intelligent Control and Automation, 2006. WCICA 2006. Volume1, pp.987 – 991, 2006.
[14] Shih-Liang Chen, Tingting Hwang, Wen-Wei Lin. “Randomness Enhancement for a Digitalized Modified-Logistic Map Based Pseudo Random Number Generator”, 2010 International Symposium on VLSI Design, Automation and Test (VLSI-DAT 2010), pp.164-167, 2010.

Chunyan Han was born in Liaoning province of China in 1973. She earned the bachelor degree in Computational Mathematics and Application Software in 1996 from Fudan University and the master degree in Computer Architecture in 2002 from Jilin University, China. Her main research interests include information system engineering and distributed systems.

She is currently a PHD candidate of the college of Information Science and Engineering of Northeastern University. She has published more than 10 research papers from 2009.

Jianzhong Qiao was born in Xingcheng of Liaoning province of China in 1964. He got his bachelor degree in Computer Software from Xian Jiaotong University, Xian, China in 1986, master degree in Computer Software from Shenyang Institute of Computing Technology of Chinese

Academy of Sciences, Shenyang, China in 1991 and PHD in Artificial Intelligence from Dalian University of Technology, Dalian, China. His current research interests include distributed computing, operating system, software architecture.

He is a professor in college of Information Science and Engineering, Northeastern University.

Yunxiao Wang was born in Dalian of Liaoning province of China in 1988. He began the study of Software Engineering in Software College of Northeastern University, Shenyang, China from 2007 to 2011. Then he got his bachelor degree in Software Engineering in 2011. In July 2011, he continued to study in Software Engineering in Northeastern University as a master degree candidate.

Now he is a student in Information System Engineering lab of Software College of Northeastern University.

Yixian Liu was born on July 9th, 1982 in Anshan, China. He got his bachelor degree of Engineering in Computer Science and Technology from Wuhan Universiry, China in June 2004 and received his master degree of Science in Computer Science from New University of Lisbon, Portugal and Free University of Bozen-Bolzano, Italy in 2007.

From December, 2008, he got a professional position at Software College of Northeastern Univerisy, China. His research interests include computational logics, software engineering, service science and cloud computing.

Zhiliang Zhu was born in 1962. He earned his PHD degree in Computer Science from Northeastern University, China in 2002. His main research interests include chaos-based digital communications, complex-network theories, network communication security, information integrate, complexity software system, etc.

He is the professor and PHD supervisor of Software College, Northeastern University, Shenyang, China. As a person in charge or a principal participant, he has finished more than 10 national, provincial and ministerial level research subjects of China.

Prof. Zhu is a senior member of quite a few professional academic committees, for instance, China Computer Federation (CCF), the Chinese Institute of Electronics, etc.