面向拟态云服务的异构执行体调度算法

普黎明，刘树新，丁瑞浩，王凯
（信息工程大学，河南 郑州 450002）

摘 要：针对当前异构调度模型仅从空间或者时间特性进行建模设计，缺少对时空特性的综合考虑且存在动态性和异构性不够均衡的问题。为使拟态云服务系统的动态、异构和冗余特性能够平衡互补，提出一种基于优先级和时间片的执行池调度算法，该算法基于执行池相似性指标进行优先级预排序，结合时间片等策略进行方案调度。实验结果表明，所提算法有较好的动态性，结合时间片策略能够获得动态性和异构性的综合平衡，且算法耗时较低。

关键词：拟态云服务；动态异构冗余；相似度
中图分类号：TP309
文献标识码：A
doi: 10.11959/j.issn.1000−436x.2020052

Heterogeneous executor scheduling algorithm for mimic cloud service

PU Liming, LIU Shuxin, DING Ruihao, WANG Kai
Information Engineering University, Zhengzhou 450002, China

Abstract: At present, most heterogeneous scheduling models were only designed from the perspective of space or time, without comprehensive consideration the characteristics of time and space, and there was a problem of insufficient balance between dynamics and heterogeneity. In order to balance the dynamic, heterogeneous and redundant characteristics of the mimic cloud service system, an executor pool scheduling algorithm based on priority and time slice was proposed, which performed priority pre-sorting based on the executor pool similarity indicator, and scheduled the program in combination with strategies such as time slice. The experimental results show that the algorithm has good dynamics, and its time slice strategy can achieve a balance of dynamics and heterogeneity, and the algorithm takes less time.

Key words: mimic cloud service, dynamic heterogeneous redundancy, similarity

1 引言

在云计算产业蓬勃发展的大环境下，云安全问题已成为阻碍其发展的难题，未知漏洞或后门成为云安全中最主要的威胁。云计算本质上是在现有技术的基础上建立的，已有技术的漏洞后门会直接转移到云服务平台上[1]，云服务提供商向用户提供大量一致化的基础软件（如操作系统、数据库和应用软件等）资源，又加剧了漏洞后门向云端的聚集，带来了大范围的安全问题与服务隐患。

网络空间拟态防御（CMD, cyber mimic defense）是国内团队提出的改变网络空间“易攻难守”游戏规则的技术，把拟态防御技术应用到云环境中构建拟态云服务，将大大增强系统的安全防御效能，大幅提升攻击者的攻击难度，增强云服务系统的安全性。拟态防御技术以动态异构冗余（DHR, dynamic heterogeneous redundancy）为核心架构，该架构由输入代理、异构执行体池、输出裁决器、控制器等组成。控制器根据调度、裁决等策略指挥系统各部件协同工作，在其控制下，输入代理向异构

收稿日期：2019−12−31；**修回日期**：2020−02−29
基金项目：国家科技重大专项基金资助项目（No.2018ZX03002002）；国家自然科学基金资助项目（No.61521003）
Foundation Items: The National Science and Technology Major Project of China (No.2018ZX03002002), The National Natural Science Foundation of China (No.61521003)

执行体池进行请求分发，异构执行体池处理请求并向输出裁决器发出响应，输出裁决器对各执行体的处理结果进行判决产生唯一输出[2-4]。其中，控制器对异构执行体池的调度对拟态系统的安全性有着重要的作用。目前，该技术已在存储服务器[5]、Web服务器[6]、路由器[7]、SDN控制器[8]、域名服务器[9]等领域得到应用。

传统的异构冗余技术大幅提高了系统的可靠性和容错能力[10-11]，应用比较广泛，但缺乏动态性，不关注异构体调度，抗攻击能力较DHR技术弱[12-13]。拟态云服务系统需要综合考虑执行体的动态、异构和冗余特性，通过调度机制达到三者之间平衡互补[6]。文献[14]提出的最长相异性距离组件选择算法和最佳平均相异距离的软件组件选择算法主要从空间维度讨论相异性，解决容错设计中软件组件的选择问题，但没有考虑动态情况。文献[15]提出的最大异构系统选择算法只从时间维度给出共同漏洞指标，并没有考虑空间维度的漏洞累积，而且也只选出执行体最大的异构集，动态性较差。文献[16]利用复杂性和差异性从空间维度来描述异构性指标，其调度算法稳定性较好，但动态性不足。文献[17]提出共同漏洞指标（CVI, common vulnerability indicator）量化系统相似度，其指标仅从时间维度考虑相似性，缺乏空间维度特性。文献[18]提出异构功能等价体调度模型，从空间层面来度量相似性，并据此提出随机种子最小相似度（RSMS, random seed and minimum similarity）算法，该算法兼顾动态性和可靠性，但其动态性和种子可选集容量强相关，当种子可选集较小时动态性不足。

本文首先基于相似性指标定义了执行体调度指标，从时空维度对执行池相似性进行建模；然后根据该指标提出执行池调度方案以及基于优先级和时间片的执行池调度（PSPT, pool scheduling based on priority and time slice）算法；最后通过实验和分析对PSPT算法和RSMS算法的动态性、相似性和耗时进行比较，讨论算法的综合性能。

2 执行体调度指标

异构执行体调度指标需要综合考虑异构性和动态性，执行池变换得越快，异构性越好，系统越安全。本文通过对执行体的相似性评估来反映其异构性指标，相似性和异构性成反比，且相似性的评估方法较成熟，同时通过策略控制调度得到动态性。

2.1 拟态云服务

拟态云服务（MCS, mimic cloud service）是网络空间拟态防御技术的一种应用，其构造如图1所示。根据拟态防御原理及DHR架构，MCS构造主要包括调度集、执行池、服务代理、控制器。其中调度集是多个功能等价的异构执行体集合实体，图1中表示为虚拟节点；执行池是MCS的功能执行单元，池中虚拟节点实际上映射到调度集中对应的节点，它是调度集的一个逻辑映射单元，不同的节点组合可以映射出不同的执行池；服务代理负责接收并向执行池分发用户请求，对返回的多个响应进行多模裁决并输出；控制器通过配置分发与裁决策略指挥服务代理工作，通过调度策略管理执行池和调度集映射关系。

执行池虚拟节点的数量称作执行体冗余度，简称余度。余度可根据实际情况调整，控制器在给定余度的条件下保持执行池的动态性和异构性。动态性可增加系统的不确定性，减少同种攻击连续成功的概率，提升漏洞和后门的利用难度[6]。异构性可增加系统的稳健性，并减少多个执行体同时存在共同漏洞和后门的概率，进而降低拟态逃逸的概率[2]。因此，执行池调度策略及算法的优劣直接决定了

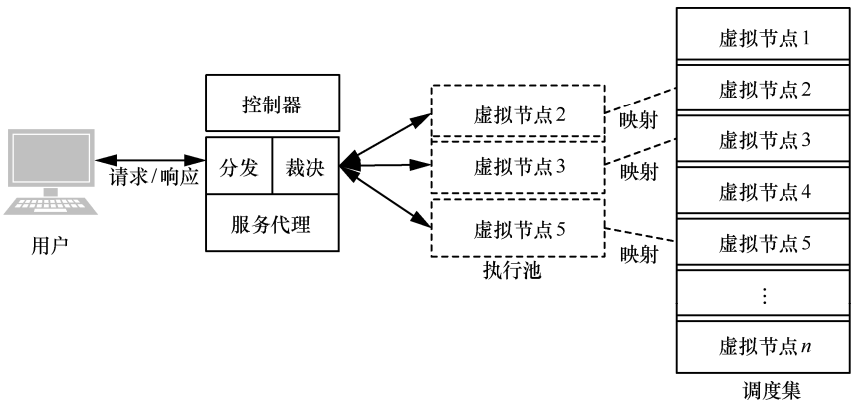

图1 拟态云服务构造示意

MCS 的安全性。

图 1 是在余度为 3 的条件下，由虚拟节点 2、3、5 组成执行池时的拟态云服务构造示意图。当余度变化时，执行池的虚拟节点数量也变化，但结构维持不变；执行池虚拟节点和调度集虚拟节点的逻辑映射关系由控制器通过调度策略来控制。

2.2　相似性指标

本文采用文献[17]的共有漏洞指标表示执行体的相似性，即通过统计执行体共有漏洞的数量来表示相似程度，共有漏洞数多代表相似程度高，反之相似程度低，共同漏洞信息可以通过 CVE（common vulnerabilities and exposures）数据库得到。

定义 1　执行体。提供云服务实际功能的实体记为 Executor_i，简写为 E_i，其中 i 表示第 i 个执行体，在 MCS 系统中称为虚拟节点。

执行体由多个类别的构件组成，例如操作系统、数据库、应用服务软件这 3 种构件组成一个应用功能实体，表示为 $E_i=\{C_{lp}\mid l,p=1,2,3,\cdots,n\}$，其中 C_{lp} 表示第 l 类的第 p 个构件。

同一个系统中，所有执行体的构件种类和数量是一致的。例如，所有执行体的第一种构件是操作系统，第二种是数据库，以此类推。若一个执行体由 3 种构件组成，则其他执行体也同样由这 3 种构件组成，但每个种类的构件数量可能是不一样的，例如某系统中操作系统构件只有 Windows 2012，数据库构件有 MySQL、Oracle、SQL Server 等。

定义 2　调度集。系统中所有功能等价的执行体集合记为 $E=\{E_i\mid i=1,2,3,\cdots,n\}$，调度集中所有的执行体统一编号，即 i 表示编号，n 表示执行体数量，该集合中的执行体是控制器备选的调度对象。

定义 3　执行池。执行池是 MCS 系统 DHR 架构的任务执行部件，系统中同一时刻在线工作的功能等价执行体集合记为 $P=\{E_i\mid 1\leqslant i\leqslant n\}$，且 $|P|=r$，其中 r 表示执行体的余度，考虑到输出时多数一致裁决原则对执行体数量的要求，取 $3\leqslant r\leqslant n$，即执行池至少要有 3 个执行体。由定义可知，$P\subset E$ 表示执行池的执行体由控制器从调度集中调入。

考虑到组成执行体的构件随时间在不断发展变化（例如，Windows 已经从最初的 3.1 版本发展到如今的 Windows 10），其漏洞的种类和数量也发生了变化，可能会变得越来越多，或者越来越少。因此，在选择执行池配置时应考虑时间维度，例如，相比最近出现较少漏洞的执行体可能会有更好的安全性，最近共有漏洞较少的执行体集也会有更好的异构性；最近出现的漏洞威胁较高，时间权重配置高一些，较早出现的漏洞威胁较低，时间权重配置低一些。

定义 4　时间权重因子[17]。定义为

$$\alpha_i=1-\frac{y-i}{\text{years}} \tag{1}$$

其中，$i\in\{y-\text{years}+1,\cdots,y-2,y-1,y\}$；$y$ 表示年份（例如 2019 年，则 $y=2019$）；years 表示参与漏洞统计的年数，至少为 1 年，通常不会有使用超过 10 年的构件版本。由式(1)可知，在若干年时间内，越接近 y 权重越大，y 的时间权重因子取值为 $\alpha_y=1$。α 因子从时间维度考虑了构件历史漏洞的重要程度，也可理解为攻击者利用历史漏洞的可能性。

定义 5　空间权重因子。执行体由多个类别的构件组成，从系统结构角度看是多个空间层次的构件组成，例如通常所说的存储层、计算层、服务层，或者硬件层、系统层、应用层等。每个层次（类别）的构件在执行体整体安全性中的重要性是不一样的，对于应用型服务来说，攻击者能直接攻击的目标必然是最顶层的应用服务软件，比如 Web 服务，该层构件的安全性直接决定了执行体的整体安全性，因此该构件的空间权重应设置较大。定义为

$$\beta_l=\frac{l}{\text{tiers}} \tag{2}$$

其中，$\text{tiers}>0$ 表示组成执行体的构件层数，例如由存储层、计算层、服务层组成的执行体 $\text{tiers}=3$；$l\in\{1,2,3,\cdots,\text{tiers}\}$ 表示第 l 层的构件，取值由底层向顶层递进，底层的构件 $l=1$，顶层的构件 $l=\text{tiers}$。

由定义 5 可知，底层构件的空间权重因子 β 取值最小，顶层构件的空间权重因子取值为 $\beta_{l=\text{tiers}}=1$。β 因子从空间维度考虑了各层（类）构件的重要程度，也可理解为攻击者利用该构件攻击的可能性。

定义 6　构件共有漏洞指标。同一种类的构件对 C_{lp} 和 C_{lq} 的共同漏洞数记为 $v_i(A,B)$，其定义为

$$\text{CVI}_{ly}(C_{lp},C_{lq})=\sum_{i=y-\text{years}+1}^{y}\alpha_i v_i(C_{lp},C_{lq}) \tag{3}$$

式(3)表示第 l 类构件对第 p 和 q 号构件在 y 年的共有漏洞指标，该指标从时间维度考虑了最近 years 年的构件共有漏洞情况，在一定程度上也可由历史表

现反映或预测未知漏洞的部分情况。共有漏洞数量越大，CVI 越大，构件越相似，反之异构性越小。

定义 7 执行体共有漏洞指标。结合定义 1 和定义 5 可以得到执行体对的 CVI，表示为

$$\mathrm{CVI}_y\left(E_j,E_k\right)=\sum_{l=1}^{\mathrm{tiers}}\beta_l\mathrm{CVI}_{ly}(C_{lp},C_{lq}) \tag{4}$$

其中，$C_{lp}\in E_j, C_{lq}\in E_k$，$l$ 表示层次（或类别），tiers 表示组成执行体的构件层数。该指标从空间维度考虑了执行体的共有漏洞情况。

执行体 CVI 物理意义如图 2 所示。

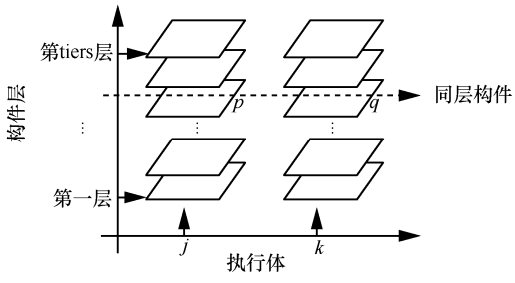

图 2 执行体 CVI 物理意义

从图 2 中可看出，执行体 CVI 从横向角度表示构件对（j 号执行体的某层 p 号构件和 k 号执行体的同层 q 号构件）之间的共有漏洞指标，该指标回溯了最近若干年的共有漏洞情况，体现了时间维度特性；从纵向角度表示执行体对的共有漏洞指标由各层构件对（从第一层到第 tiers 层）的 CVI 累计得出，并考虑了各层的重要性程度，体现了空间维度特性。

定义 8 执行池漏洞指标。定义为

$$\mathrm{CVI}_y(P)=\frac{1}{C_r^2}\sum_{j=1,k=2}^{r-1,r}\mathrm{CVI}_y(E_j,E_k) \tag{5}$$

其中，$j<k$ 且 $1\leqslant j\leqslant r-1, 2\leqslant k\leqslant r$，$r$ 表示该执行池的余度，j、k 表示执行体在当前执行池中的编号。该指标表示执行池中执行体对 CVI 的平均值。

3 执行池调度算法

对于给定的构件集，其组成的调度集显然是确定的，满足 r 余度要求的执行池数量也是确定的，即从调度集中构建 r 余度执行池的数量是确定的。当构件集发生变化（新增或减少构件）时，调度集和执行池也随之变化，但直到构件集下一次变化之前，调度集和执行池是确定的。此外，由于 CMD 负反馈机制等原因引起执行体下线或清洗，也会改变执行体集和执行池。因此，构件集、调度集和执行池在总体上具有不确定性，是动态变化的，但在一定的时间片内是相对确定的。执行体的调度是在一定的条件下启动的，例如构件集变化了，可能有更好的相似性组合，可以调入该组合；执行体下线了，引起执行池的变化，需要调入一个执行体；动态策略触发，需要调整执行池等。

不论何种原因触发调度，系统都可以预先准备好执行池调度方案，这比在触发时再计算相关方案能够得到更好的调度效率。因此有如下定义。

定义 9 调度方案。调度方案是以 $\mathrm{CVI}_y(P)$ 升序排列的执行池集合，记为 $F=\{P_i\mid i=1,2,\cdots,N\}$，其中 P 表示执行池，i 表示排序后的序号，i 值越小表示该序号的执行池相似度越小，即 $\mathrm{CVI}_y(P)$ 越小，该执行池将优先调度，即序号就是调度优先级。

通常情况下，对于给定余度的执行池，应选调 $\mathrm{CVI}_y(P)$ 较小的执行池，但执行池内的执行体之间可能存在局部极值的情况[2-3]，虽然平均相似度较小，但某一对执行体的相似度比较大，它们的共有漏洞被攻击成功的概率相对较高。调度方案应考虑这种情况，可设置一个相似度阈值，如果执行池中存在超过阈值的执行体对，则降低该执行池优先级，放入方案末端，也可以根据策略删除该执行池。

定义 10 相似度阈值。记为 threshold，一般情况下 threshold 取值应大于执行体 CVI 的平均值，且应小于最大值，具体取值可根据经验调整。

对于执行体数量为 n 的调度集（已根据构件的兼容属性配置好执行体），以 r 余度构建执行池，可以构建出包含 C_n^r 个执行池的列表，记作 PoolList。系统在启动时，或在执行体下线及上线后，新的调度任务还未触发前，分别计算出 C_n^r 个执行池的 $\mathrm{CVI}_y(P)$ 值，并按照前文原则排序得出调度方案 F，此时调度方案 F 就是排序后的 PoolList，同时可知调度方案 F 已经包含了 r 余度下所有可能的执行池。调度方案生成算法伪代码如算法 1 所示。

算法 1 调度方案生成算法

输入 调度集 E，余度 r，相似度阈值 threshold，调度策略 Policies

输出 调度方案 F

1) PoolList = BuildPools(E, r) //按照余度要求，从调度集 E 中构建出 C_n^r 个执行池 PoolList

2) OverList = null //初始化超阈值执行池列表

3) for i = 0; $i<C_n^r$; i++

4) CVI_y(PoolList [i]) //计算各执行池的漏

洞指标，结果保存在执行池结构信息中

5) if PoolList [*i*]. over-threshold == true and Policies.isRemoved == true

6) PoolList [*i*].remove()//若存在超过阈值的执行体对且策略要求删除，则从方案列表 PoolList 中删除

7) else if PoolList [*i*]. over-threshold == TRUE

8) Move(PoolList [*i*], OverList) //若存在超过阈值的执行体对，则移到超阈值列表 OverList

9) end if

10) end for

11)if Policies.isRemoved == true

12) return PoolList //返回已过滤超阈值执行体池的方案

13) else

14) AscendingSort (PoolList) //对执行池列表按 CVI 值进行升序排序

15) AscendingSort (OverList) //对超阈值执行池列表按 CVI 值进行升序排序

16) return PoolList + OverList //返回包含超阈值执行体池的方案

17) end if

在系统执行过程中，若因安全原因下线执行体，则需选调另一个不包含该执行体的执行池；另外，考虑到稳定性要求，通常会选调包含当前执行池中非下线执行体的执行池，以减少更换的执行体。若是动态策略触发调度操作，为保障良好的动态性和较长的调度周期，只需选调下一优先级的执行池，按照调度方案 F 循环调度。执行体下线和上线以及余度 r 变化会引起执行池的变化，在本次调度完成之后，需要使用算法 1 重新生成调度方案，但不会立即触发调度操作；动态策略触发调度操作不会使调度池变化，不需要重新生成调度方案。

定义 11　时间片策略。对于策略触发的调度，系统根据方案优先级顺序给出一个调度时间间隔，称为时间片，记为 $T_F=\{t_i \mid i=1,2,\cdots,N\}$，其中 T_F 表示调度方案 F 的时间片策略，t_i 表示方案中第 i 个执行池的时间片。优先级高的调度间隔长，反之调度间隔短，以此在受控条件下平衡动态性和异构性要求，使之达到互补。系统根据时间片策略调用算法 2 对执行池进行动态调度，该算法称为 PSPT 调度算法。

算法 2　PSPT 调度算法

输入　当前池序号 cp，调度方案列表 PoolList，下线执行体 E_{down}，是否策略触发 isPolicy

输出　当前执行池序号

1) if isPolicy == true

2) return (cp+1) mod C_n^r //若是策略触发的调度，则返回下一个优先级的执行池

3) end if

4) TempList = null //初始化临时可选列表

5) for $i=0$; $i<C_n^r$; i++

6) if $E_{\text{down}} \in$ PoolList [*i*] //过滤包含下线执行体的执行池

7) continue

8) end if

9) if PoolList [*i*] $\supset$ (PoolList [cp]$-E_{\text{down}}$) //若包含当前池的全部非下线执行体，则选中该执行池

10) return *i*

11) end if

12) TempList.add(*i*) //可选序号临时保存

13) end for

14) return TempList[0] //返回可选列表中优先级最高的执行池

4　实验分析

本节对调度算法进行实验和分析，把 PSPT 算法与文献[18]提出的 RSMS 算法对比，测试算法的动态性、平均相似度、耗时等效果。实验设备采用 Thinkpad T480 笔记本电脑，配置为 Intel Core i7 1.80 GHz CPU 和 16 GB RAM，软件系统采用 Ubuntu Linux 版本 18.04LTS 和 JDK 12.0.1，工具为 Java 和 Python。

根据文献[6,19]对余度与成本及安全增益的关系分析可知，余度为 3 的执行池实用性较好，且可以达到最佳折中效果。因此，本文实验主要以余度为 3 的执行池开展。

4.1　算法动态性比较

算法的动态性体现在调度方案的重复周期，理想的动态性是调度的方案尽量不重复，但受于可选异构执行体和余度的限制，调度方案实际上是一个有限集合，方案在调度过程中必然会出现重复，重复的平均周期越长，动态性越好。

分析 RSMS 算法可知，该算法的动态性主要由随机函数来体现，当随机函数选中种子执行体时，该种子对应的执行池也就确定了，也即调度方案和

种子是对应的，方案的调度周期等效于种子执行体重复出现的周期。而 PSPT 算法的调度周期由调度方案列表的长度决定，对于给定的调度集，调度方案长度是确定的，该长度值为 C_n^r，其中 n 为调度集的执行体个数，r 为执行池余度。

实验对 100 个执行体的调度集进行算法调度测试，执行池余度设为 3，假设执行体被调度的概率相同。为简化分析聚焦周期测试，暂不考虑相似度阈值，加入相似度阈值因素后，可选范围会缩小，2 种算法的平均周期值都会有所降低，即动态性会变差。每个算法进行 100 次实验，出现重复方案时按一次计算。实验结果如图 3 所示，其中周期表示的是调度次数，粗虚线表示周期的平均值。

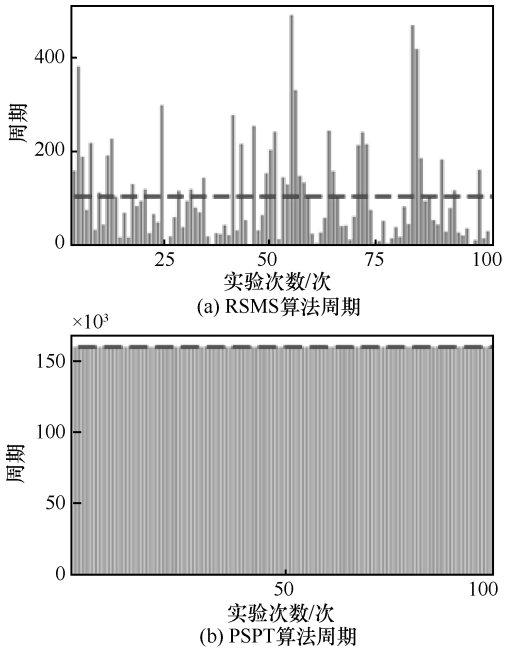

图 3　算法周期测试

分析图 3 可知，RSMS 算法的调度周期符合随机函数特点，取值比较发散，平均周期约为 100；PSPT 算法的调度周期实验值符合算法预期值 C_{100}^3=161 700，该算法的周期固定为 161 700，平均周期约是 RSMS 算法的 1 617 倍。实验结果与前述分析一致，PSPT 算法的动态性大幅优于 RSMS 算法。

4.2　算法相似度比较

RSMS 算法选中种子执行体时，也就确定了执行池，换句话说，对于给定的 100 个执行体的调度集，只有 100 个可能的种子，对应可选中的就是 100 个可能的执行池，而每个执行池都有 CVI 值。因此，为简化实验，聚焦算法的随机特性带来的相似度变化，在给定余度条件下，本文在执行池 CVI 最小值和总平均值（调度集所有执行体组合的平均 CVI 值）之间随机给出 100 个执行池作为 RSMS 算法的可选集，PSPT 算法的可选集为 C_{100}^3=161 700 个；假设余度为 3 的执行池 CVI 值服从参数为[5,15]的 β 分布[18]，则概率密度曲线和对应的执行池 CVI 值分布如图 4 所示。

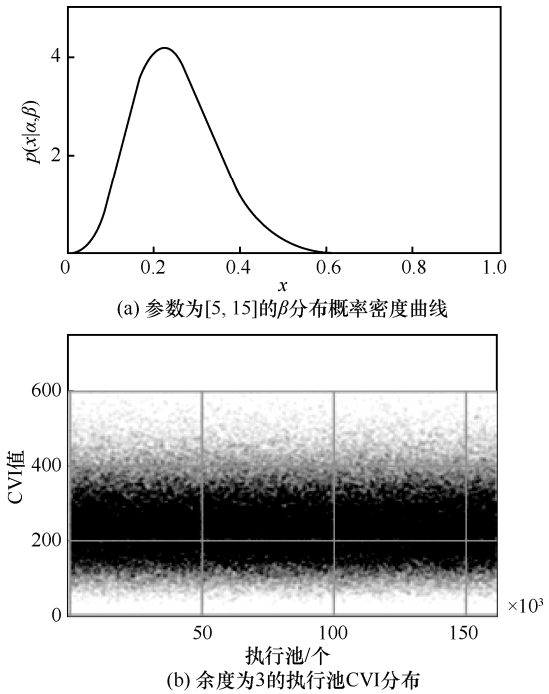

图 4　实验数据 CVI 值分布特征

图 4 中 CVI 数据总平均值为 249.7。考虑到 RSMS 算法每个周期的调度次数有很大不同（见 4.1 节分析），为得到较好的平均数据，进行 100 个周期的调度实验并计算 CVI 平均值；PSPT 算法每周期的调度次数相同，而且一个周期即可覆盖方案中所有的执行池，该算法进行一个周期的调度实验即可计算出平均 CVI。

实验 1　不考虑阈值和时间片策略，余度 $r=(3,4,5)$，测试调度算法得到方案的平均相似度。实验结果如表 1 所示。

表 1　不同算法的调度方案平均相似度

算法	r=3	r=4	r=5
RSMS 算法	174.06	195.82	212.37
PSPT 算法	249.70	249.70	249.70

由表 1 可知，RSMS 算法的平均相似度随余度增加而增加；PSPT 算法的平均相似度不随余度变化，维持在总平均值。当余度较小时，RSMS 算法

的平均相似度大幅优于 PSPT 算法，随着余度的增加差距逐渐减小，当余度增加到等于执行体数量时，平均相似度将趋于一致。

实验 2　不考虑阈值，将余度设为 3，针对 PSPT 算法启用时间片策略并测试单位时间内的平均相似度。时间片策略数据如下：基准时间片取 10 个单位时间，调度方案 F 列表以 $\frac{\text{len}F}{2}$ 为界，后半部分顺序取值区间为[10,100]，前半部分顺序取值区间为[100,10K]，其中 $K=\{10,20,30,\cdots,200\}$，方案 F 从该区间中由大到小获取时间片，实验结果如图 5 所示。

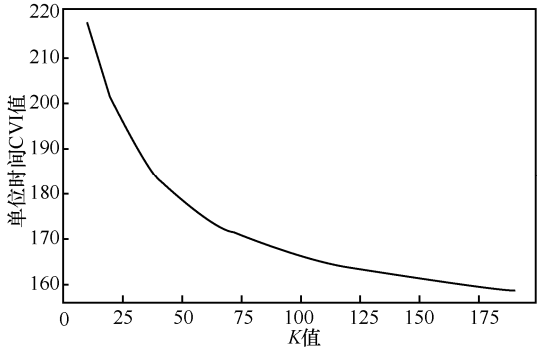

图 5　单位时间 CVI 值随时间片策略变化关系

从图 5 可以看出，随着 K 值增大，单位时间内 CVI 值快速减小，逼近表 1 中 RSMS 算法的平均值。经分析可知，因为调度方案前半部分的执行池 CVI 值较小，增大该部分时间片将会在一段时间内降低 CVI 值，该段时间内执行池的相似性较低，但动态性有所损失。实际应用时可根据情况设置合适的时间片策略，获得异构性和动态性的平衡，进而得到较好的安全性。

实验 3　不考虑时间片策略，将余度固定为 3，调整相似度阈值，测试相似度阈值对算法平均相似度的影响。实验结果如图 6 所示。

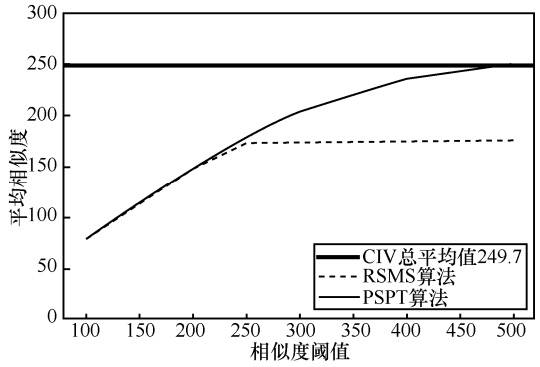

图 6　不同算法平均相似度随相似度阈值变化关系

从图 6 中可以看出，随着相似度阈值的减小，RSMS 算法的平均相似度变化不大，当相似度阈值低于总平均值 249.7 时，平均相似度快速减小；PSPT 算法的平均相似度随相似度阈值快速减小。经分析可知，当相似度阈值减小时，PSPT 算法快速过滤掉相似度较大的执行池，使获得的方案列表缩短，平均相似度降低，调度周期也变短；当相似度阈值低于总平均值时，RSMS 算法也会较多地过滤掉可选方案，使平均相似度快速下降，同时调度周期也缩短。实际应用时，因前者的平均相似度由算法本身决定，不易附加干预手段，需要控制好余度的选取，以达到成本和相似性的平衡，余度过高会增加成本。后者需要控制好阈值选取，以取得较好的平均相似度且平衡动态性要求，且后者受策略控制较大，可根据实际经验干预并调整策略。

4.3　算法耗时比较

调度算法的执行，大多是由策略驱动的，执行体下线或上线引发的调度是少数。因此，实验只考虑策略驱动的调度操作，聚焦算法本身的耗时比较。考虑到算法耗时主要受到调度集规模的影响，通过增加 2 种算法的调度集规模来测试耗时情况，调度集规模设为 3~1 000，对每个规模调度集分别测量算法的 CPU 耗时。实验结果如图 7 所示。

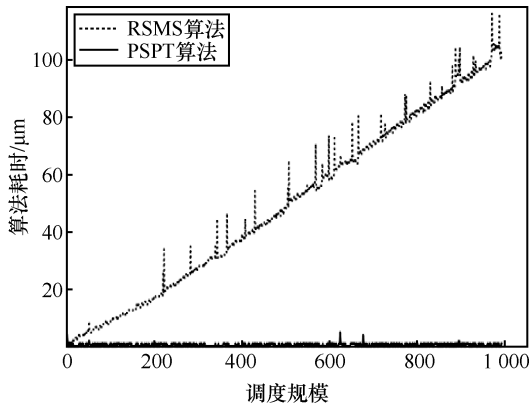

图 7　调度规模耗时情况

由图 7 可以看出，RSMS 算法耗时随规模总体呈线性增加，PSPT 算法耗时基本维持不变。经分析可知，RSMS 算法主要由循环语句产生耗时，该语句循环次数与规模直接关联，时间复杂度为 $O(n)$；PSPT 算法处理的数据已经过排序处理，可直接按序取出，耗时与规模无关，时间复杂度为 $O(1)$。PSPS 算法能够稳定维持较低的时间成本和较小的

调度时延，优于 RSMS 算法。

5 结束语

拟态云服务是网络空间拟态防御技术的一种应用，执行体的调度机制是其关键技术，本文针对调度机制给出包含时空特性的执行体调度指标，以此提出一种基于优先级和时间片的执行池调度算法 PSPT，该算法在动态性和时间成本上优于 RSMS 算法，配合时间片策略也可在相似性上接近 RSMS 算法，且在满足冗余性基础上可以兼顾动态性和异构性。后续考虑对服务代理接收执行体返回值的网络时延情况进行研究，据此优化调度方案和算法。

参考文献：

[1] 张玉清, 王晓菲, 刘雪峰, 等. 云计算环境安全综述[J]. 软件学报, 2016, 27(6): 1328-1348.
ZHANG Y Q, WANG X F, LIU X F, et al. Survey on cloud computing security[J]. Journal of Software, 2016, 27(6): 1328-1348.

[2] 邬江兴. 网络空间拟态防御原理[M]. 北京: 科学出版社, 2018.
WU J X. Cyberspace mimic defense[M]. Beijing: Science Press, 2018.

[3] 邬江兴. 网络空间拟态防御研究[J]. 信息安全学报, 2016, 1(4): 1-10.
WU J X. Research on cyber mimic defense[J]. Journal of Cyber Security, 2016, 1(4): 1-10.

[4] 扈红超, 陈福才, 王禛鹏. 拟态防御 DHR 模型若干问题探讨和性能评估[J]. 信息安全学报, 2016, 1(4):40-51.
HU H C, CHEN F C, WANG Z P. Performance evaluations on DHR for cyberspace mimic defense[J]. Journal of Cyber Security, 2016, 1(4): 40-51.

[5] 陈越, 王龙江, 严新成, 等. 基于再生码的拟态数据存储方案[J]. 通信学报, 2018, 39(4): 21-34.
CHEN Y,WANG L J, YAN X C, et al. Mimic storage scheme based on regenerated code[J]. Journal on Communications, 2018, 39(4): 21-34.

[6] 仝青, 张铮, 张为华, 等. 拟态防御 Web 服务器设计与实现[J]. 软件学报, 2017, 28(4): 883-897.
TONG Q, ZHANG Z, ZHANG W H, et al. Design and implementation of mimic defense Web server[J]. Journal of Software, 2017, 28(4): 883-897.

[7] 马海龙, 伊鹏, 江逸茗, 等. 基于动态异构冗余机制的路由器拟态防御体系结构[J]. 信息安全学报, 2017, 2(1): 29-42.
MA H L, YI P, JIANG Y M, et al. Dynamic heterogeneous redundancy based router architecture with mimic defenses[J]. Journal of Cyber Security, 2017, 2(1): 29-42.

[8] 李传煌, 任云方, 汤中运, 等. SDN 中服务部署的拟态防御方法[J]. 通信学报, 2018, 39(S2): 121-130.
LI C H, REN Y F, TANG Z Y, et al. Mimic defense method for service deployment in SDN[J]. Journal on Communications, 2018, 39(S2): 121-130.

[9] 王禛鹏, 扈红超, 程国振. 一种基于拟态安全防御的 DNS 框架设计[J]. 电子学报, 2017, 45(11): 139-148.
WANG Z P, HU H C, CHENG G Z. A DNS architecture based on mimic security defense[J]. Acta Electronica Sinica, 2017, 45(11): 139-148.

[10] 殷斌, 陆熊, 陶想林. 非相似三余度飞控计算机设计和可靠性分析[J]. 测控技术, 2015, 34(5): 53-56.
YIN B, LU X, TAO X L. Design of a prototype flight control computer system with triple dissimilar redundancy[J]. Measurement & Control Technology, 2015, 34(5): 53-56.

[11] 臧红伟, 韩炜, 高德远. 非相似余度计算机系统及其可靠性分析[J]. 哈尔滨工业大学学报, 2008, 40(3): 492-494.
ZANG H W, HAN W, GAO D Y. Dissimilar redundancy computer system and reliability analysis[J]. Journal of Harbin Institute of Technology, 2008, 40(3): 492-494.

[12] 王伟, 杨本朝, 李光松, 等. 异构冗余系统的安全性分析[J]. 计算机科学, 2018, 45(9): 183-186, 194.
WANG W, YANG B C, LI G S, et al. Security analysis of heterogeneous redundant systems[J]. Computer Science, 2018, 45(9): 183-186, 194.

[13] 王伟, 曾俊杰, 李光松, 等. 动态异构冗余系统的安全性分析[J]. 计算机工程, 2018, 44(10): 42-45, 50.
WANG W, ZENG J J, LI G S, et al. Security analysis of dynamic heterogeneous redundant system[J]. Computer Engineering, 2018, 44(10): 42-45, 50.

[14] 姚文斌, 杨孝宗. 相异性软件组件选择算法设计[J]. 哈尔滨工业大学学报, 2003, 35(3): 261-264.
YAO W B, YANG X Z. Design of selective algorithm for diverse software components[J]. Journal of Harbin Institute of Technology, 2003, 35(3): 261-264.

[15] WANG Y W, WU J X, GUO Y F, et al. Scientific workflow execution system based on mimic defense in the cloud environment[J]. Frontiers of Information Technology & Electronic Engineering, 2018, 19(12): 1522-1537.

[16] 张杰鑫, 庞建民, 张铮, 等. 面向拟态构造 Web 服务器的执行体调度算法[J]. 计算机工程, 2019, 45(8): 14-21.
ZHANG J X, PANG J M, ZHANG Z, et al. The executors scheduling algorithm for the Web server with mimic construction[J]. Computer Engineering, 2019, 45(8): 14-21.

[17] GARCIA M, BESSANI A, GASHI I, et al. Analysis of operating system diversity for intrusion tolerance[J]. Software: Practice and Experience, 2014, 44(6): 735-770.

[18] 刘勤让, 林森杰, 顾泽宇. 面向拟态安全防御的异构功能等价体调度算法[J]. 通信学报, 2018, 39(7): 188-198.
LIU Q R, LIN S J, GU Z Y. Heterogeneous redundancies scheduling algorithm for mimic security defense[J]. Journal on Communications, 2018, 39(7): 188-198.

[19] 魏帅, 于洪, 顾泽宇, 等. 面向工控领域的拟态安全处理机架构[J]. 信息安全学报, 2017, 2(1): 54-73.
WEI S, YU H, GU Z Y, et al. Architecture of mimic security processor for industry control system[J]. Journal of Cyber Security, 2017, 2(1): 54-73.

[作者简介]

普黎明（1976− ），男，云南嵩明人，信息工程大学副研究员，主要研究方向为网络安全、网络体系结构。

刘树新（1987− ），男，山东潍坊人，博士，信息工程大学助理研究员，主要研究方向为复杂网络、网络信息挖掘。

丁瑞浩（1988− ），男，河南郑州人，信息工程大学助理研究员，主要研究方向为网络安全。

王凯（1980− ），男，河南许昌人，博士，信息工程大学副研究员，主要研究方向为电信网安全。