MEF:基于在线容错的云计算资源调度方法

都繁杰[1] 李 静[1] 王 亮[2] 夏 天[2]
[1](南京航空航天大学计算机科学与技术学院 江苏 南京 211106)
[2](国网上海市电力公司信息通信公司 上海 200000)

摘 要 云计算系统效率的关键是提升资源利用率与性能,系统可靠性的关键是备份组件。针对云环境下任务完成时间最小化、容错成本优化问题,提出一种基于在线容错的云计算资源调度方法。在线容错具有静态、动态的属性。静态容错根据组件可靠性和完成过程可靠性改进 LeaderRank 算法备份关键组件;动态容错在故障时快速替换故障组件;通过 MEF 算法实现任务完成时间短、容错成本低的多目标优化。实验表明,与现有容错方法对比,MEF 算法不仅降低了容错费用,而且增强了故障时的系统效率。

关键词 云计算 在线容错 多目标优化 资源调度 云组件排名

中图分类号 TP302 **文献标志码** A DOI:10.3969/j.issn.1000-386x.2024.03.051

MEF: CLOUD RESOURCE SCHEDULING METHOD BASED ON ONLINE FAULT TOLERANCE

Du Fanjie[1] Li Jing[1] Wang Liang[2] Xia Tian[2]
[1](*College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, Jiangsu, China*)
[2](*Information and Communication Company of State Grid Shanghai Electric Power Company, Shanghai 200000, China*)

Abstract The key to cloud computing system efficiency is to improve resource utilization and performance, and the key to system reliability is backup components. In order to minimize task completion time and optimize fault-tolerant cost in cloud environment, an online fault-tolerant resource scheduling method for cloud computing is proposed. Online fault tolerance has static and dynamic properties. The LeaderRank algorithm was improved to backup key components according to component reliability and completion process reliability. The dynamic fault tolerance could quickly replace the fault components when the fault occurs. The MEF algorithm was used to achieve the multi-objective optimization with short task completion time and low fault tolerance cost. Experimental results show that compared with the existing fault-tolerant methods, MEF algorithm not only reduces the cost of fault-tolerant, but also enhances the system efficiency.

Keywords Cloud computing Online fault tolerance Multi-objective optimization Resource scheduling Cloud component

0 引 言

随着云计算、物联网、5G、大数据,以及人工智能的不断普及和广泛应用,通过新一代信息化技术不断实现各类资源的整合与共享,逐步形成一种全新的大规模复杂云系统。由于参与计算的节点种类多样、位置分布稀疏且通常无法有效控制,容易产生安全问题;并且云服务供应商在传输、处理和存储的过程中均存在网络拥塞的风险,如何提高系统容错能力、实现高效快速计算成为新的挑战。

备份技术是一种广泛使用的容错技术,已有许多云计算服务商使用备份技术来提供不同级别的容错保障,OpenStack 云平台将文件和对象分布在不同服务器上,使用数据备份来保证数据的可靠性;在 Hadoop 的 EBS 等 DFS 系统中使用资源复制来保证系统的可靠

收稿日期:2020-11-04。国家电网公司总部科技项目(SGSHXT00JFJS1900093)。**都繁杰**,硕士生,主研领域:云计算。**李静**,副教授。**王亮**,工程师。**夏天**,工程师。

性。虽然应用备份技术可保证云计算的可靠性,但是其对计算、存储资源需求相对较高,并且如何解决备份的最优放置问题,是保证最佳容错效果的关键。除了备份技术外,当任务执行失败后,需要调用不同资源重新进行调度,以保证任务顺利完成。同时,针对云数据中心海量规模的用户访问与请求,需要在确保系统可靠性的前提下,缩短用户完成时间并保证用户服务质量 QoS。

综上所述,在资源调度时,大多数调度方法仅从其控设备获取组件信息,没有考虑监控设备的成本;随着云组件数量剧增,相应的备份成本会造成巨大的开支。然而,减少监控组件的数量,调度过程发生故障,会增加容错时间,使得任务的完成时间相应增加。为了解决这样的多目标优化问题,本文提出一种基于在线容错的云计算资源调度方法。

在线容错类似人体免疫系统,人体免疫系统包括非特异性免疫和特异性免疫,非特异性免疫针对多种病原体起作用,具有普适性;特异性免疫针对特定病原体起作用。针对云计算系统的组件,根据其功能进行分类,对于承载关键功能的关键组件进行非特异性免疫,对于非关键组件进行特异性免疫,因此在线容错分为静态容错与动态容错。首先对各个组件建立 NHPP 模型,静态容错根据组件状态筛选出位置关键、可靠性强的两类组件,位置关键组件进行备份;发生故障时,动态容错算法在两个关键位置中选择其他节点进行替换。把用户完成时间以及容错等指标使用评价函数模型联合在一起;最后根据 MEF 算法求解系统的最大评价值。研究表明,基于在线容错的云计算资源调度机不仅确保大规模计算系统的可靠性和安全保障,也进一步提高调度效率,做到多指标联合优化。

本文的主要贡献包括:

(1) 提出了一种在线容错算法,包括静态容错与动态容错。静态容错分析云计算组件结构关键性以及本身可靠性,初步筛选出位置关键、可靠性好的两类云计算组件,对关键位置组件进行主备备份;动态容错用于发生故障后,将故障组件之间的关键组件作为开始与结束的节点,重新筛选出可靠性高的组件,继续执行任务,达到容错的目标。

(2) 将用户的完成时间以及容错作为优化目标,使用线性加权法进行多目标优化,得到基于多目标优化的评价函数 E。将多目标问题转化为单一目标问题,有利于求解效率的提高。

(3) 设计最大评价值优先算法 MEF 算法求解系统的最大评价值。PSO 根据静态容错对种群初始化,PBIL 算法进一步寻找最优分配。若发生故障,PBIL 算法根据动态容错进行再次调度。通过两次调度,得到用户完成时间短、可靠性高资源调度。

1　相关工作

云计算领域容错方法,分为反应性方法和主动性方法。主动容错在实际问题发生前进行检测,预测故障并替换可疑组件。反应式容错:当故障发生时,反应性容错可减少故障对应用程序执行的影响。以更少的成本获得更好的容错效果,即需要一种方法来识别云计算组件中的关键服务。FTCloud 通过 PageRank 算法,选择重要的节点来部署离线容错。云中实时任务的基于 PSO 的高效节能容错静态调度算法,提出了一种基于粒子群优化的高效节能容错静态调度算法。PSO-CFTD 从系统结构的可靠性分析,部署备份,是一种自适应机制来选择适当的计算资源。这些方法属于静态容错方式,仅仅针对关键组件的备份。通过动态节点替换实现高效的容错,用健康节点快速替换故障节点,是动态容错方法。一种新颖的云计算容错任务调度算法,采用主/副版本技术提供容错机制,通过离散粒子群优化算法对截止时间内执行时间、执行成本进行多目标优化。面向云应用系统的容错,从云应用组件的可靠性及响应时间等方面描述云应用容错需求,给出可用容错即服务的最优化求解方法。以上两种算法都没有针对底层资源进行可靠性建模,缺少对各个组件的评估。一种动态节点替换算法 DNR,该算法通过可模制和延展性作业的灵活性来查找替换组件,用健康组件快速替换故障组件,当故障较多时该方法性能变差。

针对在线容错的多目标调度问题,考虑容错以及用户完成时间多个目标,达到容错效果好,用户完成时间短的目标。PSO 被扩展用于任务的分配,但没有分析动态容错的,所以要设计新的调度算法来实现在线容错。鉴于此,本文提出了一种在线容错算法,包括静态容错与动态容错。静态容错初步筛选出关键组件进行备份;动态容错在发生故障后,用健康组件替换故障组件继续执行任务;将用户完成时间与容错作为优化目标,进行多目标优化,得到评价函数 E。设计最大评价值优先算法(MEF 算法)求解系统的最优分配策略。

2　系统框架

云系统中,用户通过 Map/Reduce 机制,统计为不

同的任务集合 $Task_1, Task_2, \cdots, Task_n$，与云组件集合 $Cc_1, Cc_2, \cdots, Cc_m$ 进行匹配。在线容错考虑调度过程中云计算组件故障，旨在以最低成本，完成用户提交的任务，做到系统成本低，用户完成时间短的目标。首先通过静态容错筛选关键组件（比如一个支付系统中负责支付功能的组件），对其备份，避免对全部组件进行备份，节约成本；系统发生故障时，寻找可靠性高的组件进行替换，及时完成用户的任务。

上述过程，发生了多次调度。静态容错强调对关键组件进行备份，为第一次调度，即根据用户的任务，分配相应的组件，对关键组件备份，为资源调度提供蓝图；云系统执行期间，若发生故障，可判断是非关键组件故障（若系统发生故障，关键组件的备份组件立即运行，故是非关键组件故障，这里关键组件每天都会检测，故关键组件不会发生全部故障的可能），静态容错无法发挥作用，需要进行再次调度。通过动态容错找到替换组件，使任务正常运行，达到容错的目的。随着云计算用户、规模变大，任务复杂性增加，求解时间变长，用户完成时间变长；发生故障时，寻找替换组件，进行再次调度，这个过程中偏向于找可靠性高、迁移成本低、系统可用性更好的组件。为了解决在线容错产生的求解时间变长、寻找替换组件困难的问题，设计评价函数 E，将用户完成时间 T 以及组件的替换成本作为目标，使用线性加权法进行多目标优化，简化求解目标，从理论上缩短计算量；其次，对于多次调度，仅采用一种调度算法不能快速、高效求解，故需要设计新的调度算法。设计最大评价值优先算法（MEF 算法），将评价函数 E 作为适应度函数，将 PSO 与 PBIL 算法进行结合，PSO 对种群进行初始化，初步确定求解范围，接着采用 PBIL 算法，从宏观控制种群进化方向，进而更快得到最优分配 $\{<Task_1, Cc_1>, <Task_2, Cc_2>, \cdots, <Task_n, Cc_m>\}$ 以及相应的评价值，实现多目标优化的资源调度。调度流程如图 1 所示。

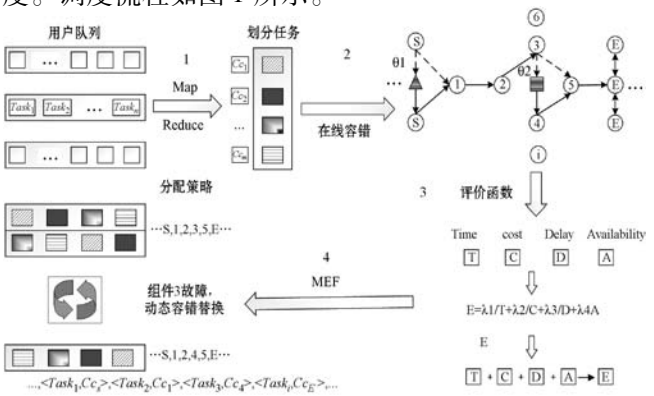


图 1 基于在线容错的云计算资源调度框架图

3 基于在线容错的多目标优化函数

静态容错通过对关键组件备份，可以实现用较低的成本达到较好的容错效果，完成初步调度；调度过程中发生故障，则可以初步判断是非关键组件故障，再次调度，调用动态容错，在两个关键组件之间寻找替换组件，缩短用户完成时间，保障任务顺利进行。整个过程中，用户完成时间 T、动态容错成本 C、系统延迟 D，以及系统可用性 A，共同影响资源调度的效率。

3.1 在线容错

假设组件故障率服从故障率不变的均匀泊松分布，但组件使用时间，频率不同，其故障模型也有差异。关键组件使用时间长、频率高，障率高；非关键组件使用频率低，其模型应与关键组件不同。根据组件长时间运行错误的发生率不是一个恒定的量而是随时间 t 变化的函数，在线容错对以上两种组件分别建立 NHPP 模型（非齐次泊松过程模型）来描述其累计故障数。建立的 NHPP 可靠性模型如图 2 所示。

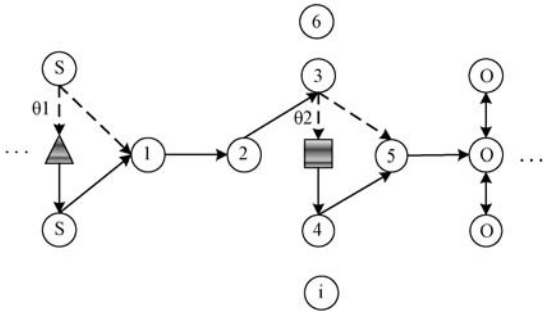


图 2 NHPP 模型

一般性失效故障的期望函数 $m(t)$ 性质如下：

$$\begin{cases} m(0)=0 \\ \lim\limits_{t\to\infty} m(t)=a \end{cases} \tag{1}$$

根据以上假设，可以得到：

$$m(t+\Delta t)-m(t)=\theta_1[a-m(t)]\Delta t+o(t) \tag{2}$$

$$m(t)=a(1-e^{-\theta_1 t}),\ a>0,\ \theta_1>0 \tag{3}$$

$$R_{x_i}(t\mid t_{i-1})=\exp\{-a[e^{-\theta_1 t}-e^{-\theta_1(t+t_{i-1})}]\} \tag{4}$$

其中：a 表示云组件总故障数，$\{x_i, i=1,2,\cdots\}$ 表示故障间隔，R 表示第 i 次失效发生后云组件可靠性。

同理，云组件发生严重性失效时，$m(t)=a(1-(1+\theta_2 t)e^{-\theta_2 t})$，同样，用 $\{x_i, i=1,2,\cdots\}$ 表示故障时间间隔，其云组件可靠性函数 R 如下：

$$R_{x_i}(t\mid t_{i-1})=\exp\left\{\begin{array}{l}-a[(1-(1+\theta_2 t)e^{-\theta_2 t} \\ -(1-(1+\theta_2 t+t_{i-1}))e^{-\theta_2(t+t_{i-1})}]\end{array}\right\} \tag{5}$$

完成过程可靠性 Pt 定义如下：

$$Pt = \frac{e^{-\lambda t}}{L_n \times S_n} \tag{6}$$

式中:L_n表示链接到云组件的设备数量;S_n表示过程结构类型的数量,复杂结构导致更高的故障率,因此这两个参数与过程可靠性负相关;$e^{-\lambda t}$是时间加权指数,表示过程可靠性随时间的下降程度。

静态容错考虑组件本身可靠性 R 以及完成过程可靠性 Pt,采用改进的 LeaderRank 算法对组件进行排名,选出关键组件。假设有 n 个云计算组件,迭代矩阵 $\boldsymbol{M}$ 是 n 维方阵:

$$M_i^j = w_i^j \times \frac{R}{Pt} \tag{7}$$

式中:M_i^j表示组件 i 到 j 的转移概率;w_i^j是邻接矩阵,表示 i 到 j 的权重。背景节点是保证 LeaderRank 算法收敛的关键,其迭代矩阵如下:

$$M_{i,n+1} = \frac{1}{n}, M_{n+1,i} = \frac{2\sum_{i=1}^{n} M_i^j}{\sum_{i=1}^{n}\sum_{j=1}^{n} M_i^j} \tag{8}$$

静态容错算法如算法 1 所示。

算法 1　静态容错算法

输入:组件转移矩阵 $\boldsymbol{M}$。

输出:组件的重要程度 $SR_i(v)$。

1　随机分配各组件重要程 $SR_i(0)\, i = 1, \cdots, n+1$。

2　Loop:

3　更新 $SR_i(t) = \sum_{j=1}^{n+1} \frac{M_{j,i}}{\sum_{k=1}^{n+1} M_{j,k}} \times SR_i(t-1)$。

4　until: $\exists \varepsilon \to 0, |SR_i(t) - SR_i(t-1)| < \varepsilon$。

5　分配背景组件 $SR_{n+1}(v)$给所有组件。

6　$SR_i(v) = SR_i(t) + \sum_{j=1}^{n+1} \frac{M_{n+1,i}}{\sum_{k=1}^{n} M_{n+1,k}} \times SR_{i+1}(t)$。

7　return 各组件重要程度 $SR_i(v), i = 1, \cdots n+1$。

动态容错:对于严重性失效,多发生在云组件执行过程。故障产生时,在两个关键组件之间,寻找替换服务组件,可以达到优良的容错效果,其实质是一种查找替代组合的算法。通过采用启发式替代路径搜索算法,将贪婪函数和松弛函数作为启发式函数求解,以找到接近最优的路径。动态容错算法数学符号如表 1 所示。

表 1　动态容错基本符号表

数学符号	含义	数学符号	含义
D	系统延迟	D_{m}	故障迁移延迟
D_i^{s}	软件响应延迟	D_i^{h}	硬件响应延迟
$D_{i,i+1}^{\mathrm{C}}$	网络通信延迟	D_{d}	故障检测延迟
C	容错成本	C_{m}	故障迁成本
C_{cp}	计算成本	C_{lb}	负载均衡率
A	系统可用性	a_i^j	组件 i 可用性
D_{cons}	延迟时间 QoS 要求	C_{cons}	容错成本 QoS 要求
A_{cons}	系统可用性 Qos 要求	T_{cons}	用户完成时间 QoS 要求
U_i^j	在第 j 个时间内服务 i 可用的次数		
N_i^j	i 在第 j 个周期中执行服务的次数		
T	用户完成时间		

云服务过程中系统延迟 D 如下:

$$D = D_{\mathrm{d}} + D_{\mathrm{m}} + \sum_{i=1}^{n} (D_i^{\mathrm{s}} + D_i^{\mathrm{h}}) + \sum_{i=1}^{n-1} D_{i,i+1}^{\mathrm{C}} \tag{9}$$

容错成本 C 如下:

$$C = \sum_{i=1}^{n} (C_{\mathrm{m}}^i + C_{\mathrm{cp}}^i + C_{\mathrm{lb}}^i) \tag{10}$$

系统可用性 A,指制造服务可以在特定检查时间正常执行的可能性,表示如下:

$$\begin{cases} A = \prod_{i=1}^{n} A_i^{m+1} \\ A_i^{j+1} = A_i^j + a_i^j \\ A_i^1 = 0, a_i^j = \dfrac{U_i^j}{N_i^j} \end{cases} \tag{11}$$

动态容错是找到一种最佳的服务组合,替换发生故障的服务,并同时满足 QoS 约束。可以将问题抽象为0 - 1 整数约束的多目标优化问题,其公式如下:

$$\begin{cases} \min\ T_{1\to n}, D_{1\to n}, C_{1\to n}, \max\ A_{1\to n} \\ 0 < T_{1\to n} < T_{\mathrm{cons}} \\ 0 < D_{1\to n} < D_{\mathrm{cons}} \\ 0 < C_{1\to n} < C_{\mathrm{cons}} \\ A_{\mathrm{cons}} < A_{1\to n} < 1 \end{cases} \tag{12}$$

式中:T_{cons}、D_{cons}、C_{cons}、A_{cons}是为保证用户 QoS 需求的最低标准,不可能为 0。正式引入贪婪函数 G:

$$G(P_{i\to j}) = \min\left\{\frac{T_{i\to j}}{T_{\mathrm{cons}}}, \frac{D_{i\to j}}{D_{\mathrm{cons}}}, \frac{C_{i\to j}}{C_{\mathrm{cons}}}\right\}, \max\left(\frac{A_{i\to j}}{A_{\mathrm{cons}}}\right) \tag{13}$$

以及松弛函数 H:

$$H(P_{i\to j}) = \frac{R_{x_i}}{F_i}\begin{bmatrix} \dfrac{T_{\mathrm{cons}}}{T_{i\to j}} & \dfrac{D_{\mathrm{cons}}}{D_{i\to j}} & \dfrac{C_{\mathrm{cons}}}{C_{i\to j}} & \dfrac{A_{i\to j}}{A_{\mathrm{cons}}} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} \tag{14}$$

动态容错算法流程如算法 2 所示。

算法 2 动态容错算法

输入:各组件参数:D_d,D_m,D_i^s,D_i^h,$D_{i,i+1}^C$,C_m,C_{cp},C_{lb},A_i^{m+1},D_{cons},C_{cons},A_{cons},T_{cons}。

输出:关键组件的最优路径 $P_{s\to o}$,P_s表示开始关键节点,P_o表示结束关键节点,以及该分配情况的松弛值 $H(P_{s\to o})$。

1 初始化每个组件的松弛函数值 $H(P_{s\to o})=0$。

2 使用贪婪函数 $G(P_{s\to o})$作为求解目标,从开始关键节点到终止关键节点寻找最优路径。

3 If $G(P_{s\to o})>1$ then

4 找到的路径未满足 QoS 要求。

5 Else

6 使用松弛函数 $H(P_{s\to o})$寻找最优路径。

7 执行向前搜索,找到最优路径 $P_{s\to o}$。

8 End

9 return 最优路径 $P_{s\to o}$,该分配情况的松弛值 $H(P_{s\to o})$

再次回到图 2 的 NHPP 可靠性模型中,组件{S、O}是静态容错算法确定的关键组件,对其进行备份,发生故障时,切换备份组件,实现静态容错;组件{1,2,…,n}表示一般组件,其中组件 3 发生故障,为了不影响用户任务动进行,动态容错寻找满足 QoS 要求的替换组件。可以看出,初次调度的组件路径 <S,1,2,3,5,O>,替换成了 <S,1,2,4,5,O>;由于动态容错满足贪婪函数以及松弛函数,故该分配策略更好。

3.2 基于多目标优化的评价函数

云计算环境下的资源调度需要同时考虑多种复杂多变的应用需求,并兼顾各种性能需求,包括数据中心的资源利用率、经济效益、用户服务质量等。这些问题通常相互关联,相互促进或抑制,并且由于用户对云资源需求级别不同,需要针对不同应用环境动态调整,因此云计算环境下调度问题具有多目标优化的特征。

云系统通过 Map\Reduce 机制,对用户任务进行划分,然后任务管理器对任务进行调配,满足一定需求,例如截止日期限制、负载均衡等。因此,本文通过设计多目标优化的评价函数,将调度过程中多个目标优化为一个综合目标,即评价值 E;设置权重系数,动态适用不同用户对资源的需求等级,以做到全局负载均衡。

在以上讨论中,我们得到资源调度过程中两个目标,即贪婪函数与松弛函数。我们把松弛函数作为资源调度算法的评价函数。对于云任务 mt,我们将 E 作为评价函数。为了方便讨论,将松弛函数的权重系数简化为[0.25,0.25,0.25,0.25],则:

$$E_{mt}=R\begin{bmatrix}\frac{T_{cons}}{T} & \frac{D_{cons}}{D} & \frac{C_{cons}}{C} & \frac{A}{A_{cons}}\end{bmatrix}\begin{bmatrix}0.25\\0.25\\0.25\\0.25\end{bmatrix} \tag{15}$$

为了便于讨论静态容错与动态容错,将静态容错评价函数 E 的动态容错成本 C 看作 C_{cons},同理系统延迟 D 看作 D_{cons},并且系统可用性 A 看作 A_{cons},那么静态容错过程的评价函数将优化为:

$$E_{mt}=R\begin{bmatrix}\frac{T_{cons}}{T} & \frac{D_{cons}}{D_{cons}} & \frac{C_{cons}}{C_{cons}} & \frac{A_{cons}}{A_{cons}}\end{bmatrix}\begin{bmatrix}0.25\\0.25\\0.25\\0.25\end{bmatrix} \tag{16}$$

4 最大评价值优先资源调度算法

基于在线容错的多目标优化评价函数,不仅考虑用户的完成时间,还考虑容错过程,将这两个目标优化为一个评价函数,极大降低资源调度过程的复杂性,做到调度算法的初步优化。由于在线容错的性质,静态容错对关键组件备份,即将调度任务较多的分配到关键组件上,为具体的资源调度提供依据;动态容错则是在执行过程中发生故障,调用替换组件,再次调度;这个过程中,可能发生一次甚至多次调度,单一的调度算法显然不能满足调度任务。

调度方法在满足低计算开销和高可扩展性的同时,还需要具有优异的全局搜索能力和收敛速度,对于一般群智能算法中的收敛速度慢、容易陷入局部最优解、时间/空间复杂度高、求解实际问题的效率不高等缺陷。本文设计最大评价值优先算法——MEF 算法,综合利用 PSO 的全局搜索能力强以及 PBIL 算法收敛速度快的特点,以实现云环境下资源的快速调度。MEF 算法,将评价函数 E 作为适应度函数,将 PSO 与 PBIL 算法进行结合,静态容错时,PSO 对种群进行初始化,初步确定求解范围;发生故障时,调用动态容错,采用 PBIL 算法,从宏观控制种群进化方向,进而更快得到最优分配以及相应的评价值,完成多次资源调度,实现容错。

4.1 基于 PSO 初始化种群

由于 PSO 可设置大量的粒子充斥解空间,增大搜索到更优解的概率,拥有较好的全局搜索能力;适用于本文的资源调度方法。

初始化。先设置最大迭代次数、目标函数的自变量个数、粒子的最大速度,位置信息为整个搜索空间,并在速度区间和搜索空间上随机初始化速度和位置。将 Max-Min 的启发式规则带入,得到了一个种群,重复多次速度和位置更新操作,并将获得的个体加入到种群中,直到种群规模达到要求。

适应度函数。适应度是用来描述种群中不同个体

能够达到或者接近最优解的程度,适应度值越大,越接近最优解。这里简单地采用系统的总评价值来表示个体的适应度。

4.2　基于 PBIL 算法求解动态容错

执行任务发生故障时,继续使用 PSO 全局求解,消耗时间长,计算量大;动态容错强调寻找替换组件,是一种局部调度,因此,使用 PSO 不适用于此环节。PBIL 算法是一种基于现有解学习的进化算法,算法基于当前优秀个体的学习得到概率模型,通过概率模型控制产生新的群体。

通过 PSO 初始化种群之后,采用 PBIL 算法继续求解,这里用评价值作为适应度函数,比遗传算法的交叉变异有更高的效率,最终得到了在系统评价值最大的情况下的资源分配情况,即让每个云任务都得到了最优的云组件去处理。

采用蒙特卡洛方法以频率代表概率的方法进行计算。设 $S_g(z_i)$ 为第 g 代第 i 位变量 z_i 在优势种群中的取值集合,有以下两种情况:

(1) 若解在优势种群中,每位变量 x_i 取不同值出现的频次记为 $F(z_i=k),k=1,2,\cdots,m$。

(2) 若解不在优势种群中,为使种群保持较高的多样性,依据该子任务的平均评价值相对于整体平均评价值的远近进行计算。

$$P_g(z_i=k)=\begin{cases}F(z_i=k)/M & k\in S_g z_i\\ \lambda\times F(z_i=k)/M & \text{其他}\end{cases}\tag{17}$$

$\lambda=|Aa-ave_i|/Aa$ 表示惩罚系数,ave_i 为任务 i 在所有资源上的一个平均评价值。通过学习率函数,以期望寻找全局最优解,为防止算法陷入局部最优,需要提高学习率增长速率。

4.3　MEF 算法

输入:云任务集合 $\{mt_1,mt_2,\cdots,mt_n\}$,云组件集合 $\{Cc_1,Cc_2,\cdots,Cc_m\}$。

输出:云组件最优的分配方案 $O_g=\{z_1,z_2,\cdots,z_n\}$。

1　初始化种群 $O_0=\varnothing$,选择种群 $O_0^S=\varnothing$,计数器 $g=0$。

2　根据静态容错,筛选关键组件,执行 PSO 将任务集合分配到云组件集合。

3　PSO 产生初试种群 O_0 结束。

4　While $g<G$ do

5　　计算个体适应度。

6　　If 系统发生故障 then 采用动态容错算法。

7　　$Z_g^S\leftarrow$根据适应度选择前 M 个体作为种群解。

8　　对每一位计算概率。

9　　根据抽样策略产生新的种群。

10　　$g=g+1$

11　return 分配策略 O_g 以及系统最大评价值。

5　实验与分析

5.1　数据集与实验环境

使用 CloudSim Plus 和 Pajek 生成云计算服务网络。数据级采用公开的 WS-DREAM 数据集,响应时间和吞吐量反映 QoS 值。数据集总结如表 2 所示。

表 2　WS-DREAM 数据集

属性	值	属性	值
用户总数	150	Web 服务总数	100
最短响应时间/s	0.009	最长响应时间/s	28.96
平均响应时间/s	1.73	响应时间标准差/s	3.65
最低吞吐量/kBps	0.03	最高吞吐量/kBps	643.36
平均吞吐量/kBps	4.10	吞吐量标准差/kBps	12.17
Web 服务的总调用数	1 542 884		

CloudSim Plus 是一个功能齐全且记录完整的仿真框架。它易于使用和扩展,可对云计算基础架构和应用程序服务进行建模、仿真和实验。MEF 算法是对 PSO 和 PBIL 算法的优化,PSO 初始化种群,实现静态容错;发生故障时,使用 PBIL 算法再次调度,实现动态容错,使用 CloudSim Plus 框架易于实现。

将 WS-DREAM 数据集中的组件可靠性值($r\in[33\%,89\%]$),假设 $0.8\times r$ 是组件云服务的 SLA 约束。云服务可靠性计算如下:

$$r^+=(100-r)\times rand()+r\tag{18}$$

$$r^-=r\times rand()\tag{19}$$

式中:r^+ 和 r^- 表示执行过程中的可靠性扰动的上限和下限。重构阈值(Ref)表示执行过程可靠性的平均值。即 $Ref=ave(r^-,r^+)$。容错率(Top-k)表示对 $k\%$ 云计算组件进行冗余容错。

5.2　静态容错实验

这里使用 Pajek 生成无标度有向分量图,并且使用生成的边缘权重表示分量调用概率。设置 150 个节点生成了 1 000 个调用序列,并在这些调用序列上应用了现有的容错机制,对比算法如表 3 所示。

表 3　静态容错算法对比

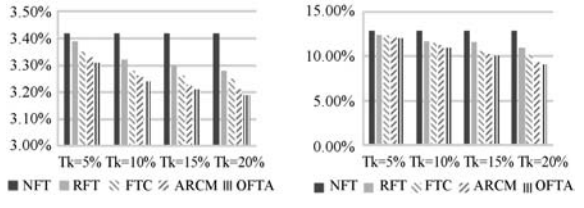
算法	定义
NFT	不采用容错策略,旨在计算云组件故障率
RFT	随机选择 $k\%$ 组件进行容错
FTCloud	通过排序算法对 $k\%$ 组件进行容错
ARCMeas	通过体系结构可靠性排序,然后容错
OFTA	采用动态容错算法进行容错

为了比较这些容错算法的效果,设置两个变量,即容错率(Top-k)以及重构阈值(Ref)两个指标。从 100 次独立的实验中获得的结果的平均值如表 4 所示。

表 4 静态容错实验结果

Ref	Top-k = 5%				
	NFT	RFT	FTC	ARCM	OFTA
5%	3.42%	3.39%	3.35%	3.33%	3.31%
10%	6.62%	6.37%	6.33%	6.11%	6.01%
20%	12.9%	12.47%	12.31%	12.18%	12.04%
Ref	Top-k = 10%				
	NFT	RFT	FTC	ARCM	OFTA
5%	3.42%	3.32%	3.28%	3.26%	3.24%
10%	6.62%	6.01%	5.78%	5.51%	5.43%
20%	12.9%	11.71%	11.52%	11.24%	11.02%
Ref	Top-k = 15%				
	NFT	RFT	FTC	ARCM	OFTA
5%	3.42%	3.3%	3.26%	3.23%	3.21%
10%	6.62%	5.68%	5.36%	5.14%	5.03%
20%	12.9%	11.59%	10.61%	10.22%	10.04%
Ref	Top-k = 20%				
	NFT	RFT	FTC	ARCM	OFTA
5%	3.42%	3.28%	3.25%	3.21%	3.19%
10%	6.62%	5.41%	5.19%	4.87%	4.76%
20%	12.9%	10.94%	10.18%	9.33%	9.08%

随着 Ref 从 5% 增加到 20%,云计算组件故障率会单调增加。较高的 Ref 值会导致较高故障率的制造服务,导致计算组件可靠性下降。Ref 较低,云计算组件的故障率相对较低,故五类算法的可靠性类似;Ref 较高,意味着云计算组件故障率更高,在这种情况下,OFTA 算法可靠性更高,体现出更大的优势,如图 3 所示。



(a) Ref = 5% (b) Ref = 20%

图 3 不同 Top-k 下的系统故障率

5.3 动态容错实验

静态容错通过识别关键组件并进行冗余备份,但不能对所有组件进行备份,故不能完全排除故障。动态容错实验,发生故障时,筛选合适节点替换故障节点,保障任务顺利完成,从而达到容错的目的。

在静态容错实验的基础上,设定参数(Top-k = 20%、Ref = 10%),QoS 度量是判断替代方案是否良好的重要指标。实验选择了四个 QoS 指标:延迟 D、时间 T、成本 C 和可用性 A,来证明在线容错算法的有效性。

DNR 算法:动态处理节点故障,通过利用可模制和延展性作业的灵活性来查找替换节点。

OFTA 算法:采用动态容错算法寻找合适的组件。

通过对 100 个、500 个节点对比,对发生故障后节点替换结果进行统计,结果如表 5 所示。

表 5 动态容错实验

Node	算法	Delay	Time	Cost	Avaliability
100	DNR	0.72	0.68	0.62	0.28
	OFTA	0.69	0.66	0.60	0.29
500	DNR	0.52	0.50	0.47	0.42
	OFTA	0.49	0.48	0.45	0.44

可以看出,在线容错在系统延迟、执行时间、成本以及系统可靠性方面优势明显。随着云计算节点规模的扩大,可替换节点变多,系统延迟时间明显变短;由于节点较多,性能优良的节点多,任务执行时间也变短;容错成本也有降低;系统可用性变高,资源利用率变大。因为如果节点中有更多服务,则通过更多候选服务选择更好的解决方案的可能性更大。由此可以看出,在线容错不仅找到了可替换的健康节点,其执行时间、系统吞吐量也尽可能地降低,达到最优替换,实现负载均衡的目的,其柱状图如图 4 所示。

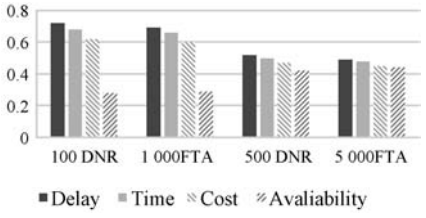


图 4 不同节点规模的 Qos 值

由于本方法是在云组件充足的条件下进行的,即在大型云计算资源池中有较多的组件可供替换(40% 组件处于空闲状态)。业务需要量大时,可供替换组件较少,动态容错会花费更多的时间寻找替换,从而导致算法性能下降。如何保证在较少可用资源的环境下,仍能快速找到合适组件进行替换,是本文下一步的目标。

5.4 执行时间实验

在线容错分为静态容错与动态容错,执行过程中,组件发生故障,动态容错,筛选健康组件进行替换,是一种资源再调度的过程。两次或者多次的资源调度导致计算巨大,计算时间延长,导致用户等待时间延长,浪费计算资源。MEF 算法是一种全新的资源调度算

法,其有效地结合了 PSO 和 PBIL 算法的优势,从宏观角度控制进化,使算法收敛速度大幅降低。

本实验在动态容错实验的基础上,再对算法的执行时间进行分析。通过与 PSO、PBIL 算法对比,分别在 100 个、500 个节点上进行仿真实验。

由图 5 可以看出,MEF 算法收敛速度明显快于 PSO、PBIL 算法,MEF 算法初始化种群时,采用了启发式的 PSO,不仅可以快速确定优势种群的范围,还可以扩大全局搜索能力;使用 NHPP 可靠性模型,可快速确定组件状态,利用 LeaderRank 算法筛选关键组件,可加快算法收敛速度;发生故障时,进行再次调度,采用 PBIL 算法,从宏观控制种群进化方向,从而提高了算法的收敛速度。随着种群规模从 100 变为 500,MEF 算法的求解速度,显著快于其他两种算法。

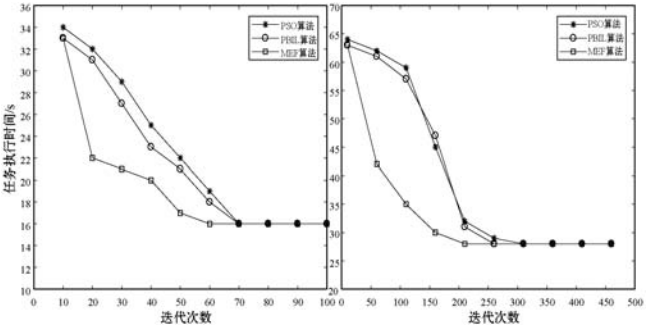


图 5　不同子任务在不同算法下的任务执行总时间

为了排除实验过程人为因素的干扰,设计了 De Jong Function N. 5 来测试算法的全局搜寻能力以及收敛速度。De Jong Function N. 5 是具有 25 个稀疏尖峰的多模态函数,f5(-32, -32) =0.998,其表达式如下:

$$f(x) = \left(0.002 + \sum_{i=1}^{25} \frac{1}{i + (x_1 - a_{1i})^6 + (x_2 - a_{2i})^6}\right)^{-1} \tag{20}$$

如图 6 所示,MEF 用 PSO 对种群初始化,并引入 Max-Min 启发规则,扩大种群的搜寻范围,增强全局搜索能力;使用 PBIL 算法,从宏观控制种群的进化方向,极大地提高了收敛速度。实验结果表明,MEF 算法求解速度更快,更适合在线容错的云计算资源调度。

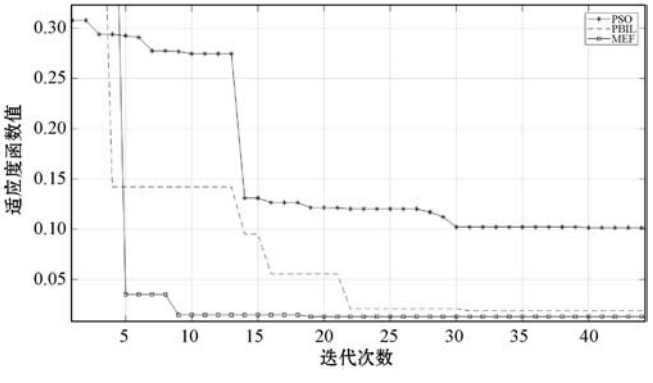


图 6　Dejong 函数收敛测试

所给调度方法主要适用于云计算资源充分环境的容错资源调度。所给方法是针对现有容错和资源调度方法进行的改进,但是暂时在 Java 语言的 CloudSim Plus 框架下进行了模拟实验。然后基于 WS-DREAM 的公开数据集对所给方法的有效性进行实验验证。

6　结　语

本文提出一种新的基于在线容错的云计算资源调度模型,通过在线容错算法提高系统可靠性,静态容错算法筛选位置关键以及可靠性高两类组件,并对关键组件进行备份;发生故障,动态容错算法在故障组件之间的关键组件寻找可靠性高的组件替换故障组件,使任务健康运行;通过 MEF 算法实现资源调度,PSO 根据静态容错对种群进行初始化,PBIL 进一步优化,发生故障时,调用动态容错算法,再次调度。实验部分验证了在线容错算法的可靠性以及 MEF 算法的收敛速度,平衡了用户完成时间以及容错成本,做到了两个方面的优化。下一步将在在线容错模型的基础上,结合绿色计算,进一步优化资源调度模型,在可用资源较少的情况下,做到可靠性高、性能优异、绿色节能全方位优化调度。

参　考　文　献

[1] Qiu X W, Dai Y S, Xiang Y P, et al. Correlation modeling and resource optimization for cloud service with fault recovery [C]//IEEE International Conference on Cloud Computing Technology and Science, 2019, 7(3): 693 - 704.

[2] 段文雪,胡铭,周琼,等. 云计算系统可靠性研究综述[J]. 计算机研究与发展,2020,57(1):102 - 123.

[3] Talwani S, Chana I. Fault tolerance techniques for scientific applications in cloud[C]//2nd International Conference on Telecommunication & Networks, 2017: 1 - 5.

[4] Hasan M, Goraya M S. Fault tolerance in cloud computing environment: A systematic survey[J]. Computers in Industry, 2018, 99: 156 - 172.

[5] 周平. 云服务可用性和可靠性测评与优化方法[D]. 北京:北京邮电大学,2019.

[6] Zheng Z B, Zhou T C, Lyu M R, et al. Component ranking for fault-tolerant cloud applications[J]. IEEE Transactions on Services Computing, 2012, 5(4): 540 - 550.

[7] Kumar M, Sharma S C, Goel A, et al. A comprehensive survey for scheduling techniques in cloud computing[J]. Journal of Network and Computer Applications, 2019, 143: 1 - 33.

[8] Guo P Z, Liu M, Xue Z, et al. A PSO-based energy-efficient fault-tolerant static scheduling algorithm for real-time tasks in clouds[C]//IEEE International Conference Computer and Communications,2018,2537 - 2541.

[9] Wang L. Architecture-based reliability-sensitive criticality measure for fault-tolerance cloud applications[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(11):2408 - 2421.

[10] Sivagami V M, Easwarakumar K S. An improved dynamic fault tolerant management algorithm during VM migration in cloud data center[J]. Future Generation Computer Systems, 2019,98:35 - 43.

[11] Noshy M, Ibrahim A, Ali H, et al. Optimization of live virtual machine migration in cloud computing: A survey and future directions[J]. Journal of Network and Computer Applications,2018,110:1 - 10.

[12] 陈煌宁,郭文忠,陈星. 一种新颖的云计算容错任务调度算法[J]. 小型微型计算机系统,2016,37(10):2194 - 2198.

[13] 杨娜,刘靖. 面向云应用系统的容错即服务优化提供方法[J]. 软件学报,2019,30(4):1191 - 1202.

[14] Prabhakaran S, Neumann M, Wolf F. Efficient fault tolerance through dynamic node replacement[C]//ACM International Symposium Cluster Cloud and Grid Computing,2018: 163 - 172.

[15] Langhnoja H K, Joshiyara P H. Multi-objective based integrated task scheduling in cloud computing[C]//3rd International Conference on Electronics, Communication and Aerospace Technology,2019:1306 - 1311.

[16] 罗云,唐丽晴. 云计算调度粒子群改进算法[J]. 计算机系统应用,2019,28(7):151 - 156.

[17] 孙海胜. 分布估计算法在云计算资源调度中的应用研究[D]. 合肥:中国科学技术大学,2018.

[18] 高尚. 分布估计算法及其应用[M]. 北京:国防工业出版社,2016.

[19] Zheng Z, Trivedi K S, Qiu K, et al. Semi-Markov models of composite web services for their performance, reliability and bottlenecks[J]. IEEE Transactions on Services Computing, 2016,6(1):1939 - 1374.

[20] 吴佳,曾惟如,陈瀚霖,等. 基于隐马尔可夫模型的软件状态评估预测方法[J]. 软件学报,2016,27(12):3208 - 3222.

[21] Moreno-Vozmediano R, Montero R S, Llorente I M. Key challenges in cloud computing: Enabling the future internet of services[J]. IEEE Internet Computing, 2013, 17(4): 18 - 25.

[22] Xu X Y, Tang M L, Tian Y C. QoS-guaranteed resource provisioning for cloud-based MapReduce in dynamical environments[J]. Future Generation Computer Systems, 2018, 78:18 - 30.

[23] Chen N S, Fang X P, Wang X. A cloud computing resource scheduling scheme based on estimation of distribution algorithm[C]//2nd International Conference on Systems & Informatics,2015:304 - 308.

[24] Antonelli F, Cortellessa V, Gribaudo M, et al. Analytical modeling of performance indices under epistemic uncertainty applied to cloud computing systems[J]. Future Generation Computer Systems,2020,102:746 - 761.

[25] Samal A K, Dash A K, Jena P C, et al. Bio-inspired approach to fault-tolerant scheduling of real-time tasks on multiprocessor-a study[C]//IEEE Power, Communication and Information Technology Conference,2016:905 - 911.

(上接第 296 页)

[27] Blondel V D, Guillaume J L, Lambiotte R, et al. Fast unfolding of communities in large networks[J]. Journal of Statistical Mechanics: Theory and Experiment, 2008, 2008: 10008.

[28] Clauset A, Newman M E, Moore C. Finding community structure in very large networks[J]. Physical Review E, 2004,70(6):066111.

[29] Lancichinetti A, Fortunato S, Radicchi F. Benchmark graphs for testing community detection algorithms[J]. Physical Review E,2008,78(4):046110.

[30] Lancichinetti A, Fortunato S. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities[J]. Physical Review E Statistical Nonlinear and Soft Matter Physics,2009,80(1):016118.

[31] http://www-personal. umich. edu.

[32] Zachary W. An information flow model for conflict and fission in small groups[J]. Journal of Anthropological Research,1977,33(4):152 - 173.

[33] Lusseau D, Schneider K, Boisseau O, et al. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations[J]. Behavioral Ecology and Sociobiology,2003,54(4):396 - 405.

[34] Newman M E, Girvan M. Finding and evaluating community structure in networks[J]. Physical Review E,2004,69(2): 26113.

[35] Girvan M, Newman M E. Community structure in social and biological networks[J]. Proceedings of the National Academy of Sciences,2002,99(12):7821 - 7826.

[36] Danon L, Diaz-Guilera A, Duch J, et al. Comparing community structure identification[J]. Journal of Statistical Mechanics: Theory and Experiment,2005,2005:09008.