Advances in Intelligent Systems Research, volume 133
2nd International Conference on Artificial Intelligence and Industrial Engineering (AIIE2016)

A General Conceptual Model of Recommendation Based on Multi-Agent Approach

Xunhui Zhang, Tao Wang, Gang Yin, Xinjun Mao, Huaimin Wang
National University of Defense Technology, China

***Abstract*—In recent years, recommendation is becoming more and more popular in many fields. However, there are still some problems, such as single point failure, lack of ability to support different devices and etc. To solve these problems, researches have been carried out using multi-agent approach but some new problems occur. In this paper, we draw an overview of the multi-agent recommendation systems according to the recent study in this field. Through our investigation, we summarize the challenges that multi-agent recommendation systems face and come up with a general conceptual model called GMMR. To some extent, our work may give researchers a comprehensive understanding of the significance of multi-agent recommendation systems and provide a useful model for developers.**

Keywords-multi-agent recommendation; RSs; GMMR

I. INTRODUCTION

Recommendation systems (RSs), also called recommender systems are computer applications for recommending specific items according to users' needs. And they have been used in many areas, such as websites, books, movies and e-commerce [1]. Many approaches occur during the development of RSs, including collaborative filtering, content-based filtering, knowledge-based recommendation, hybrid recommendations and so on [2]. Amazon uses item-to-item collaborative filtering method to recommend books [3]. YouTobe uses the combination of content-based filtering and knowledge-based recommendation to recommend videos to users [4]. Google generates its new recommendation algorithm using three approaches: collaborative filtering, probabilistic latent semantic indexing and covisitation counts [5]. However, there are some shortages of traditional RSs.

- Single point failure: traditional RSs tend to use the C/S model to do the recommendation job. If the server breaks down, the whole system stops.
- Support of different devices: traditional RSs do not consider much about the differences between different devices.
- Combination of different algorithms: most traditional RSs just simply use single algorithm to do the recommendation jobs, whose result is not accurate enough sometimes.
- Recommendation speed is slow: the performance of traditional RSs is not fast enough to meet users' needs sometimes.

To handle these problems, many researchers suggest using agent technology in RSs. Also there are many RSs focusing on specific domains. Even though these systems solve the problems that we mentioned above, some new problems occur.

- There isn't a general model which can handle recommendation jobs in different domains.
- Lots of effort is needed when adding a new agent into the system.
- These systems can not protect themselves effectively and recover from breaking down.
- The speed of recommendation is still not fast enough to meet users' requirements.

From these problems, we conclude four research questions.

Q1 How to focus on different fields in one model?

Q2 How to improve the scalability of the system?

Q3 How to improve the ability of self-protection and self-healing?

Q4 How to make recommendation faster?

In this paper, we will study some existing frameworks of multi-agent systems, and summarize the ideas of each researcher. Then we come up with a general conceptual model to help solve these research questions.

II. RELATED WORK

The development of Internet and smart devices has led to a rapid change of software systems. Almost all the people want the applications we use to be smart and autonomous, which can bring us more convenience and less effort, so do RSs. According to the shortages of traditional RSs we mentioned above, many researches have been carried out using multi-agent approach.

A. Systems that Solve the Single Point Failure Problem

TRUST! is a distributed multi-agent system for information recommendation created by Haiming Lu [6]. This system has many service agents which focus on the same function. If one agent is down, another agent with the same function can take its place. In this way, the whole system can avoid breaking down from single point failure. Nevertheless, the system is still lack of self-healing and self-protection ability. When all the agents with the same function break down, the system will stop working.

B. Systems that Support Different Devices

Domenico Rosaci and Giuseppe created ARSEC, a multi-agent recommender in e-commerce [7]. The architecture of ARSEC has a special part called the customer agent. This kind of agent can collect information from different users who use different kinds of devices. Because agents themselves have differences, a new agent will be added when a new kind of device joins the system, which can help to support different devices. Although this system helps solve the problem using multi-agent approach, the scalability of the system is still not that good. When we want to add a new agent into the system, we need to compare the differences with other customer agents and then set the new agent with some special features.

C. Systems that Combine Different Algorithms Together

Kurapati created a multi-agent TV recommender in 2007 [8]. This system has two different parts, namely implicit and explicit recommender agent. They focus on different methods of recommendation. This mechanism makes the result more accurate than those with one algorithm only. That is to say, multi-agent approach can combine different algorithms together which makes it easily to be applied to other fields.

D. Systems that Use Multi-Agent Approach

Gil created an e-learning recommendation system in 2008 [9], which used the multi-agent approach. After experiment, they found that the system can retrieve and recommend documents faster when testing with ACM CR categories. However, in the end-user satisfaction measures, the timeliness of the system is still not good enough to meet users' satisfaction. That is to say, this multi-agent recommendation system is still slow, which need more improvements. The same problem also exists in some other multi-agent RSs including Implicit [10], Turist@ [11], and APRS [12].

In conclusion, the multi-agent approach does solve some problems that traditional RSs have. But there are still some properties need to be improved.

III. General Conceptual Model

In order to solve the problems that we mentioned above and combine different RSs together, we come up with a generic model called GMMR.

A. Architecture

Inspired by some famous architecture of multi-agent systems and the theory of RSs [13,14,15]. Also in order to adapt to the requirements mentioned above. We come up with a general model called GMMR (general conceptual model of multi-agent recommendation systems), whose architecture is shown in figure I.

In this model, there are five major parts: user agents, meta agents, server agents, client agents, and blank agents. Each part is constituted by the same kind of physical object. That is to say, all the agents in the system have the same structure. But when we initialize the system, different functions are distributed to different agents.

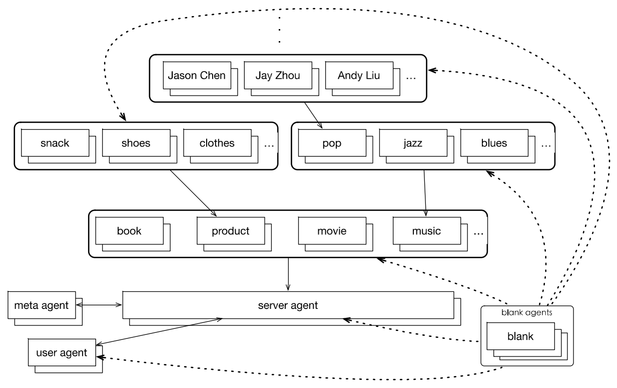

FIGURE I. GMMR ARCHITECTURE

1) User agent

This agent is used to handle user inputs and present results. There are many kinds of user input forms, such as speech, handwriting, keyboard input, mouse input and etc. The first thing of the system is to change different input forms into computer text. Each agent is used to handle a kind of input. Also it will store information of each user when they login.

After the process of recommendation, the related user agent will be used to show the recommendation result. That is to say, it is also used to transform computer text into different kinds of expression.

2) Meta agent

Meta agent is a kind of auxiliary agent which can help its server agent translate the request into some key words. That is to say, it can extract important information from computer text. For example, a user wants to have a look at the music recommendation result. The corresponding computer text can be "I want to listen to Jason Chen's music". Then the meta agent can extract the key words including, Jason Chen and music.

Another use of meta agent is to decide the way to present the final recommendation result. This is related to the user's habit and the user input form.

3) Server agent

This kind of agent acts as the job dispatcher in this model. It is in connection with all the agents in its branch and other server agents.

- Connection with user agents: after transforming the user input into computer text, user agents will transfer data to their server agent first. After handling the problem, the server agent will gather all the information and return the result to related users.
- Connection with meta agents: after getting data from related user agents, the server agent will firstly judge whether it can be used directly. If not, the server agent will call the meta agent to extract key words.
- Connection with client agents: judging from the key words of user request. The server agent will find related client agents to handle the task.

4) Client agent

This kind of agent record the relationship between different levels and do the specific recommendation job. We regard the server agent as the first level agent. In figure Ⅰ, we can see that there are four different agents in the second level, namely book agent, product agent, movie agent and music agent. In the third level, there are also many agents, among which the pop agent, jazz agent and blues agent are all related to the music agent. Different levels of agents handle different size of tasks. The recommendation result of an agent equals to the combined results of all its son agents.

The agent structure is shown in figure II.

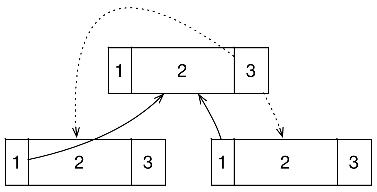

FIGURE II. AGENT STRUCTURE

Each agent can be divided into three parts. The first part records its father agent. If it is a server agent, which means it has no father agent, the first part will be “null”. The second part is the main part of each agent, which records its basic information (capability, name and id), running algorithm and related data. The third part records all the son agents in the next level. If it is a leaf agent, this part will be set to “null”. Part one and part three will change when adding or deleting an agent.

5) Blank agent

There is a large number of this kind of agent in the system. But it doesn’t work at ordinary times. Only when one of the running agents is down, a blank agent will take the place of that agent and copy all the information at the same time. The former agent will erase all the data and turn into a blank agent.

From figure I, we can see that all the agents are in pairs. The only difference is their addresses. The structure is shown in figure III.

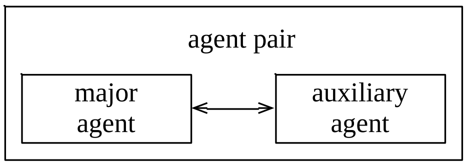

FIGURE III. STRUCTURE OF AGENT PAIR

In each pair of agents, there is a major agent and an auxiliary agent. The major agent is used to communicate with other agents. Also it can assign tasks to its related auxiliary agent. When there is a running task on the major agent, it will assign some tasks in the waiting queue to the auxiliary agent. When the task is very complex, the major agent will divide the task into two subtasks and assign some of the subtasks to the auxiliary agent. If the auxiliary agent breaks down, a blank agent will just take its place. But if the major agent breaks down, the auxiliary agent will take the responsibility to communicate with other agents, and the blank agent will turn into an auxiliary agent.

For the communication of agents, we can use the existing languages and protocols such as KQML [16], FIPA-ACL [17], ICL [13]. Or you can create your own language and protocol.

B. Process Flow

From the architecture we mentioned above, we present the elements of GMMR and their abilities. Now we will give you an example of music recommendation so that you can have a better understanding of the process flow of our model.

The user speaks into the microphone a sentence, “I want to listen to some pop music”. The system will present the user some music that he may like.

The process flow includes four main steps:

1) User agents transform user input into computer text.

The user agent which handles speech is monitoring user’s input. After the voice message comes, it will use the algorithm predefined in the the agent and transform the speech into computer text. Then the agent will transfer the text to related server agent.

2) Meta agent extracts key words from the text.

Server agent firstly judges whether the computer text is in the form of key words. In this example, the server agent gets the text “I want to listen to some pop music”. Obviously, there are not only key words, but also some other elements including subject, predicate and so on. Therefore, the server agent will transfer the message to the meta agent so that it can help the server extract key words. In this example, the key words are “music” and “pop”. After this process, the meta agent will return the key words to server agent.

3) Server agent dispatches the job to different client agents level by level.

After getting the key words, the server will dispatch tasks to different agents according to the key words. key words have their priorities. In this example, “music” is in the first place and “pop” is the next. The server agent will firstly find whether there is an agent named music in the second level. If so, it will dispatch the task to that agent and the music agent will find the pop agent in its subtree. If not, it will dispatch the task to all the agents in the second level, and the agents in this level will continually dispatch the task until they finally find the music agent or reach the leaf agent. All the leaf agents and the pop agent become the processing agent group. The process is shown in figure IV.

4) Server agent gets the final result and present it to the user.

After finding the processing agent, the related agents will do the recommendation task according to the information obtained from the user agent, and transfer the result to their father agents. The father agent will gather all the results from the son agents and transfer them to the upper level agent. This process will repeat until all the results reach the server agent.

Finally, the server agent integrates the results into the final result. After deciding which way to present the result by the meta agent, the server agent transfers the result to the related user agent.

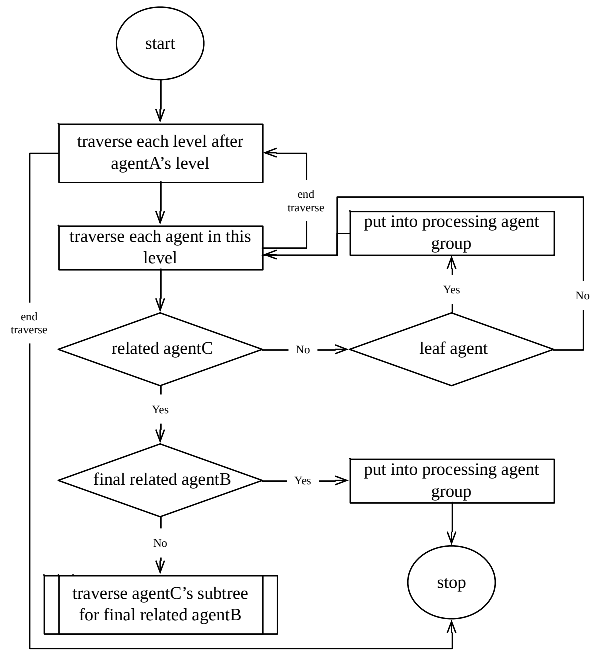
start
traverse each level after agentA's level
end traverse
traverse each agent in this level
put into processing agent group
Yes
No
end traverse
related agentC
No
leaf agent
Yes
final related agentB
Yes
put into processing agent group
No
stop
traverse agentC's subtree for final related agentB

FIGURE IV. THE PROCESS OF TRAVERSING AGENT'S SUBTREE FOR FINAL RELATED AGENT

C. *Advantages of GMMR*

In this part, we will present the advantages of GMMR. It solves the problems that multi-agent RSs have nowadays, which we have mentioned before. Next we will explain how our model deal with each research question.

Q1: Even though agents have the same structure, different functions and data will be added into different agents after initialization. Because agents have the ability of working independently, different agents can focus on part of a specific domain. That is to say, each subtree of the server agent focus on a specific domain. In this way, this model can do the recommendation task in many fields.

Q2: Each agent is in a different level of the system, when a new agent is added, we can easily find its place according to its functions. Also, each agent has three important parts. One is the pointer to its father agent, another part is a field which stores all its son agents. In this case, when a new agent come, we can easily modify different fields and find the precise location. That is to say, GMMR has a high level of scalability.

Q3: Blank agents and agent pair structure in GMMR can help improve the system's ability of self-protection and self-healing. When a running agent is down, the blank agent can take its place, and the down agent will turn into a blank agent. Meanwhile, each agent in the system has a running copy which can reduce the whole system's probability of breaking down.

Q4: GMMR uses the level architecture. The deeper the level is, the more specific it will focus on. According to this architecture, the model will process it in the deepest agents when a task comes. These agents contain the least amount of related data. Hence, in this respect, GMMR can accelerate the recommendation speed.

IV. REVIEW AND CONCLUSION

With the rapid development speed of recommendation, we can see that traditional RSs can hardly meet our needs. Because of the independency, intelligence, autonomy of agent and the development of multi-agent systems, researchers tend to apply the multi-agent approach into RSs.

Many applications and practices indicate that multi-agent approach can solve the problems of traditional RSs, but new requirements occur. We need to come up with new ways to solve the problems including focusing on different fields, scalability of the system, the ability of self-protection and self-healing, much faster speed.

According to these problems, we did a lot of research work and summarized many ideas of former researchers. Finally, we presented a general conceptual model called GMMR. Through agent's independency, it can focus on many fields of recommendation without any restriction. Also, the model has a good scalability because of the agent's structure. What more, the blank agents and the agent pairs help improve the ability of self-protection and self-healing. Finally, the level structure makes the recommendation process faster.

To some extend, our work presents a concept of a new recommendation model based on multi-agent approach, which may give researchers in this field a little inspiration.

V. LIMITATIONS AND FUTURE WORK

Our model can solve the problems mentioned above to some degree. But some new problems appear:

A. *Data Redundancy.*

When we add a new agent into the system, data transformation is needed. If the data is easy to separate, we just move it from the father agent to the new agent. However, if the data is so complex that we cannot separate easily, the system will make a copy of the data. Therefore, there may exists some duplicated data in the system.

B. *Cost of Communication*

Our model has no limitation to the number of levels of agents. When the branch is very deep, the communication between agents lead to a big cost, which may be even more than the cost of computing on agents.

C. *Time Cost*

The structure of the agent may increase time cost of finding the target agent, although it can help reduce the time cost of finding the target agent in the server agent and avoid the system from becoming centralized to some extent.

Meanwhile, there are some limitations in our work: Firstly, we just come up with the idea of the general model, but have not realized a system based on our model. Secondly, we haven't considered much about the communication among agents, which is an important part in the model.

In the future, we will mainly focus on the shortages and limitations of this work and make our model better.

Acknowledgment

The research is supported by the National Natural Science Foundation of China (Grant No.61432020,61472430,61502512) and National Grand R&D Plan (Grant No. 2016-YFB1000805). We would like to thank Yarong Zeng and Liqian Li for their collaboration. We would also thank the anonymous reviewers for providing us constructive comments and suggestions.

References

[1] Kim M C, Chen C. A scientometric review of emerging trends and new developments in recommendation systems[J]. Scientometrics, 2015, 104(1): 239-263.

[2] Felfernig A, Jeran M, Ninaus G, et al. Basic approaches in recommendation systems[M]//Recommendation Systems in Software Engineering. Springer Berlin Heidelberg, 2014: 15-37.

[3] Linden G, Smith B, York J. Amazon. com recommendations: Item-to-item collaborative filtering[J]. Internet Computing, IEEE, 2003, 7(1): 76-80.

[4] Davidson J, Liebald B, Liu J, et al. The YouTube video recommendation system[C]//Proceedings of the fourth ACM conference on Recommender systems. ACM, 2010: 293-296.

[5] Das A S, Datar M, Garg A, et al. Google news personalization: scalable online collaborative filtering[C]//Proceedings of the 16th international conference on World Wide Web. ACM, 2007: 271-280.

[6] Lu H M, Lu Z X, Li Y D. TRUST!-A distributed multi-agent system for community formation and information recommendation[C]//Systems, Man, and Cybernetics, IEEE International Conference October 07-10, 2001, 3:1734-1739.

[7] Rosaci D, Sarné G M L. A multi-agent recommender system for supporting device adaptivity in e-commerce[J]. Journal of Intelligent Information Systems, 2012, 38(2): 393-418.

[8] Kurapati K, Gutta S, Schaffer D, et al. A multi-agent TV recommender[C]//Proceedings of the UM 2001 workshop "Personalization in Future TV. 2001.

[9] Gil A B, Peñalvo F J G. Learner Course Recommendation in e-Learning Based on Swarm Intelligence[J]. J. UCS, 2008, 14(16): 2737-2755.

[10] Birukou A, Blanzieri E, Giorgini P. Implicit: a multi-agent recommendation system for web search[J]. Autonomous Agents and Multi-Agent Systems, 2012, 24(1): 141-174.

[11] Batet M, Moreno A, Sánchez D, et al. Turist@: Agent-based personalised recommendation of tourist activities[J]. Expert Systems with Applications, 2012, 39(8): 7319-7329.

[12] Huang L, Dai L, Wei Y, et al. A personalized recommendation system based on multi-agent[C]//Genetic and Evolutionary Computing, 2008. WGEC'08. Second International Conference on. IEEE, 2008: 223-226.

[13] Martin D L, Cheyer A J, Moran D B. The open agent architecture: A framework for building distributed software systems[J]. Applied Artificial Intelligence, 1999, 13(1-2): 91-128.

[14] Case D M, DeLoach S A. Obaa++: an agent architecture for participating in multiple groups[C]//Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems. International Foundation for Autonomous Agents and Multiagent Systems, 2014: 1367-1368.

[15] Bradshaw J M, Dutfield S, Benoit P, et al. KAoS: Toward an industrial-strength open agent architecture[J]. Software agents, 1997: 375-418.

[16] Finin T, Fritzson R, McKay D, et al. KQML as an agent communication language[C]//Proceedings of the third international conference on Information and knowledge management. ACM, 1994: 456-463.

[17] Bellifemine F, Poggi A, Rimassa G. JADE–A FIPA-compliant agent framework[C]//Proceedings of PAAM. 1999, 99(97-108): 33.