基于序贯博弈多智能体强化学习的综合模块化航空电子系统重构方法

张 涛,张文涛,代 凌,陈婧怡,王 丽,魏倩茹
(西北工业大学软件学院,陕西西安 710065)

摘 要: 动态重构是一种有效的综合模块化航空电子系统故障容错方法. 重构蓝图定义了系统故障环境下的应用迁移与资源重配置方案,是以最小代价重构恢复系统功能的关键. 在复杂多级关联故障模式下,如何快速自动生成有效重构蓝图是其难点. 针对该问题,本文提出一种基于序贯博弈多智能体强化学习的综合模块化航空电子系统重构方法. 该方法引入序贯博弈模型,将因受故障影响而需要迁移重构的应用软件定义为博弈中的智能体,根据应用软件优先级确定序贯博弈的顺序. 针对序贯博弈过程中多智能体间竞争与合作的问题,算法使用强化学习中的策略梯度,通过控制与环境交互中的动作选择概率来优化重构效果. 应用基于有偏估计的策略梯度蒙特卡洛树搜索算法更新博弈策略,解决了传统策略梯度算法震荡难收敛、计算耗时长问题. 实验结果表明,与差分进化、Q学习等方法相比,所提算法的优化性能和稳定性均具有显著优势.

关键词: 综合模块化航空电子系统; 序贯博弈; 策略梯度; 多智能体强化学习; 蒙特卡洛树搜索; 重构
中图分类号: TP391 **文献标识码:** A **文章编号:** 0372-2112(2022)04-0954-13
电子学报 URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20211268

Integrated Modular Avionics System Reconstruction Method Based on Sequential Game Multi-agent Reinforcement Learning

ZHANG Tao, ZHANG Wen-tao, DAI Ling, CHEN Jing-yi, WANG Li, WEI Qian-ru
(*School of Software*, *Northwestern Polytechnical University*, *Xi'an*, *Shaanxi* 710065, *China*)

Abstract: Dynamic reconfiguration is an efficient fault-tolerant approach for integrated modular avionics(IMA) systems. The reconfiguration blueprint defines the application migration and resource reconfiguration scheme in the system failure environment, which is the key to reconfiguring and recovering the system function with minimum cost. How to generate effective reconfiguration blueprints rapidly and automatically in complex multi-level associated failure modes is the difficulty. This paper proposes an IMA system reconfiguration method based on sequential game multi-agent reinforcement learning to solve the problem. The sequential game model is introduced in this method. We define the application software needs to be migrated as the agent in the game. The sequence of sequential game is determined according to the priority of the application software. Aiming at the problem of competition and cooperation among multiple agents in the process of sequential game, the algorithm introduces policy gradient of reinforcement learning and optimizes the reconfiguration effect by controlling the action selection probability in interaction with the environment. The policy gradient Monte Carlo tree search algorithm based on biased estimation is applied to update game strategy, which solves the problems of oscillation, difficulty in convergence, long calculation time of the traditional policy gradient algorithm. Experimental results indicate that compared with differential evolution and Q-learning methods, the proposed algorithm has significant advantages in convergence and efficiency.

Key words: integrated modular avionics(IMA) system; sequential game; policy gradient; multi-agent reinforcement learning; Monte Carlo tree search; reconfiguration

收稿日期:2021-09-17;修回日期:2022-03-08;责任编辑:王天慧
基金项目:国家自然科学基金(No.61901388,No.62001386);航空科学基金(No.2015ZD53055,No.20185853038,No.201907053004);上海航天科技创新基金(No.SAST2021-054)

1　引言

在综合模块化航空电子(Integrated Modular Avionics,IMA)系统[1]中,动态重构是一种有效的故障容错方法,已成功被应用于F-22,F-35,A380,B787等飞机航电系统[2]. 重构蓝图定义了受故障影响的各个应用软件的动态迁移与系统资源重配置策略. 在故障发生时,系统将依据所生成重构蓝图对受故障影响的应用软件进行动态迁移,使得应用软件能够恢复正常运行[3,4]. 目前主要针对少数单一故障模式,基于专家经验人工设计重构蓝图,系统故障容错能力有限[5]. 而对于复杂多级交联故障模式,由于系统可用资源限制,需要多个应用软件动态迁移,甚至牺牲低优先级应用软件,使得重构蓝图的人工规划困难[6,7]. 因此,研究自动化生成重构蓝图的方法,是提高IMA系统重构容错能力的关键.

重构蓝图生成需要综合考虑系统负载均衡、重构影响、重构恢复时间和重构降级等多个因素,是一个典型的NP完全问题[8,9]. 为解决多目标优化的系统重构问题,传统动态规划[10,11]方法以空间换时间,在数据量大时会造成很大的浪费;分支界限算法[12]在单目标优化上有优势但是无法考虑多目标综合优化;模拟退火[13]等启发式算法虽然可解决多目标优化重构调度问题,但却容易陷入局部最优;基于种群进化思想的遗传[14]和差分进化(Differential Evolution,DE)[15]等算法虽然可获得更优重构调度解,但求解时间过长;使用强化学习中的Q学习[16]在低维重构调度上可以实现快速重构调度,但容易震荡,难以收敛;基于模拟退火的Q学习[17]同时结合了模拟退火和Q学习的优点,收敛效果更好,但算法迭代的次数多,无法快速求解. 单纯的策略梯度[18]通常采用随机梯度下降的算法方法快速求解,但在算法收敛上具有一定的震荡特性,并且随着迭代次数的增加其无偏估计量的计算耗时也随之增加,导致在迭代次数较大时算法迭代更新速度过慢. 而有偏估计[19]可以有选择地提取历史经验,在有效地减少计算耗时的同时使期望向造成高回报值的动作方向快速偏移. 使用蒙特卡洛树搜索(Monte Carlo Tree Search,MCTS)[20]可以在已知策略空间上探索出最优的动作策略,若是策略空间探索不足则容易陷入局部最优.

综上考虑,本文提出一种结合序贯博弈[21]的多智能体强化学习算法来模拟重构应用软件的重构过程,将多目标的优化过程转化为序贯博弈过程中的竞争与合作目标,以优化求解重构蓝图. 重构中每个待调度应用软件均作为一个智能体,各个智能体不仅要争取自身最优条件,而且要满足整体的多目标优化需求,直到达到混合均衡条件[22]或最大博弈次数. 算法首先根据重构应用软件优先级确定其所对应智能体的博弈顺序. 然后每个智能体使用策略梯度,根据各自动作是否满足约束以及满足约束的程度进行自身策略的迭代. 接着使用MCTS方法,根据重构后的目标函数值优劣对全部智能体的策略进行更新,从而快速逼近均衡条件. 与传统优化算法和传统强化学习算法相比,所提出算法的优化性能和稳定性更优.

2　IMA系统重构

2.1　IMA系统重构架构

在IMA系统重构架构中,当CPU发生故障时,需要根据重构蓝图将受故障影响的应用软件迁移到其他CPU,为其重新配置资源,使应用软件恢复正常运行. 如图1所示,在CPU_2发生故障后,将其分区内的应用软件C和应用软件D分别迁移调度到CPU_1和CPU_3的分区1与分区5中. 基于重构蓝图,被牺牲的应用软件和需要迁移的应用软件越少,应用软件功能恢复时间越短,系统负载越均衡,则重构蓝图质量越好.

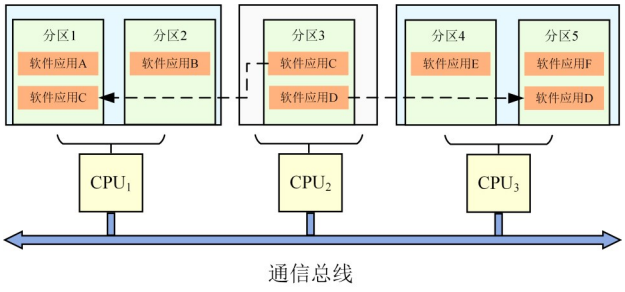

图1　重构架构示意图

2.2　重构模型

将IMA系统中的一组CPU处理器记作$C=\{C_1,C_2,\cdots,C_b\}$,其中C代表CPU的集合,第l个CPU表示为C_l,b代表CPU的总数量. b个CPU中共有m个可用分区,记作$P=\{P_1,P_2,\cdots,P_m\}$,其中第j个分区表示为P_j,其属性有分区内存PM和分区框架时间Td. 分区P_j中部署了软件序列M^{P_j},记为$M^{P_j}=\{M_1^{P_j},M_2^{P_j},\cdots,M_c^{P_j}\}$,其中分区$P_j$中部署的第$k$个应用软件表示为$M_k^{P_j}$,其属性有内存RAM、优先级Priority、最大运行时间WCET和截止时间deadline. 在系统故障时,将生成受故障影响软件集,记为$D=\{D_1,D_2,\cdots,D_n\}$,其中第i个应用软件表示为D_i. 如图2所示,在分区P_2发生故障后,IMA系统会将部署在该分区的软件D_1,D_2和D_3分别迁移至其他可用分区P_4,P_1和P_5中,使受到影响的软件继续运行. 由于系统重构时需要将受故障影响软件迁移至其他可用的分区中,因此需要计算迁移后分区的CPU使用率、内存使用率对应的分区负载,确保应用软件的资源可调度性.

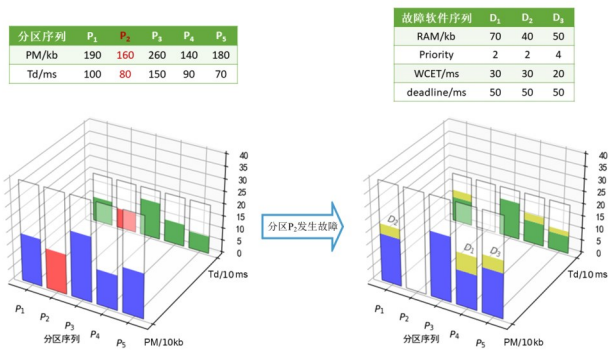

图2 重构模型示例图

(1)分区CPU使用率

分区CPU使用率体现了分区内应用软件的最大运行时间占用分区框架时间的比例. 具体公式如下所示:

$$C_{\text{use}}(P_j)=\frac{\sum_{k=1}^{c}\text{WCET}_{M_k^{P_j}}}{\text{Td}_{P_j}} \tag{1}$$

其中,$\text{WCET}_{M_k^{P_j}}$表示分区P_j中第k个应用软件$M_k^{P_j}$的最大运行时间;Td_{P_j}表示分区P_j的分区框架时间.

当应用软件D_i被调入P_j后,有

$$C_{\text{use}}(P_j,D_i)=\frac{\sum_{k=1}^{c}\text{WCET}_{M_k^{P_j}}+\text{WCET}_{D_i}}{\text{Td}_{P_j}} \tag{2}$$

其中,WCET_{D_i}表示应用软件D_i的最大运行时间.

重构可调度性约束1. 分区资源限制

当待重构应用软件D_i被调入分区P_j后,分区当中的CPU使用率应满足$C_{\text{use}}(P_j,D_i)\leqslant C_{\text{use-max}}$.

(2)分区内存使用率

分区内存使用率体现了分区内应用软件的内存占用分区内存的比例. 具体公式如下所示:

$$RM_{\text{use}}(P_j)=\frac{\sum_{k=1}^{c}\text{RAM}_{M_k^{P_j}}}{\text{PM}_{P_j}} \tag{3}$$

其中,$\text{RAM}_{M_k^{P_j}}$表示分区P_j中第k个应用软件$M_k^{P_j}$的内存容量;PM_{P_j}表示分区P_j的内存容量.

当应用软件D_i被调入P_j后,有

$$RM_{\text{use}}(P_j,D_i)=\frac{\sum_{k=1}^{c}\text{RAM}_{M_k^{P_j}}+\text{RAM}_{D_i}}{\text{PM}_{P_j}} \tag{4}$$

其中,RAM_{D_i}表示应用软件D_i的内存容量.

重构可调度性约束2. 分区内存限制

当待重构应用软件D_i被调入分区P_j后,分区当中的内存应满足调用条件$RM_{\text{use}}(P_j,D_i)\leqslant RM_{\text{use-max}}$.

(3)分区负载

分区负载综合考虑了分区CPU使用率和分区内存使用率,是对两个指标的统一处理,分区负载越低,意味着该分区应用软件占用的资源越少,分区剩余的资源越多. 具体公式如下所示:

$$\begin{aligned}&L(P_j)=\mu_1 C_{\text{use}}(P_j)+\mu_2 RM_{\text{use}}(P_j)\\&\text{s.t.}\quad \mu_1+\mu_2=1\end{aligned} \tag{5}$$

当第D_i个应用软件被调入P_j后,有

$$\begin{aligned}&L(P_j,D_i)=\mu_1 C_{\text{use}}(P_j,D_i)+\mu_2 RM_{\text{use}}(P_j,D_i)\\&\text{s.t.}\quad \mu_1+\mu_2=1\end{aligned} \tag{6}$$

2.3 应用软件重构问题

结合IMA系统重构模型,重构中每个待调度的应用软件都期望在成功调度的同时被调入剩余资源最丰富的分区,本文使用式(6)作为应用软件被调入分区后资源丰富情况的数学表达,分区负载越小,代表该分区剩余的资源越多. 设重构影响集$I=\{I_1,I_2,\cdots,I_n\}$为重构过程中系统受影响的软件序列集,初始时重构影响集等于受故障影响的软件集,即$I=D$.

当影响集I内所有的应用软件被调入分区序列P中时,设$X=\{[x_{11},x_{12},\cdots,x_{1m+1}],\cdots,[x_{n1},x_{n2},\cdots,x_{nm+1}]\}$,$x_{ij}$表示软件$I_i$被调入至$P_j$的分区中. $x_{ij}=1$表示软件被正常调度,$x_{ij}=0$表示不进行调度,当$x_{ij}=1,j=1+m$时表示软件I_i被牺牲.

因此,对每个待调度的软件I_i,有目标函数f_1:

$$\begin{aligned}\min f_1&=\sum_{j=1}^{m}L(P_j x_{ij},I_i)\\&=\sum_{j=1}^{m}\left[\mu_1 C_{\text{use}}(P_j x_{ij},I_i)+\mu_2 RM_{\text{use}}(P_j x_{ij},I_i)\right]\end{aligned}$$

$$\text{s.t.}\begin{cases}x_{ij}\in\{0,1\}\\\sum_{j=1}^{m+1}x_{ij}=1\\\mu_1+\mu_2=1\\\forall RM_{\text{use}}(P_j,I_i)\leqslant RM_{\text{use-max}}\\\forall C_{\text{use}}(P_j,I_i)\leqslant C_{\text{use-max}}\\C_{\text{use}}(0,I_i)=RM_{\text{use}}(0,I_i)=0\end{cases} \tag{7}$$

并且在重构结束后,在满足式(7)约束的情况下,系统期望拥有一个综合评价高的重构蓝图. 针对系统重构后的综合评价,本文借鉴文献[17]中的4个评价指标,分别改进为表示重构后系统的负载均衡、重构影响度、重构恢复时间和重构降级率4个指标,定义如下.

(1)负载均衡

负载均衡表示系统资源使用的均衡程度,可以表示重构后系统资源使用的分布情况. 在重构结束后,可以依据所有可用分区的负载计算系统资源的使用均衡程度,由此定义负载均衡指标:

$$\mathrm{LB}(P,I,X)=1-\sqrt{\frac{1}{m}\sum_{j=1}^{m}\left[L\left(P_j,\sum_{i=1}^{n}x_{ij}I_i\right)-\overline{L(P^X,I)}\right]^2}$$
$$\text{s.t.}\quad L(P_j,0)=L(P_j) \tag{8}$$

$$\overline{L(P^X,I)}=\frac{1}{m}\sum_{j=1}^{m}L\left(P_j,\sum_{i=1}^{n}x_{ij}I_i\right) \tag{9}$$

其中,n表示发生故障需要重构的软件数,m表示可调度分区总数,$\overline{L(P^X,I)}$表示软件序列I按X调度后所有分区负载均衡的均值.

(2)重构影响度

重构影响度指执行重构后对原系统软件状态的影响程度. 按照软件对系统影响的等级将应用软件的重要性分为五个优先级. 数字越大,应用软件就越重要. 重构应尽量不影响原系统的软件状态,优先级高的软件应该优先被调入,若调入失败,即不满足约束时,应调换出优先级低的软件放入待调度软件序列,同时放入重构影响集I中,若分区中的软件优先级均比待调入软件优先级高,则调度失败,直至所有的分区均无法满足约束,则停止且将该软件放入重构牺牲集De. 设因优先级被调出的软件共k个,则重构影响集I中共有原先序列D中的n个软件和新调出的k个软件. 由此定义重构影响度指标:

$$\mathrm{Im}(\mathrm{De},I)=\begin{cases}1-\dfrac{\sum_{i=1}^{n_m}\mathrm{Pri}_{\mathrm{De}_i}+\sum_{j=n+1}^{n+k}\mathrm{Pri}_{I_j}}{\sum_{i=1}^{n+k}\mathrm{Pri}_{I_i}}\end{cases} \tag{10}$$

其中,Pri表示软件优先级;n_m表示重构牺牲集中的软件个数,即重构后因为可用资源不足而不得不牺牲的软件总数;$n+k$是重构影响集的个数,表示发生故障的处理器中需要重新配置的软件总数;$\mathrm{Pri}_{\mathrm{De}_i}$表示重构牺牲集中第$i$个软件的优先集;$\sum_{j=n+1}^{n+k}\mathrm{Pri}_{I_j}$表示重构影响集中从原系统因优先级低被抢占调出的软件个数.

(3)重构恢复时间

系统重构过程中进程的恢复占据了主要时间,并假设当多个进程位于同一处理器时,重构加载是串行的,重构时间需要累加计算;当多个进程位于不同处理器时,重构加载是并行的,将其中最大的恢复时间作为系统重构时间. 由此定义重构恢复时间指标:

$$T_{\mathrm{re}}=1-\frac{T_s}{T_{\max}}$$
$$\text{s.t.}\quad T_s<=T_{\max} \tag{11}$$

其中,$T_{\max}$表示最大重构时间,通常取值为CPU主框架时间;T_s表示系统重构时间,取IMA系统内所有处理器的重构恢复时间的最大值,其计算公式为

$$\begin{aligned}&T_s=\mathrm{Max}(\{T_{C_l}\}),T_{C_l}=\mathrm{Max}(\{T_{P_j}\})\\&l\in[1,b],j\in[1,b_m]\end{aligned} \tag{12}$$

其中,T_{C_l}表示处理器C_l的重构恢复时间,取该处理器内b_m个分区中最大的分区重构时间;T_{P_j}表示P_j分区内需要重构进程的重构时间和,其计算公式为

$$T_{P_j}=\sum_{k=1}^{N_{\mathrm{re}}}T_{M_k^{P_j}} \tag{13}$$

其中,N_{re}表示分区P_j内进行重构加载的进程数量;$T_{M_k^{P_j}}$表示进程$M_k^{P_j}$的重构时间,与进程大小有关.

(4)重构降级率

在IMA系统的重构过程中,位于故障处理器内的部分进程可能由于系统冗余资源不足、重构时间限制等未能及时完成重构,则定义重构牺牲集De,表示重构结束后仍然剩余不得不被牺牲的软件. 由此定义重构降级率指标:

$$\mathrm{Sacrif}=1-\frac{\sum_{i=1}^{n'}\mathrm{Pri}_{\mathrm{De}_i}}{\sum_{k=1}^{\mathrm{Nm}}\mathrm{Pri}_{M_k^{P_j}}} \tag{14}$$

其中,n'表示重构牺牲集De的进程数,即需要牺牲的进程总数;Nm表示系统中的全部进程总数;$\mathrm{Pri}_{M_k^{P_j}}$表示进程$M_k^{P_j}$对应的重要等级.

重构结束后蓝图四个指标的值越大,说明重构蓝图的质量越高. 因此,有目标函数f_2:

$$\begin{aligned}&\max f_2=\lambda_1\mathrm{LB}(P,I,X)+\lambda_2\mathrm{Im}(\mathrm{De},I)+\lambda_3T_{\mathrm{re}}+\lambda_4\mathrm{Sacrif}\\&\text{s.t.}\begin{cases}0<\lambda_i<1,i\in[1,4]\\\lambda_1+\lambda_2+\lambda_3+\lambda_4=1\end{cases}\end{aligned} \tag{15}$$

3　序贯博弈强化学习重构方法

3.1　强化学习序贯博弈模型

本文设置的序贯博弈模型是在强化学习常用的马尔科夫决策过程的基础上[23],由六元组$<G,S,\pi,A,R,H>$构成.

智能体G:重构中需要调度的每个应用软件定义为一个自主的智能体,它们独立地与环境交互并根据对前面智能体行为的观察采取自己的策略,为自己实现最大收益或最小损失.

状态S:状态为当前系统中所需重构(故障的)的应用软件序列D,以及所有可被调度(正常的)的分区状态序列P与CPU状态序列C. 设状态$S=<C^s,P^s,D^s,Disp>$,其中处理器$C^s=\{C_1,C_2,\cdots,C_b\}$,分区$P^s=\{P_1,P_2,\cdots,P_m\}$和$D^s=\{D_1,D_2,\cdots,D_n\}$分别代表可被调度的CPU序列、可被调度的分区序列和所需重构的应用软件序列. $\mathrm{Disp}=\{D_i\rightarrow P_j\}$代表应用软件进程$D_i$分配到分区$P_j$的映射[24]关系.

策略π:智能体在当前状态下,选取动作的策略可

以表示为以策略空间分布的概率选取分区作为执行动作. 每个智能体在多智能体序贯博弈中遵循自己的策略,旨在当受环境和其他智能体的策略影响时使代理的奖励最大化和成本最小化.

动作A:选取动作的过程其实就是重构中调度的过程,即在当前状态S下选中某个分区,将待调度的应用软件部署到该分区的调度过程. 例如在调度第i个应用软件时,动作$A=\mathrm{re}<I_i,P_j,C_l>$,其中re表示将应用软件$I_i$重新部署到$C_l$处理器中$P_j$分区. 执行动作$A$后,状态从$S_t$进行更新$S_t\rightarrow S_{t+1}$.

回报函数R:本文设计了两个回报函数,分别对应智能体执行动作的成本以及奖励. 将智能体调度动作满足约束的程度作为智能体执行动作的成本回报值,回报值越高,意味着动作成本越少. 在重构中所有智能体所有的动作都执行结束后,对重构后的系统状态做出的综合评价作为智能体们执行动作奖励的回报值,回报值越高,意味着该次重构效果越好.

策略迭代函数H:本文基于两个回报函数设计了两种策略迭代的算法,使用基于强化学习的策略梯度算法和MCTS算法分别将智能体的成本回报值和总体回报值反馈给智能体的策略,使得策略按着智能体期望的方向趋近更新.

如图3所示,在博弈t_1中,首先由最高优先级软件D_1对应的智能体G_1在初始状态S_0下以策略π_1做出动作A_j的选择,即选择将应用软件D_1调度到分区P_j.

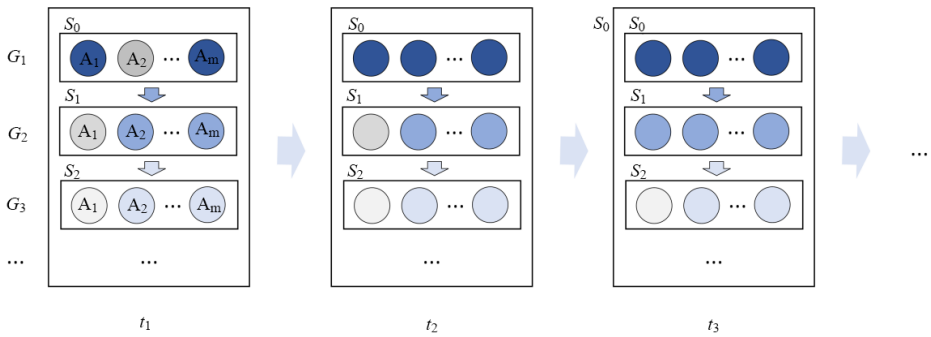

图3 序贯博弈模型示意图

计算执行该动作的成本回报值后该智能体的策略相应的进行更新,状态也相应的改变为S_1. 次一级的智能体需要在上一级智能体的行为结果S_1上做出动作选择,直至所有的智能体调度完毕,该次重构结束. 重构结束后对重构后系统进行综合评价,并作为智能体们的奖励,它们对应的策略亦做出更新. 这轮博弈t_1结束,接着开启下一轮的博弈t_2,直至达到混合纳什均衡或设定的最大博弈数.

3.2 博弈竞争模型

每个智能体是相互独立的,其对动作都有一个相应的策略空间与历史成本经验池. 单个智能体调度动作执行后,所部署分区的CPU使用率与内存使用率越小,说明该智能体本次重构调度的成本越小,若超过设定的约束值,则该次重构调度不佳.

当智能体G_i对应的应用软件D_i选择的动作为调入分区P_j时,对目标函数1进行转化有

$$r_t^i=\begin{cases}1-[\mu_1 C_{\mathrm{use}}(P_j,D_i)+\mu_2 RM_{\mathrm{use}}(P_j,D_i)]\\ -1,\quad C_{\mathrm{use}}(P_j,D_i)>C_{\mathrm{use-max}}\\ \qquad \text{or } RM_{\mathrm{use}}(P_j,D_i)>RM_{\mathrm{use-max}}\end{cases}\tag{16}$$

其中,r_t^i表示在第t次博弈智能体G_i依据策略分配后获的成本回报值. 成本回报值越大,代表该次动作对此次调用所付出的成本越少,对该智能体越有利.

因此,在博弈的竞争模型中,通过博弈交互的成本回报值高低和历史的经验池,每个智能体自主执行策略学习,最优化自己动作所得到的回报,不需要考虑其他智能体的回报情况. 它们相互竞争,每个智能体的都旨在优化自己的成本回报值.

3.2.1 博弈竞争策略

重构初始时按应用软件优先级Priority排列后,确定调度顺序为$D_1\gg D_2\gg\cdots\gg D_n$,即智能体$G_1,G_2,\cdots,G_n$,每个智能体$G_i$都有自己的博弈策略$\theta^{i-1}$,因此构成多智能体的策略矩阵$\boldsymbol{\theta}$(初始时为0矩阵):

$$\boldsymbol{\theta}=\begin{pmatrix}\theta_1^0 & \cdots & \theta_{m+1}^0\\ \vdots & \ddots & \vdots\\ \theta_1^{n-1} & \cdots & \theta_{m+1}^{n-1}\end{pmatrix}\tag{17}$$

从状态S_0开始,重构中每个智能体的调度动作是基于上一个智能体调度产生的新环境执行的,策略更新顺序为$\theta^0\gg\theta^1\gg\cdots\gg\theta^{n-1}$,所以智能体的策略之间是相互影响的. 并且每个智能体是单一独立且动作离散的,因此可以使用softmax分布对当前状态S和每个动作即每个待选分区设置一个偏好值θ_a^s,偏好值越大,被选中的概率就越大.

因此,策略π就是一个关于偏好函数θ_a^S的一个函数. 即

$$\Pr(A=a|s)=\frac{e^{\theta_a^S}}{\sum_{b=1}^{m+1}e^{\theta_b^S}}=\pi(S,a) \tag{18}$$

$\Pr(A=a|S)$表示在状态S下选择动作a的概率,即在状态S下选择动作a的策略. 初始时,所有的偏好值均为0,即开始时每个动作被选取的概率一样. 因此,有

$$\begin{aligned}\frac{\partial\pi(S,a)}{\partial\boldsymbol{\theta}}&=\frac{\partial\pi(S,a)}{\partial\theta^s}=\frac{\partial}{\partial\theta^s}\left[\frac{e^{\theta_a^S}}{\sum_{b=1}^{m+1}e^{\theta_b^S}}\right]\\&=\frac{\frac{\partial e^{\theta_a^S}}{\partial\theta^s}\sum_{b=1}^{m+1}e^{\theta_b^S}-e^{\theta_a^S}\frac{\partial\sum_{b=1}^{m+1}e^{\theta_b^S}}{\partial\theta^s}}{\left(\sum_{b=1}^{m+1}e^{\theta_b^s}\right)^2}\\&=\frac{\mathbb{Z}_{a=b}^S e^{\theta_a^S}\sum_{b=1}^{m+1}e^{\theta_b^S}-e^{\theta_a^S}e^{\theta^S}}{\left(\sum_{b=1}^{m+1}e^{\theta_b^S}\right)^2}\\&=\mathbb{Z}_{a=b}^S\pi(S,a)-\pi(S,a)\pi(S)\\&=\pi(S,a)(\mathbb{Z}_{a=b}-\pi(S))\end{aligned} \tag{19}$$

其中,$\mathbb{Z}_{a=b}^S$表示在状态S下的一个示性函数,$\mathbb{Z}_{a=b}^S=\begin{cases}1, & \text{if } a=b\\0, & \text{other}\end{cases}$.

竞争策略的迭代函数H_1:本文中,强化学习策略梯度主要用于智能体竞争策略的更新,策略矩阵θ中每一行向量为每个智能体对应的策略空间. 由式(19)定义优化的成本回报函数作为博弈竞争的优化目标. 这里定义每一智能体的平均回报,即第N轮博弈的第G_i个智能体时,策略π_N^i下的累计回报$\rho(\pi)$:

$$\begin{aligned}\rho(\pi)&=\lim_{N\to\infty}\frac{1}{N}E\{r_1^S+r_2^S+\cdots+r_N^S|\pi\}\\&=\sum_S d^\pi(S)\sum_a\pi(S,a)r^S\end{aligned} \tag{20}$$

其中,$d^\pi(S)$是基于策略π生成的马尔可夫链关于状态的静态分布,即从$S_0\gg S_1\gg\cdots\gg S_{n-1}$代表智能体$G_1,G_2,\cdots,G_n$调度时对应的第1个应用软件到第$n$个应用软件所面临的状态;$r_t^S(a)$为在$S$状态下第$t$次执行$a,a\in[1,1+m]$动作的回报值. 当$N$趋近于无穷时,$\overline{r^S}$为$\rho(\pi)$的无偏估计量. 随着$N$的增大,无偏估计量的计算时间也增大,为减少计算耗时,可以以一定的比例抽取历史池. 这里使用有偏估计量$\overline{r^S}=\frac{1}{x}\operatorname{top}_x\{r_1^S+r_2^S+\cdots\}$(表示取$N$个$r$里最优的$x$个的平均值)作为$\rho(\pi)$有效的历史池,在牺牲全局最优性的同时,使事件概率分布尽量往历史最优方向靠,增加收敛性.

设存在一个在s状态下依策略π对动作a的期望评价值$Q^\pi(S,a)$. 则有

$$Q^\pi(S,a)=\sum_{t=1}^{\infty}E\{r_t^S(a)-\rho(\pi)|\pi,a\in[1,1+m],S\in[0,n-1]\} \tag{21}$$

其中,$r_t^S(a)$代表第t轮博弈智能体在状态S下执行动作a的回报.

此时,有

$$\frac{\partial\rho(\pi)}{\partial\theta}=\sum_S d^\pi(S)\sum_a\frac{\partial\pi(S,a)}{\partial\theta}Q^\pi(S,a) \tag{22}$$

当确定状态S时,即对应单一智能体时,由式(19)(21)(22)有

$$\begin{aligned}\frac{\partial\rho(\pi,S)}{\partial\theta^s}&=\sum_a\frac{\partial\pi(S,a)}{\partial\theta^S}Q^\pi(S,a)\\&=\sum_a\frac{\partial\pi(S,a)}{\partial\theta^s}E\{r_t^S(a)-\rho(\pi)|\pi\}\\&=\sum_a(\mathbb{Z}_{a=b}-\pi(S))(r_t^S(a)-\overline{r^{'S}})\end{aligned} \tag{23}$$

详细证明过程可以参考文献[25,26].

则θ^S更新按梯度上升思想有

$$\theta^S(t+1)=\theta^S(t)+\alpha\frac{\partial\rho(\pi)}{\partial\theta} \tag{24}$$

即在s状态下,在选择动作a并获得收益r_t^S后,策略矩阵的偏好值更新方式为

$$H_1:\begin{cases}\theta_a^S(t+1)=\theta_a^S(t)+\alpha(r_t^S(a)-\overline{r^{'S}})(1-\pi(S,a))\\\theta_{b\neq a}^S(t+1)=\theta_{b\neq a}^S(t)-\alpha(r_t^S(a)-\overline{r^{'S}})\pi(S,b)\\\text{s.t. }\alpha\in[0,1]\end{cases} \tag{25}$$

3.2.2　博弈竞争策略更新

本文采用策略梯度算法更新智能体策略,原因是它能够直接优化策略的期望回报,并以循环更新的方式直接在策略空间中迭代最优策略. 在对系统状态环境与对应动作所可能产生的结果一无所知的情况下,使用策略梯度可以快速地让动作策略向趋近于期望方向更新. 博弈竞争策略的更新如图4所示.

智能体G_i在第t轮博弈下基于自我的策略π_t选择动作a_t,回报函数根据策略和状态对动作计算回报值并根据策略迭代函数H_1进行更新策略,接着状态S_{i-1}对

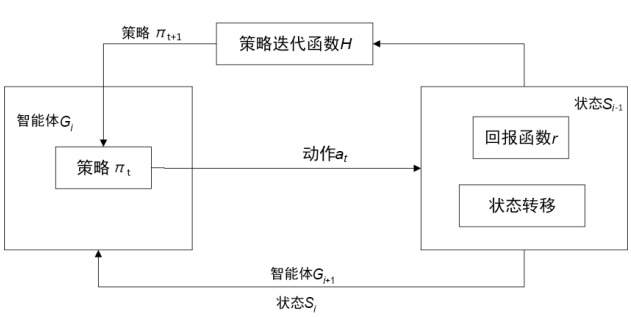

图4　博弈策略更新流程示意图

智能体做出的动作进行状态转移，生成新的状态S_i，并传递给下一个智能体G_{i+1}. 竞争策略更新如算法1所示.

算法1 博弈竞争策略更新算法

输入:初始化或上轮博弈后的策略矩阵$\boldsymbol{\theta}$

输出:策略矩阵$\boldsymbol{\theta}$

FOR $t=0$ to N:

  FOR S in $\boldsymbol{\theta}$,$\boldsymbol{\theta}=\{\theta^0,\theta^1,\cdots,\theta^{n-1}\}$:

    FOR a in $\{\theta_1^S,\theta_2^S,\cdots,\theta_{1+m}^S\}$:

      $\Pr_a=\pi(S,a)$

  END FOR

  依概率$\{\Pr_a$ in $S\}$选取动作a,$a\in[P_1,P_{1+m}]$

  依动作由式(16)计算此次动作的$r_t^S(a)$

  IF len$(\overline{r^{tS}})<x$

  $\overline{r^{tS}}=\dfrac{1}{t+1}\{r_1^S+r_2^S+\cdots+r_t^S+r\}$

  ELSE

  $\overline{r^{tS}}=\dfrac{1}{x}\underset{x}{\text{top}}\{r_1^S+r_2^S+\cdots+r_t^S+r\}$

  由式(25)更新θ^s

  $\theta_a^S(t+1)=\theta_a^S(t)+\alpha(r_t^S(a)-\overline{r^{tS}})(1-\pi(S,a))$

  $\theta_{b\neq a}^S(t+1)=\theta_{b\neq a}^S(t)-\alpha(r_t^S(a)-\overline{r^{tS}})\pi(S,b)$

  END FOR

END FOR

3.3 博弈合作模型

当重构的所有智能体都调度结束后，系统对指标综合的评价将作为智能体们的奖励，它们共用一个奖励历史经验池. 若智能体只是贪婪地竞争自己的最优策略，经过一定博弈轮次的迭代后，基于有偏估计策略梯度迭代后的策略空间将适于每一个智能体各自的最优条件. 但是，单一智能体的贪婪并不代表所有智能体的动作集合后的最终系统状态指标的综合评价是最优的. 因此，需要引入智能体的合作模型，使得智能体在竞争中亦能平衡自己成本与奖励的相互权重，直至混合纳什均衡，达到多目标优化的效果. 系统对指标的综合评价越高，智能体对应的奖励回报值就越高.

当智能体一轮博弈结束后，对目标函数2进行转化有

$$\begin{aligned}&\text{Ret}=\lambda_1\text{LB}(P,I,X)+\lambda_2\text{Im}(De,I)+\lambda_3T_{\text{re}}+\lambda_4\text{Sacrif}\\&\text{s.t.}\quad\begin{cases}0<\lambda_i<1,i\in[1,4]\\\lambda_1+\lambda_2+\lambda_3+\lambda_4=1\end{cases}\end{aligned}\tag{26}$$

因此，在博弈的合作模型中，通过博弈轮次结束后系统指标评价的奖励回报值和对应的历史经验池，智能体统一的对策略进行更新，最优化自己动作所得到的奖励回报值与成本回报值. 它们相互合作，每个智能体都在确保自己付出低成本的同时，获得更高的奖励回报值.

3.3.1 博弈合作策略更新

本文使用MCTS算法对多智能体最终评价的奖励回报值进行策略更新. MCTS作为一种经典的启发式策略搜索方法，被广泛用于游戏博弈问题中的行动规划[27]. 它基于对搜索空间的随机探索，利用探索结果在内存中建立了一个初始搜索树，并且在准确估计最有前途的动作值方面逐渐变得更好.

基于博弈合作的MCTS包括选举、模拟、拓展和反向传播4个步骤，更新过程如图5所示.

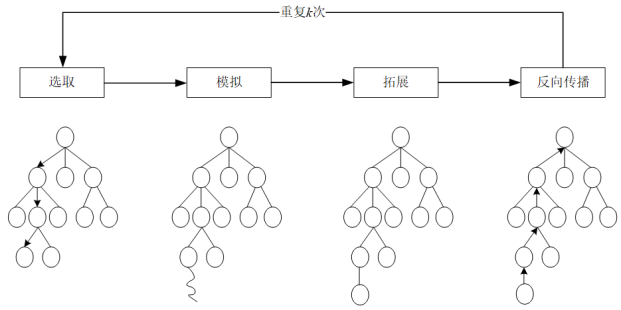

图5 MCTS流程示意图

初始状态下，所有智能体的成本回报经验池与奖励历史池均为空. 智能体按照重构的调度顺序进行序贯博弈，每个智能体依据策略空间不断模拟，模拟一定轮次后开始扩展，随着扩展的进行，对已扩展的智能体的策略以选举的方式选择动作. 每个轮次结束后都会计算相应的奖励回报值方向传播给所有的智能体. 直到所有智能体都没有欲望对策略进行更改，奖励回报值趋于平稳，达到混合纳什均衡，博弈结束.

设step$\in[0,\text{Num}]$代表已经博弈扩展的层数，通常设Num$=n$，n表示故障软件数.

(a)选举：智能体选择自己策略中最大的偏好值对应的分区作为调度的动作.

当step$\geqslant s$时，选取当前状态策略空间θ^S中概率最大值即值最大的偏好值θ_a^S作为要执行的动作. 否则，进行模拟. 当step$=0$时，第一层进行模拟.

(b)模拟：智能体依照策略空间的概率分布，依概率选择动作.

当step$<s$，当前所在的状态空间依策略空间θ^s概率计算后，依概率进行选择动作，直至到达最后一个状态的动作执行完后计算Ret值并保存进历史池. 使用有偏估计量$\overline{\text{Ret}'}=\dfrac{1}{x}\underset{x}{\text{top}}\{\text{Ret}_1,\text{Ret}_2,\cdots\}$，将行为概率对应分布的最大概率向最优概率靠近.

(c)扩展：确定该智能体是进行选举策略还是模拟策略.

在经过多次选举与模拟后，探索层数加一，即step$=$step$+1$，继续进行选举和模拟. 当扩展至最后一

层时,所有的智能体选择的策略都是选举最大偏好值作为调度的动作.

(*d*)反向传播:依据系统评价的奖励回报值,更新博弈过程中所有的智能体所选动作的对应策略偏好值. 未被选中的动作不更新,更新方式为

$$H_2:\begin{cases}\theta_a^S(t+1)=\theta_a^S(t)+\gamma(\mathrm{Ret}-\overline{\mathrm{Ret'}})(1-\pi(S,a))\\ s.t.\ \ \gamma\in[0,1],S\in\{0,1,\cdots,n-1\}\end{cases}\tag{27}$$

当step=0时,循环过程未进行选择步骤,只进行模拟,且保存模拟后的回报值,表示对空间的初步探索. 博弈合作策略更新如算法2所示.

算法2　蒙特卡洛策略梯度收敛策略空间

输入:经过算法1迭代后的策略矩阵$\boldsymbol{\theta}$

输出:策略空间$\boldsymbol{\theta}$

$T=\{\}$

FOR step=0 to Num:

  FOR $t=0$ to N:

    FOR s in S,$S=\{\theta^0,\theta^1,\cdots,\theta^{n-1}\}$:

    IF step$<s$:

      FOR a in $\{\theta_1^s,\theta_2^s,\cdots,\theta_{1+m}^s\}$:

        $\mathrm{Pr}_a=\pi(s,a)$

      依概率$\{\mathrm{Pr}_a$ in $s\}$选取动作A,$A\in[P_1,P_{1+m}]$

      END FOR

    ELSE :

      选择最大θ_a^S作为动作a,$a=P_a$,$T=T+(s,a)$

  END FOR

  循环结束后计算Ret,且

  $\overline{\mathrm{Ret'}}=\dfrac{1}{x}\underset{x}{\mathrm{top}}\{\mathrm{Ret}_1,\mathrm{Ret}_2,\cdots\}$

  FOR (s,a) in T:

  $\theta_a^S(t+1)=\theta_a^S(t)+\gamma(\mathrm{Ret}_t-\overline{\mathrm{Ret'}})(1-\pi(s,a))$

  END FOR

END FOR

3.4　基于序贯博弈的重构流程

基于序贯博弈的多智能体强化学习算法流程如图6所示,在基于应用软件的优先级确定智能体博弈顺序后,进行多智能体序贯竞争博弈,目的是尽可能地降低自己动作的成本,在重构结束后,进行多智能体合作博弈,目的是尽可能地提高自己所获得的奖励. 若是未满足终止条件,则重置重构状态,开启新一轮的序贯博弈,若是达到设定的终止条件,如达到最大博弈轮次或多智能体间达到混合纳什均衡状态,任何一个智能体都不再改变自己策略的时候,序贯博弈结束.

由于系统资源有限,若最终得到的博弈结果不存在应用软件由于资源有限,不得不调度失败而被牺牲的情况,则重构流程结束. 若存在,则对牺牲应用软件进行抢占式贪婪博弈. 若抢占后满足约束条件,则将替换后的应用软件放入重博弈序列继续尝试抢占,否则换个分区继续尝试,直至所有分区均抢占失败后放入重构牺牲集De. 待重博弈序列中所有的应用软件均抢占失败后,重构流程结束,如图7所示.

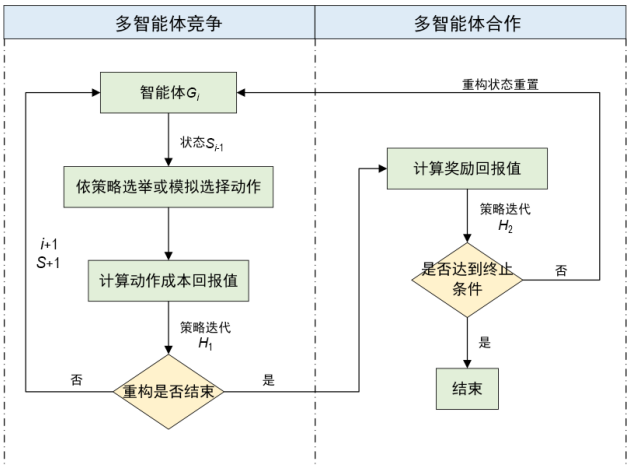

图6　序贯博弈算法流程

在发生故障后,会产生待重构调度的应用软件序列以及可重构调度的分区. 按优先级排序后将应用软

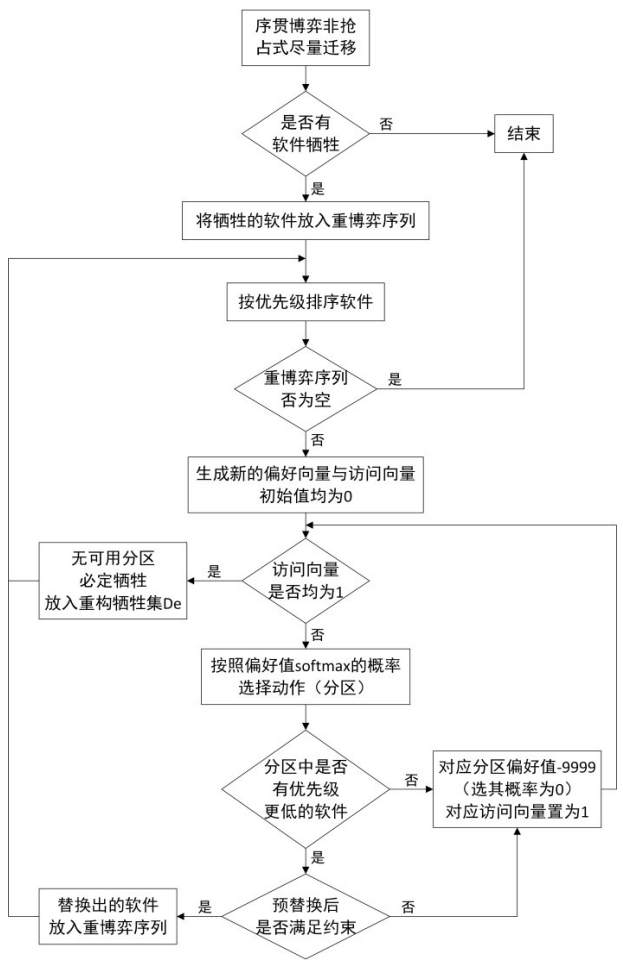

图7　重构流程

件序列转化为多智能体，它们在经过序贯博弈后生成的最优结果中，若存在由于资源限制而不得不牺牲的应用软件，则将牺牲的应用软件放入重博弈序列进行贪婪博弈. 每个被牺牲的应用软件都尽可能地尝试是否可以抢占原系统中状态正常的应用软件. 因为优先级越高的应用软件越应该被优先转移，所以待重构调度的应用软件序列需要按优先级进行排序.

4 实验与性能分析

系统硬件环境为CPUi7-7700、内存16GB，显卡GTX1070，软件环境为Windows10. 实验旨在分析智能重构算法生成高质量蓝图的能力. 首先，收集IMA系统的基本配置信息作为实验的输入数据，如表1、表2所示. 从初始配置状态开始，实验注入不同的处理器故障，产生不同的重构状态.

表1 应用软件属性数据

应用软件Id	截止时间/ms	最坏情况执行时间/ms	内存/kb	优先级
M_1	15	2	20	3
M_2	40	6	90	4
M_3	40	8	90	4
M_4	30	5	80	5
M_5	20	4	80	2
M_6	40	6	70	2
M_7	30	4	70	4
M_8	20	3	60	2
M_9	20	2	60	3
M_{10}	30	4	50	4
M_{11}	15	2	50	3
M_{12}	30	2	40	1
M_{13}	40	3	40	3
M_{14}	30	4	30	2
M_{15}	15	1	30	4

图8描绘了所采用的重新配置状态的转变，每个节点代表系统故障后的重构状态. 根据智能重构的8个环境迁移情况，对故障情形进行模拟，E_0的初始配置信息如表2所示，E_1代表在E_0的环境下C_1发生故障；E_3代表在E_1的环境下C_2发生故障；E_5代表在E_0的环境下C_2、C_3发生故障；E_7代表在已经发生C_4故障的环境E_2下C_2、C_5发生故障.

4.2 实验参数

根据实验环境设置的IMA系统初始环境状态与所设8个不同的故障环境，设置实验的基本参数. 实验参数如表3所示，故障应用软件数n与可用分区数m随着8个故障环境的转换而改变. 最大博弈次数与有偏估计长度则随着n与m的改变自适应变换. 博弈中竞争与

表2 IMA初始E_0配置信息

硬件资源	主框架时间/ms	分区	内存/kb	执行时间/ms	应用软件
C_1	50	P_1	128	15	M_{15}
		P_2	256	20	M_2
		P_3	128	15	M_{13}
C_2	50	P_1	512	50	M_3 M_4 M_{12}
C_3	50	P_1	256	30	M_6 M_7
		P_2	256	20	M_8
C_4	50	P_1	128	15	M_9
		P_2	128	10	/
		P_3	256	25	M_{10} M_{14}
C_5	50	P_1	256	30	M_1 M_5
		P_2	256	20	M_{11}

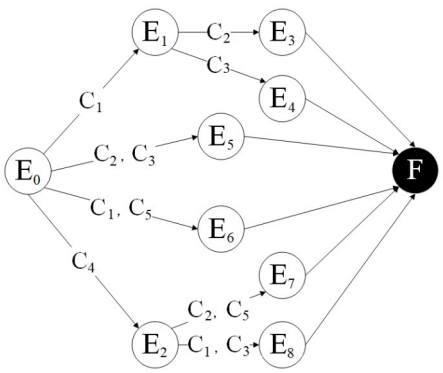

图8 智能重构的环境迁移

合作的学习率固定为0.01与0.9. 评价指标的参数与文献[17]中的一致.

表3 实验参数设置

参数	值
系统应用软件总数Nm	15
故障应用软件数n	3,3,6,6,6,6,9,9
可用分区数m	8,8,7,6,8,6,5,3
最大博弈次数N	100 nm
有偏长度x	1 nm
博弈竞争学习率α	0.01
博弈合作学习率γ	0.9
分区负载参数μ_1,μ_2	0.5,0.5
负载约束$C_{use-max}$,$RM_{use-max}$	0.8,0.8
评价指标参数$\lambda_1,\lambda_2,\lambda_3,\lambda_4$	0.1,0.35,0.35,0.2

4.3 算法性能分析

算法性能的对比将比较BE-PGMCTS、PGMCTS、Qlearn和DE四种算法的最大回报值和最大值收敛时间，分别在相同最大迭代次数下对每个算法重复训练100次，记录每次算法内部迭代情况、最大值和收敛时

间,积累后分别计算其100次的平均值. 最大回报值越大,说明算法生成的软件迁移策略即重构蓝图的优化效果越好,优化性能越强. 算法最大值的收敛时间越小,说明在相同故障环境下算法的收敛性能越好.

不同环境下不同算法具体的回报值探索曲线如图9所示,BE-PGMCTS在E_1~E_8八个故障环境中都能最早达到收敛,收敛性能最高. 而PGMCTS在E_1~E_4的简单故障环境下收敛时间比BE-PGMCTS略高一点,但是在E_5~E_8复杂环境下由于无偏估计计算耗时的增加,其最大值收敛时间也飞速增加,收敛性能不稳定. Qlearn由于自身抖动的因素收敛时间通常较长. DE算法每次迭代都要尝试种群内所有个体解对应的重构蓝图,因此最大值收敛时间通常最大. 具体数据如表4所示.

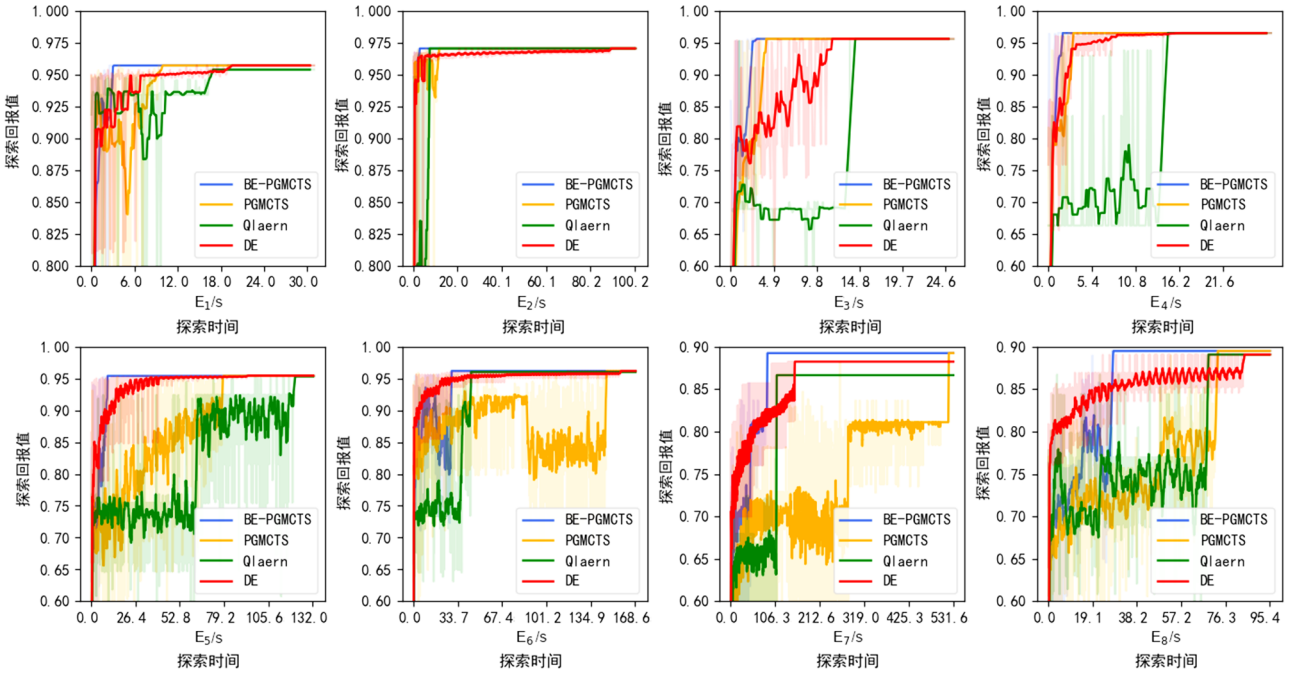

图9　不同算法的收敛性能对比

如表4所示,在资源充足的E_1~E_6故障环境下,BE-PGMCTS,PGMCTS和DE算法所求得的最大回报值基本相同. Qlearn虽然在单次迭代的算法时延最短,但是探索到的最大回报值往往无法与其他算法媲美,最大值的收敛时间也不占优. PGMCTS随着迭代次数增加,每次迭代时计算无偏估计的算法时延也增加,从而导致总体的算法收敛时延飞速增加. BE-PGMCTS由于有偏估计的计算耗时,其单次迭代的算法时延相比Qlearn会稍高一点,但是总体的算法收敛时间最小. 而DE算法每次迭代都要计算种群中所有个体的回报值,因此算法时延最高.

在因为资源限制而不得不抢占原系统资源的E_7、E_8故障环境下,DE和Qlearn算法无法保证进入贪婪博弈抢占模式时是最好的状态,因此往往无法找到更优的重构蓝图对应的回报值. 以E_8故障环境为例,BE-PGMCTS和PGMCTS算法重复100次的最大回报值平均值都在0.8949左右,而Qlearn和DE仅分别为0.8801和0.8858. 这意味着四种算法在因为资源限制而不得不贪婪抢占式博弈的情况下,Qlearn和DE更容易收敛于局部最优,而PGMCTS和BE-PGMCTS算法可以找到更好的软件迁移策略对应的重构蓝图.

综上所述,无论在资源充足还是资源不足的情况下,BE-PGMCTS都可以更为快速地得到最优回报值对应的重构蓝图,算法收敛时间最小,收敛的最大值最高,因此算法性能最高. Qlearn和DE收敛时间较长,容易陷入局部最优. PGMCTS在迭代次数递增的同时算法计算时延也会飞速提高. BE-PGMCTS对其做了改进,保证算法时延与迭代次数是一个递进的线性关系,在减少不必要的计算耗时的同时亦能保证算法的优化效果.

4.4　算法稳定性分析

由于在E_7,E_8复杂故障环境下四种算法所优化的最大回报值差距略为明显,说明在该环境下算法所求最大回报值并不稳定,存在一定的抖动情况. 因此,为检验算法的稳定性,重复100次训练,记录在E_7,E_8复杂故障环境下算法回报值迭代过程中的平均值和最大值的标准差,并计算它们在100次训练后的平均值. 其中,平均值为算法开始迭代到算法达到最大值过程的

表4 算法性能对比

故障环境	算法	最大值平均值	算法时延/(ms/次)	最大值收敛时间/s
E_1	BE-PGMCTS	0.957 2	4.93	2.474 8
	PGMCTS	0.957 2	18.60	8.686 2
	Qlearn	0.955 2	3.24	16.251 0
	DE	0.957 2	51.40	17.294 0
E_2	BE-PGMCTS	0.970 6	3.53	2.308 6
	PGMCTS	0.970 6	9.05	10.887 0
	Qlearn	0.970 6	3.83	6.736 9
	DE	0.970 6	61.60	45.707 0
E_3	BE-PGMCTS	0.956 2	3.46	2.283 6
	PGMCTS	0.956 2	8.94	3.272 0
	Qlearn	0.955 8	3.31	13.518 0
	DE	0.956 2	48.40	8.905 6
E_4	BE-PGMCTS	0.965 3	3.22	1.120 6
	PGMCTS	0.965 3	7.10	2.307 5
	Qlearn	0.965 2	2.96	13.897 0
	DE	0.965 3	36.60	14.274 0
E_5	BE-PGMCTS	0.954 1	5.87	8.893 1
	PGMCTS	0.954 1	38.10	74.486 0
	Qlearn	0.953 3	4.32	117.930 0
	DE	0.954 1	69.10	84.762 0
E_6	BE-PGMCTS	0.961 8	4.71	27.954 0
	PGMCTS	0.961 8	20.30	123.780 0
	Qlearn	0.960 2	4.14	41.806 0
	DE	0.961 8	50.70	153.690 0
E7	BE-PGMCTS	0.892 4	6.42	83.556 0
	PGMCTS	0.892 4	32.00	416.480 0
	Qlearn	0.866 4	4.30	105.380 0
	DE	0.882 3	42.70	108.500 0
E_8	BE-PGMCTS	0.894 9	5.79	26.512 0
	PGMCTS	0.894 9	18.30	66.411 0
	Qlearn	0.880 1	3.92	67.134 0
	DE	0.885 8	23.10	80.873 0

回报值平均值. 最大值标准差越低,说明算法稳定性越高,若综合考虑平均值,则可以看出算法在探索迁移策略时回报值的抖动情况,平均值越高,则越容易陷入局部最优,平均值越低,则抖动越明显. 具体数据如图10所示.

如图10所示,在E_7,E_8复杂故障环境下,BE-PGMCTS和PGMCTS算法的最大回报值最高,标准差最低,意味着它们的稳定性最高. 其中PGMCTS由于需要付出更多的计算时延去计算无偏估计,其最大回报值会比BE-PGMCTS的最大回报值略微高一点. DE算法的平均值最高,标准差略高,但是优化效果却不如BE-PGMCTS和PGMCTS,这意味着其在复杂环境下容易收敛于局部最优. Qlearn算法的标准差最高,平均值最低,说明其一直处于震荡状态很难收敛到最优的回报值. 综合来看,BE-PGMCTS和PGMCTS算法都是在改变动作概率分布的同时增加了最大蓝图指标出现的概率,探索性强,优化效果好,稳定性高.

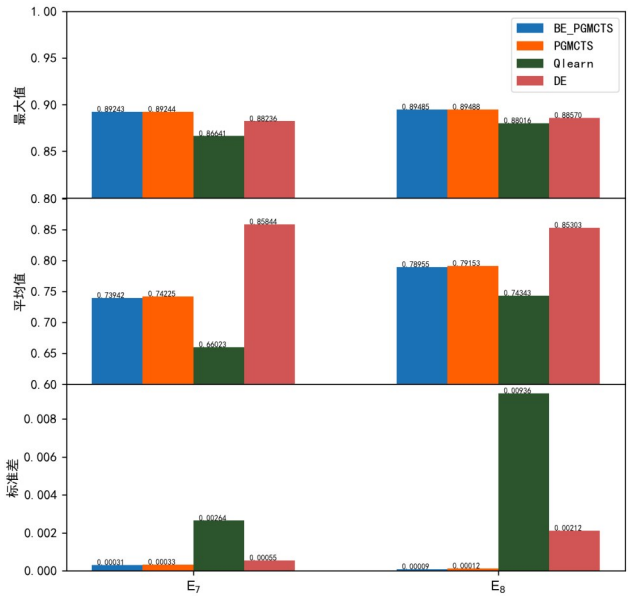

图10 不同算法的稳定性对比

5 结论

本文在IMA系统的重构方法中引入了基于序贯博弈多智能体强化学习的方法. 将重构中需要调度的应用软件定义成一个个单独的个体,定义其多智能体间的竞争目标与合作目标. 博弈策略的更新算法是在传统策略梯度算法的基础上提出基于有偏估计的策略梯度MCTS算法,解决了传统策略梯度算法震荡难收敛、计算耗时长等问题,便于多智能体在不断序贯博弈的同时可以快速地均衡博弈中竞争的成本与合作的奖励. 与传统的智能优化算法和强化学习算法作为对比,本文提出的算法有着更好的算法性能与算法稳定性.

参考文献

[1] PARR G R, EDWARDS R. Integrated modular avionics [J]. Air & Space Europe, 1999, 1(2): 72-75.

[2] 丁全心. 综合模块化航空电子系统标准述评[J]. 电光与控制, 2013, 20(6): 1-3.
DING X Q. Remarks on standards of integrated module avionic system[J]. Electronics Optics & Control, 2013, 20(6): 1-3. (in Chinese)

[3] PARTON D. Blueprint for the future[J]. Mental Health Today, 2011, 63(2): 10.

[4] JOLLIFFE G, NICHOLSON M. Exploring the Possibili-

ties Towards a Preliminary Safety Case for IMA Blueprints [M]. London, UK: Springer, 2005: 8.1-8.43.

[5] 王震,朱剑锋. 基于在线加载分区机制的重构方案的设计与实现[J]. 航空电子技术, 2016, 47(1): 6.
WANG Z, ZHU J F. Design and implementation of a reconfiguration blueprint based on online-loaded partition mechanism[J]. Avionics Technology, 2016, 47(1): 6. (in Chinese)

[6] BRIAO E W, BARCELOS D, WRONSKI F, et al. Impact of task migration in NoC-based MPSoCs for soft real-time applications[C]//2007 IFIP International Conference on Very Large Scale Integration. Virtual Conference: IEEE, 2007: 296-299.

[7] ANNIGHOEFER B , NIL C , SEBALD J , et al. Structured and symmetric IMA architecture optimization: Use case Ariane launcher[C]//IEEE/AIAA Digital Avionics Systems Conference. New York: IEEE, 2015: 6B3-1-6B3-14.

[8] 刘若辰, 李建霞, 刘静, 等. 动态多目标优化研究综述[J]. 计算机学报, 2020, 43(7): 1246-1278.
LIU R C, LI J X, LIU J, et al. A survey on dynamic multi-objective optimization[J]. Chinese Journal of Computers, 2020, 43(7): 1246-1278. (in Chinese)

[9] CALABOUGH J. Software configuration—an NP-complete problem[J]. ACM Sigmis Database, 1988, 19(2): 29-34.

[10] HOU X Y, GAO H B, DENG Z Q, et al. Path planning of lunar rover group based on theory of dynamic programming and multi-objective optimization[C]//IEEE Conference on Industrial Electronics & Applications. Virtual Conference: IEEE, 2007: 1308-1313

[11] 赵玉芳, 唐立新. 极小化总完工时间的单机连续型批调度问题[J]. 电子学报, 2008, 36(2): 367-370.
ZHAO Y F, TANG L X. Scheduling a single continuous batch processing machine to minimize total completion time[J]. Acta Electronica Sinica, 2008, 36(2): 367-370. (in Chinese)

[12] ZILINSKAS A, ZHIGLJAVSKY A. Branch and probability bound methods in multi-objective optimization[J]. Optimization Letters, 2016, 10(2): 341-353.

[13] SINGH H K, RAY T, SMITH W. C-PSA: Constrained pareto simulated annealing for constrained multi-objective optimization[J]. Information Sciences, 2010, 180 (13): 2499-2513.

[14] ZHANG J, SHANG Y, GAO R, et al. An improved multi-objective adaptive niche genetic algorithm based on pareto front[C]//2009 IEEE International Advance Computing Conference. Virtual Conference: IEEE, 2009: 300-304.

[15] LEI R, CHENG Y. A pareto-based differential evolution algorithm for multi-objective optimization problems[C]// 2010 Chinese Control and Decision Conference. New York: IEEE, 2010: 1608- 1613.

[16] ZHANG T, CHEN J, LV D, et al. Automatic generation of reconfiguration blueprints for ima systems using reinforcement learning[J]. IEEE Embedded Systems Letters, 2021, 13(4): 182-185

[17] 罗庆, 张涛, 单鹏, 等. 基于改进Q学习的IMA系统重构蓝图生成方法[J]. 航空学报, 2021, 42(8): 525792-525792.
LUO Q, ZHANG T, SHAN P, et al. Generating reconfiguration blueprints for IMA systems based on improved Q-learning[J]. Acta Aeronautica et Astronautica Sinica, 2021, 42(8): 525792-525792. (in Chinese)

[18] HUANG R T, YU T Y, DING Z H, et al. Deep Reinforcement Learning[M]. Singapore: Springer, 2020: 161-212.

[19] HE Q, HOU X.WD3: Taming the estimation bias in deep reinforcement learning[C]//2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI). Virtual Conference: IEEE, 2020: 391-398.

[20] CHASLOT J B, WINANDS M, HERIK H, et al. Progressive Strategies for Monte-Carlo Tree Search[J]. New Mathematics & Natural Computation, 2008, 4(3): 343-357.

[21] PATTERSON W, WINSTON-PROCTOR C E. Game Theory[M]. Part of the Springer Undergraduate Mathematics Series book series(SUMS). London: Springer, 2020: 87-106.

[22] MEZZETTI C, RENOU L. implementation in mixed nash equilibrium[J]. Warwick Economics Research Paper, 2012, 147(6): 2357-2375.

[23] KRISHNAMURTHY V, ABAD F. Gradient Based Policy Optimization of Constrained Markov Decision Processes[M]. Singapore: World Scientific, 2012: 503-547.

[24] POURMOHSENI B, WILDERMANN S, GLA M, et al. Hard real-time application mapping reconfiguration for NoC-based many-core systems[J]. Real-Time Systems, 2019, 55(2): 433-469.

[25] SUTTON R S, MCALLESTER D, SINGH S, et al. Policy gradient methods for reinforcement learning with function approximation[C]//Submitted to Advances in Neural Information Processing Systems(NIPS). Vertual Conference: The MIT Press, 1999: 1057-1063.

[26] SUTTON R S, BARTO A G, et al. Reinforcement Learning: An Introduction Second edition[M]. London: The

MIT Press, 2015: 265-278.
[27] SOEMERS D, PIETTE R, STEPHENSON M, et al. Learning Policies from Self-Play with Policy Gradients and MCTS Value Estimates[C]//2019 IEEE Conference on Games(CoG). Virtual Conference: IEEE, 2019: 1-8.

作者简介

张 涛 男,1976年出生,陕西扶风人. 博士,现为西北工业大学软件学院副教授. 主要研究方向为智能机器人、人工智能、强化学习和软件测试.
E-mail: tao_zhang@nwpu.edu.cn

张文涛 男,1995年出生,福建宁德福安人. 现为西北工业大学软件学院硕士研究生. 主要研究方向为强化学习、进化计算和优化调度.
E-mail: 820132512@qq.com