2009 International Conference on Communication Software and Networks

Learning Task Allocation for Multiple Flows in Multi-agent Systems

Zheng Xiao, Shengxiang Ma, Shiyong Zhang
School of Computer Science
Fudan University
Shanghai, China
e-mail: xiaozheng206@163.com

***Abstract*—Task allocation is a key problem for agent to reach cooperation in multi-agent systems. Lately task flows are replacing traditional static tasks, thus real-time dynamic task allocation mechanisms draw more attention. Though scheduling single task flow is well investigated, little work on allocation of multiple task flows has been done. In this paper a distributed and self-adaptable scheduling algorithm based on Q-learning for multiple task flows is proposed. This algorithm can not only adapt to task arrival process on itself, but also fully consider the influence from task flows on other agents. Besides, its distributed property guaranteed that it can be applied to open multi-agent systems with local view. Reinforcement learning makes allocation adapt to task load and node distribution. It is verified that this algorithm improves task throughput, and decreases average execution time per task.**

Keywords-Agent cooperation; Task allocation; Multi-agent system; Multiple task flows; Q-learning

I. INTRODUCTION

In multi-agent systems (MAS), considering agent is often not versatile, through task allocation agents can cooperate and complete complex tasks. Besides, task allocation can optimize collaboration among agents, improving execution efficiency. As to autonomous MAS, task allocation is an intelligent decision-making problem. From the view of economics, decision rule is to maximize long-term expected reward.

Most early task allocation algorithms in parallel computing [1, 2] assume that all task types and their dependencies are known before allocation. Such problems are called static task allocation. However, in loose distributed applications like MAS, tasks arriving at each node can not be foreseen, but emerge at random. Here we call the task arrival processes in accordance with a certain probability distribution task flows. Under this condition, allocation optimally for all tasks at one time is unfeasible. Instead, dynamic scheduling according to allocation of arrived tasks and anticipation of possible arriving tasks is desired. Furthermore, different form single task flow allocation which is paid much attention, we investigate the scene where multiple task flows on different agents coexist. Undoubtedly, allocation for multiple task flows is more difficult. In Figure 1, agent a considers influence from not only task flow TF1 on itself, but also from flows TF2, TF3 on other agents. For example, even if c can finish task in TF1 more quickly than b, but if c is overloaded because of its own tasks, it may be more reasonable that a allocates tasks on it to b which is relatively free. As a result, the whole system composed of the three agents can fulfill tasks more efficiently with higher task throughput. Hence, scheduling for multiple task flows demands that agents can not only adapt to flow on themselves, but also take flows on other agents into account, aiming at maximize system performance or social welfare in economics.

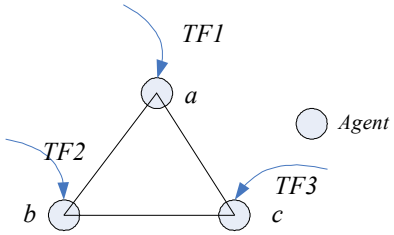

Figure 1. Scheduling for multiple task flows

Many features in MAS, which are different from them in multi-processor or cluster, further challenge task allocation. First, agent can join or exit system at will, which makes centralized allocation mechanism fail. Second, agent is locally visible, unable to observe behaviors or get all knowledge about other agents in the system. So, the task allocation is preferred which runs in distributed way with little communication or local interaction. In addition, uncertain task arrival process, cross influence from multiple task flows, and uncertain task execution cost, lead to the uncertain environment. Modeling uncertain environment is a hard job. Fortunately, reinforcement learning (RL) provides a feasible solution to this problem. RL has been applied in multi-agent collaboration problems [3, 4]. This method converges to optimal policy by receiving reinforcement signal from environment without modeling. On guide of requirements above, we propose a Q-learning based distributed algorithm to solve dynamic scheduling for multiple task flows. In this algorithm, agents interact with neighbor agents and make decisions independently. The decisions can not only adapt to task arrival process on itself, but also task flows on other agents, optimizing task allocation and increasing systematic long term expected reward.

The left is organized as follows. First, formalize the task allocation problem in multi-agent system. Then build its MDP (Markov Decision Process) model. Section 3 describes

978-0-7695-3522-7/09 $25.00 © 2009 IEEE
DOI 10.1109/ICCSN.2009.28

IEEE computer society

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on June 05,2025 at 03:18:36 UTC from IEEE Xplore. Restrictions apply.

the Q-learning algorithm for multiple task flows in detail. In the following some simulation is made, and analyze the results. In end, conclude this paper in section 5.

II. PROBLEM DEFINITION

A multi-agent system can be described using four algebraic structures A, C, AC, and AN. The set $A=\{a_1,a_2,\ldots,a_{|A|}\}$ represents all the agents in the system. The set $C=\{c_1,c_2,\ldots,c_{|C|}\}$ represents the capacity or resources of the system. The matrix $AC=[ac_{ij}]_{|A|\times|C|}$ ($ac_{ij}\in\{0,1\}$) denotes the map between agents and resources. If $ac_{ij}=0$, that means agent a does not possess the resource c_j. The matrix $AN=[an_{ij}]_{|A|\times|A|}$ ($an_{ij}\in\{0,1\}$) means network topology between agents. If $an_{ij}=1$, that means agent a_i connects a_j straightly. A task instance, represented by I_t^{ai}, means there is a task arriving at agent a_i at time t. The bounded types of task instances can be represented by $T=\{T_1,T_2,\ldots,T_{|T|}\}$ and a task flow TF is made up of the task instances sorted as the time of arrival. Task flow is an uncertain set related to time and comply with a certain probability distribution. a_i can decide to accept a task or reallocate to its neighbors. An action is in the form of $<I,a>$, and action set B of an agent can be defined as $\{<I,a>|a\in\{a_i\}\cup\{a_j|an_{ij}=1\}\}$ which means the actions the agent can choose. If task instance is assigned to Agent a_j, this action return reward $r=O_I-cost(I,a_j)$, where O_I is payoff after task I is completed, and $cost(I,a_j)$ is cost of execution.

Allocation of task flow in a multi-agent system can be described by Markov Decision Process (MDP), which is a four tuple $<S,B,\Gamma,R>$.

- S is the set of states, which is equivalent to task type ,S=T;
- B is the set of actions, the element of this set depend on agent's topology structure AN.
- Γ is state transition function: $S\times B\rightarrow Pr(S)$, where Pr(S) is a probability distribution of next state when taking the action b in state s.
- R is reward function: $S\times B\rightarrow\Re$, where r represents the immediate reward when taking the action b in state s.

We adopt discounted accumulative reward shown below as target function where γ is discounted factor.

$$\psi = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

From the formula above, expected reward can not be fixed until the goal is reached by a series of actions. Utility of current action depends on action in future. Discounted accumulative reward ψ relies on state transition function Γ, i.e. task arrival process. But in reality, this knowledge is hard to get. In the next section we will use Q-learning one method of model-free RL to solve this problem.

III. SCHEDULING MULTIPLE TASK FLOWS

In this section we try to use Q-learning to schedule multiple task flows. First, on the basis of current work introduce a general Q-learning model under single task flow. Then, point out some deficiency when updating value function under multiple task flows, and solve them through shared value function. In end, on account of properties of MAS, give an implementation of above-mentioned method.

A. Q-learning forSsingle Task Flow

When there is only one task flow arriving at system, agent should consider two factors as making decisions. One is load on each agent, which determines cost to execute some task, i.e. immediate reward of individual actions can be calculated. The other is predicting task arrival process. Because tasks emerge stochastically, though immediate reward is the highest, in the long run, system reward may not be the same. Knowing arrival process makes it possible to make the very decisions that maximize system reward in long term. Q-learning can implicitly learn and gradually adapt to the two parameters. If task I (type $T_I\in T$) arrives at agent a_0, the expected reward of action $b_j=<I,a_j>\in B$ which a_0 selects is:

$$Q(T_I,b_j)=(1-\alpha)Q(T_I,b_j)+\alpha[R(T_I,b_j)+\gamma V(T_I')]$$

Where α ($0\le\alpha<1$) is the learning rate, and $V(T_I')$ is expected reward at the next state T', written as $V(T_I')=\max_{b_i}Q(T_I',b_i)$. In a long-term, agent receives the reinforcement signal continuously and finally converges to the optimal strategy $\pi_t=\arg\max_{b_i}Q(T_I,b_i)$. According to the reinforcement learning theory, the learning process attempts to maximize $\Sigma_i\gamma^i R_i(T_I,b)$. Since there is no other task flows, when discounted accumulative reward of a single agent is maximal, the whole system performs the best as well.

B. Shared Value Function V

Under multiple task flows, the method above may fail. System reward does not depend on discounted accumulative reward of a single agent. Agent should not only finish task flow on itself, but also perhaps help do tasks from other agents. Maximizing its own reward may decrease that of other agents, lowering system reward. Below, we analyze two kinds of influence among task flows, and give corresponding solutions.

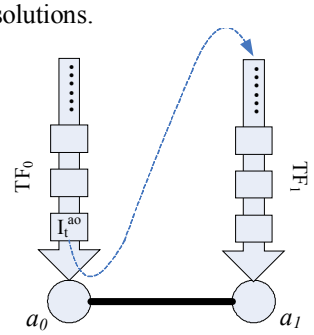

Figure 2. Influence of master-slave flows

For the first case as shown in figure 2, TF_0 and TF_1 are task flows on agent a_0 and a_1 respectively. Agent a_0 assign task I_t^{ao} ($T_I\in T$) to neighbor a_1. We call TF_0 master flow, while TF_1 slave flow. According to Q-learning for single task flow, immediate reward after a_0 selects action $<I_t^{ao},a_1>$ is $r=O_I-cost(I_t^{ao},a_1)$, which reflects load on a_1. But because there exist TF_1 on a_1, immediate reward when a_0 allocates to

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on June 05,2025 at 03:18:36 UTC from IEEE Xplore. Restrictions apply.

a_1 is not only function of load on a_1, but also function of tasks arriving at a_1 in future. For instance, at time t a_1 has light load of tasks to be executed, i.e. immediate reward computed as $r=O_I\text{-}cost(I_t^{ao},a_1)$ is maximal, but if in TF_1 there are a large batch of tasks of type T_I arriving, then it is not the optimal decision to assign I_t^{ao} to a_1. In Q-learning, value function $V(T_I)$ means long-term accumulative reward at state T_I. Generally speaking, that is reward by allocating task of type T_I. This function embodies influence on current decision of arriving tasks in future. Therefore, suggest that immediate reward should be computed as $r=O_I\text{-}cost(I_t^{ao},a_1)+V_1(T_I)$, where $V_1(T_I)$ is value function of allocated agent. In this way, load and influence of task flow on allocated agent are both examined.

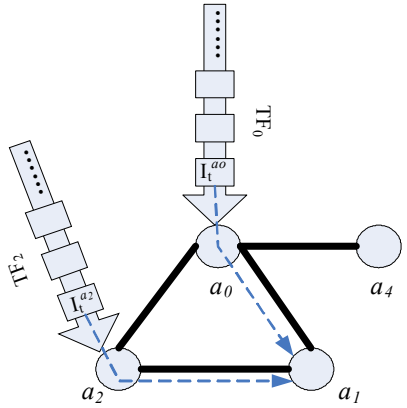

Figure 3. Influence of non master-slave flows

Except from master-slave relationship, non-slave flows also influence decision-making of master flow. Figure 3 is an example of such case. Suppose that at time t agent a_1 execute task of type T_I more quickly than agent a_4. Usually, a_0 tend to assign arrived task of type T_I to a_1, obtaining higher payoff. But this allocation neglects influence from non-slave task flow TF_2. If after time t agent a_2 has a large number of tasks for a_1, perhaps it is much more rational that a_0 assign task I_t^{ao} to suboptimal agent a_4. Reward of a_0 is cut down a little, but from long term, tasks on a_2 get quicker response, improving system performance. Policy of a_0 depends on values of its Q function. V function of a_2 implies information about task arrival distribution in TF_2. So adding values of V function of agents with non-slave flows to Q value update of a_0 will make target function transfer utility from individual gents to the entire system.

On account of analysis above, Q function should be changed as follows.

$$Q(T_I,b_j)=(1-\alpha)Q(T_I,b_j)+\alpha[R(T_I,b_j)+\gamma\sum_k\omega(i,k)V_k(T_I')]$$

Where $R(T_I,b_j)=O_I\text{-}cost(T_I,b_j)$, $k\in\{a_i\}\cup\{a_k|an_{ik}=1\}$ is neighbors of Agent a_i, $\omega(i,k)$ is weight of influence of task flows on adjoining agents. Through sharing value function, system performance is improved. Theory about this mechanism can refer to [9], which illustrate sharing value function indeed maximizes discounted accumulative reward of weighted sum of all agents' reward. This method is an effective solution used in Q-learning to optimize system reward.

C. Online Distributed Algorithm

As for a practical multi-agent system, there are still some problems to be solved before the mechanism is applied. First of all, set of state S is determined by types of tasks to be allocated. In multi-agent domains, each agent has its own task flow to allocate. Under this situation, state s is made up of types of task to be allocated on all agents, which means S is a joint state space $<S_1,S_2,\ldots,S_{|A|}>$. In an open and locally visible multi-agent system, it is difficult to get this joint state. Agents often make their decisions based on local belief and little communication. So it is necessary to localize the joint state in order to be able to learn and decide independently. A simple but effective way is to use local state set S_i as state space in Q-learning. On the other hand, because state space is localized, states of agents are unknown to each other. Fortunately, the mechanism in last subsection only needs V value of neighbors. So at the end of action execution, agents broadcast their V values of next states to their neighbors, and then update Q function concurrently, there is no need to model other agents in order to get their V values. In brief, the algorithm includes four stages: initialization, action selection, value function synchronization, Q function update. The details are described below.

Algorithm 1 Learning for multiple task flows

(1) Initialization
  for any $T_I\in T$, any $b_j\in B$
    $Q(T_I, b_j)=0$
(2) action selection
  if Explore with probability P_e
    Select action b_j randomly by uniform distribution
  else
    Select action b_j by policy $\pi=\arg\max_{b_i}Q(T_I,b_i)$
(3) Value function synchronization
  Broadcast ($V(T_I')$, Neighbors), where T_I' is the type of next task in TF.
(4) Learning
  Update (Q):

$$Q(T_I,b_j)\leftarrow(1-\alpha)Q(T_I,b_j)+\alpha[R(T_I,b_j)+\gamma\sum_k\omega(i,k)V_k(T_I')]$$

  Update (V): $V(T_I)\leftarrow\max_{b_j}Q(T_I,b_j)$
  Goto (2)

IV. Experiments and Results

A good task allocation algorithm tries to increase finished tasks in a single time unit, and meanwhile lessen the time to execute a task. In this section we validate our algorithm from the very two aspects. The former is measured by task throughput, while the latter is evaluated by average execution cost per task which includes the cost spent in waiting in queue and time on computation. The lower average execution cost is, the shorter average queue length is.

The simulation environment involves 20 agents, as in the figure 4. Every circle represents an agent, identified by the number in centre. Each agent chooses other three agents as its neighbors randomly. Suppose the system has five kinds of

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on June 05,2025 at 03:18:36 UTC from IEEE Xplore. Restrictions apply.

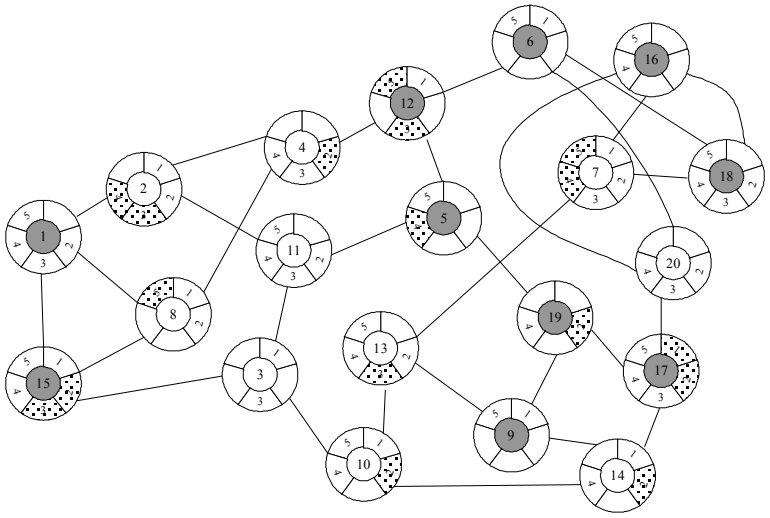

Figure 4. Multi-agent system with 20 nodes

resources and the agent can finish a task if and only if the agent has the necessary resource. To simulate heterogeneity of the agent capability, five kinds of resources is distributed on agents randomly which could be a normal resource at a probability 0.67(need five time units to finish a task), otherwise be a fast resource (need three time units). In figure 4, the number in sector represents the type of resources and shadow means it is a fast resource. The heavy color in the mid of a circle means there is a task flow on that agent and the tasks arrive at a rate of 0.3. There are five types of task arriving by uniform distribution and constrained by one resource each. A task is dropped after delivered 10 times, to prevent loop. The results are shown in figures below.

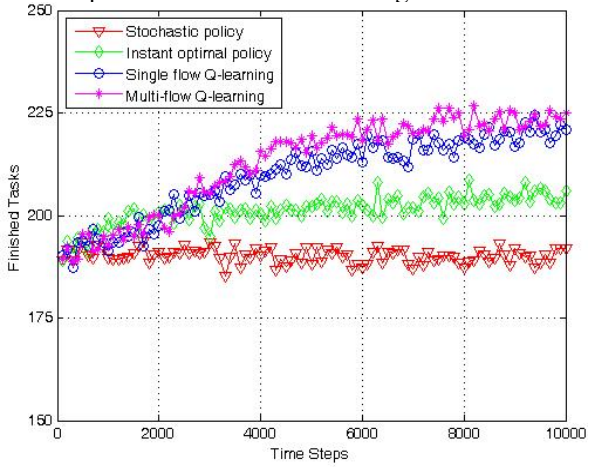

Figure 5. Throughput of every 100 steps

We compare our algorithm with stochastic, instant optimal and single flow Q-learning algorithms. From task throughput curves in figure 5, when below 2000 steps, instant optimal algorithm performs well because it can make full use of the idle system but the learning algorithms do not converge. After that, two learning algorithms stand out, especially ours. In figure 6, at the beginning due to low system load, performance of learning algorithms equals to that of instant optimal one. But with tasks arriving continuously, system is becoming saturated. The average execution cost of two learning algorithms increases slowly because of the ability of adapting to task arrival process and cross influence among multiple task flows. From both indexes, our algorithm performs better than other ones in the long run.

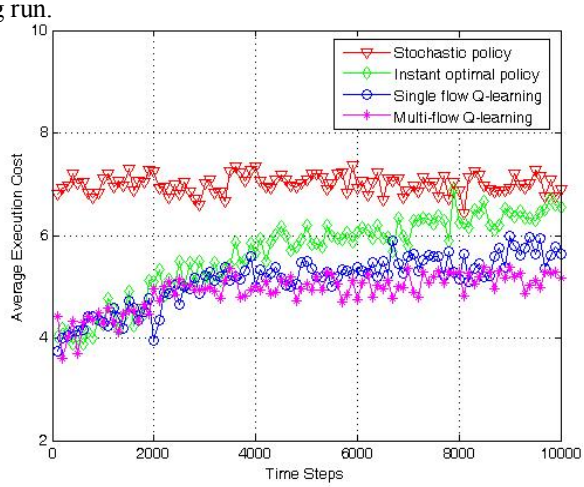

Figure 6. Average execution cost for a task

V. CONCLUSION

Research on task allocation has experienced such background or stage as parallel computing, distributed computing, and multi-agent computing. We start with allocation for single task flow, and discuss the problem when multiple task flows coexist. As to single task flow, rational decision needs to consider influence of tasks arriving later. In contrast, facing multiple task flows, cross influence among them can not be ignored. Furthermore, open, locally visible, and uncertain environment is also critical to the algorithm design. Consequently, thinking of model-free property of Q-learning, by use of value function share, a distributed

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on June 05,2025 at 03:18:36 UTC from IEEE Xplore. Restrictions apply.

scheduling algorithm for multiple task flows is proposed. This method models task allocation on each agent as an MDP, and let individual agent make decisions which maximize system utility by a little extra interaction with neighbors, helping local MDPs of each agent reach collaborative.

REFERENCES

[1] H. EI-Rewini, T. G. Lewis. Scheduling parallel program tasks onto arbitrary target machines. Journal of Parallel and Distributed Computing, 1990, vol. 9, pp: 138-153.

[2] Hesham Ali, Hesham EI-Rewini. Task Allocation in Distributed Systems. Journal of Combinatorial Mathematics and Combinatorial Computing, 1993, vol. 14, pp: 15-32.

[3] Yoav Shoham, Rob Powers, Troud Grenager. Multi-Agent Reinforcement Learning: A Critical Survey. Technical report, Stanford University, 2003.

[4] Zhong Yu, Gu Guochang, Zhang Rubo. Survey of Distributed Reinforcement Learning Algorithms in Multi-agent Systems. Control Theory and Applications, 2003, Vol. 20 No. 3, pp: 317-322.

[5] Hosam Hanna, Abdel-lllah Mouaddib. Task Selection Problem under Uncertainty as Decision-Making. In: Proc. of International Conference on Autonomous Agent and Multi-Agent System (AAMAS), 2002, pp: 1303-1308.

[6] Abdallah Sherief, Lesser Victor. Modeling Task Allocation Using a Decision Theoretic Model. In: Proc. of Fourth International Joint Conference on Autonomous Agents and Multi-agent Systems, ACM Press, 2005, pp: 719-726.

[7] Abdallah Sherief, Lesser Victor. Learning Task Allocation via Multi-Level Policy Gradient Algorithm with Dynamic Learning Rate. In: Proceedings of Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains, the International Joint Conference on Artificial Intelligence, IJCAI, 2005, pp: 76-82.

[8] Abdallah Sherief, Lesser Victor. Learning the Task Allocation Game. In: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multi-Agent Systems, ACM Press, 2006, pp: 850-857.

[9] J. Schneider, W. K. Wong, A. Moore, M. Riedmiller. Distributed Value Functions. In: Proceedings of the Sixteenth International Conference on Machine Learning, 1999, pp: 371-378.

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on June 05,2025 at 03:18:36 UTC from IEEE Xplore. Restrictions apply.