计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(10):180−186 [doi: 10.15888/j.cnki.csa.008093]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

# 拟态通用运行环境外部表决机制研究①

戴人杰, 柴小丽, 邵培南, 徐李定

(中国电子科技集团公司第三十二研究所, 上海 201808)
通讯作者: 戴人杰, E-mail: 635591775@qq.com

**摘　要**: 基于异构冗余架构的择多表决机制实现了拟态防御系统的容错机制. 在拟态通用运行环境 (Mimic Common Operating Environment, MCOE) 中. 由外部表决模块对异构执行体响应数据进行大数表决来实现这一机制. 为完善表决机制, 提高表决速度, 本文提出了基于历史表现安全性和异构置信度的大数表决机制和并行聚类算法. 改进的大数表决机制有效地修补了简单大数表决机制存在的无法产生表决结果以及忽视执行体本身安全性和相关性的问题; 并行聚类算法解决了表决过程中数据闲置的问题, 显著提高了表决速度. 此外为了保证数据比对的可靠性, 本文设计了针对结构化, 非结构化 (图像、音频、视频) 数据的特定处理流程.

**关键词**: 拟态防御; 拟态通用运行环境; 外部表决; 大数表决; 并行聚类; 异构度

引用格式: 戴人杰,柴小丽,邵培南,徐李定.拟态通用运行环境外部表决机制研究.计算机系统应用,2021,30(10):180–186. http://www.c-s-a.org.cn/1003-3254/8093.html

## Research on External Voting Mechanism of MCOE

DAI Ren-Jie, CHAI Xiao-Li, SHAO Pei-Nan, XU Li-Ding

(The 32nd Research Institute of China Electronics Technology Group Corporation, Shanghai 201808, China)

**Abstract**: The majority voting mechanism based on heterogeneous redundant architecture realizes the fault tolerance of mimic defense systems. In the Mimic Common Operating Environment (MCOE), the mechanism is achieved after the external voting module performs the majority voting on the response data of heterogeneous executors. To improve the voting mechanism and improve the voting speed, this study proposes a majority voting mechanism based on historical performance security and heterogeneous confidence and a parallel clustering algorithm. The improved majority voting mechanism can effectively tackle the problems of the simple majority voting mechanism that cannot produce voting results and ignore the security of the executors themselves and their correlation. The parallel clustering algorithm solves the problem of idle data in the voting process and significantly improves the voting speed. In addition, this study designs a specific processing flow for structured and unstructured (images, audios, videos) data to ensure the reliability of data comparison.

**Key words**: mimic defense; Mimic Common Operating Environment (MCOE); external voting; majority voting; parallel clustering; heterogeneous degree

计算机信息系统的运行可能受到来自软硬件病毒, 漏洞的安全威胁[1]. 这些攻击会导致信息系统运行异常或终止, 数据被篡改、盗取或毁坏[2]. 信息系统的安全防御技术分为被动防御和主动防御[3]. 被动防御基于对已知安全威胁特征规则的病毒、后门、漏洞等进行威胁诊断和数据清洗, 如防火墙、病毒检测、后门检测等[4]. 主

① 基金项目: 上海市科学技术委员会科研计划 (18511104402)
Foundation item: Scientific Research Project of Science and Technology Commission of Shanghai Municipality (18511104402)
收稿时间: 2021-01-02; 修改时间: 2021-01-29; 采用时间: 2021-02-02

动防御则着力于提升系统内生的架构安全性、健壮性.

邬江兴院士提出的拟态防御理论, 是一种以动态异构冗余 (Dynamic Heterogeneous Redundancy, DHR) 原理[5] 为基础的主动防御理论.

拟态通用运行环境 (Mimic Common Operating Environment, MCOE) 是实现面向 C/C++, Java 语言开发的 C/S 或 B/S 信息系统的拟态防御架构[6]. MCOE 提供对客户端服务请求的分发、执行、表决和安全威胁诊断等服务功能, 实现了服务请求执行过程的自动化, 达到对已/未知后门和漏洞主动防御, 安全威胁攻击及时阻断, 数据完整性确保等拟态防御目标[7].

拟态通用运行环境由拟态分发器, $N$ 个异构执行体, 内/外部表决器, 管理服务器和协同执行服务器组成. 拟态分发器面向客户端应用服务请求, 提供对其进行接收、处理、分发以及对 $N$ 个异构执行体运行节点服务器的服务请求响应结果的接收、处理和转发客户端等功能, 内/外部表决器分别针对服务请求执行过程中形成的 $N$ 个结果数据 (内部) 和请求响应结果数据 (外部) 的表决调用, 提供对应的表决服务, 实施表决结果的异常诊断和鲁棒性处理过程, 并向管理或分发模块反馈表决结果. 管理服务器提供了拟态应用管理、拟态资源管理和拟态安全管理等功能. 代理执行管理者命令, 周期性或心跳地归集和上报各运行环境节点的软硬件运行过程形成的数据 (如日志和资源状态) [7].

## 1 外部表决机制

### 1.1 拟态通用运行环境机制简介

拟态信息系统执行客户端请求的简略流程 (省略了内部表决、协同执行等与外部表决无关的部分流程) [8] 如图 1 所示.

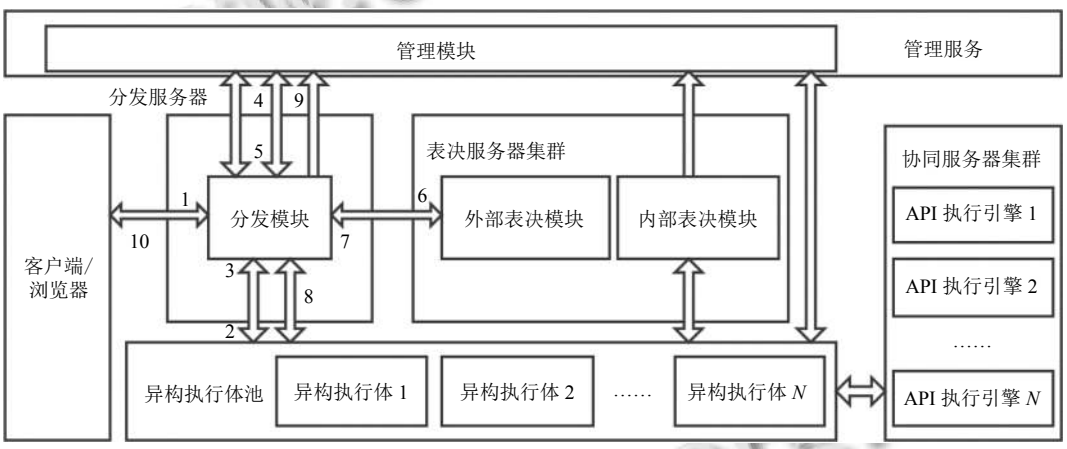


图 1 拟态通用运行环境简略架构

图 1 中的数字代表执行顺序, 详细内容如下:

(1) 客户端向分发器发送服务请求.

(2) 分发器将客户端服务请求分发到被选中的若干异构执行体.

(3) 异构执行体向分发器返回多份对服务请求的响应数据.

(4) 分发器保存异构体响应数据并向管理服务器请求资源 (外部表决) 调用.

(5) 管理服务器向分发器返回外部表决资源.

(6) 分发器将执行体响应数据发往外部表决器.

(7) 外部表决器返回表决结果, 包括表决出的异构执行体标识、执行体出错信息以及是否需要调用鲁棒性处理的信息.

(8) 分发器调用鲁棒性处理模块.

(9) 分发器向管理服务器发送外部表决定位到的执行体错误信息.

(10) 分发器向客户端返回响应结果.

### 1.2 外部表决机制

传统的 Web 服务模式是通过单个执行体产生响应数据, 并直接返回到客户端. 这种模式的安全性完全依赖于防火墙、病毒检测等被动防御策略. 而拟态信息系统采用多个执行体对同一个请求进行响应, 并表决出一个结果作为最终输出. 这样的机制使得系统拥有主动检错的能力, 多个异构体执行相同请求产生的响应通过一致性表决后产生的输出显然比单个执行体的输出更加可信. 在拟态系统中执行响应数据表决任务的就是外部表决器.

当使用 2 个执行体时, 如果有某个执行体遭受攻

击导致数据被篡改, 即响应数据不一致, 外部表决器就无法通过此次表决. 这样可以做到“不将错误的数据返回到客户端”, 但是无法做到“在少数执行体数据被篡改时, 依然可以返回正确数据”, 也无法定位到出错的执行体. 因此为了满足拟态机制的容错性要求, 至少需要 3 个执行体的响应数据才能在某个异构体响应数据被篡改的情况下仍然可以产生可靠的输出, 并可以定位到出错的执行体. 所以拟态系统表决机制要求异构执行体数$N \geq 3$.

对于$N \geq 3$的情况, 现行的表决机制一般采用大数表决, 大数表决规则为: 对输出结果进行直接的一致性比较, 将相同的数据归入一个一致性集合. 如果相同的输出结果的个数在所有输出结果总数中占到超过一半的比例, 那么这个输出结果就被作为一致同意的结果(最终结果) 反馈给用户. 如果没有超过一半的比例, 则选取其最大子集中的任意数据.

若表决通过, 则在认为正确的数据集合中任意选择一份数据, 将其对应的执行体标识连同数据出错的执行体信息返回给分发器, 再由分发器将表决结果显示的正确响应数据返回给客户端; 若表决不通过, 则向分发器返回错误信息以触发鲁棒性处理.

在此规则下, 本文的外部表决流程主要包含两个部分: (1) 比对两份数据是否一致的处理流程; (2) 在多份数据中表决出输出结果的算法.

## 2 比对流程

对不同执行体响应数据的比对工作包括 3 个步骤: (1) 分发器对数据包进行预处理; (2) 外部表决器对数据包进行预处理; (3) 外部表决器对预处理后的数据对象进行格式转换、内容比对.

### 2.1 分发器数据包预处理

分发器在调用外部表决服务之前, 会对异构执行体发来的所有数据包添加额外的头部字段: (1) 用以标记每份数据对应的异构执行体的通用唯一标识符(Universally Unique IDentifier, UUID); (2) 执行体安全性系数; (3) 执行体异构属性向量. 异构执行体安全性系数和异构属性向量在第 5 节中阐释.

### 2.2 外部表决器数据包预处理

外部表决器在收到分发器添加新的头部字段的数据包后, 进行进一步处理. 将数据包中的关键头部信息、报文实体、非关键信息 (指请求头部、请求正文、非关键响应头部字段等外部表决不需要的信息) 分离出来提取或丢弃, 将提取出来的部分存入新的对象用于接下来的比对工作.

数据包预处理过程如图 2 所示.

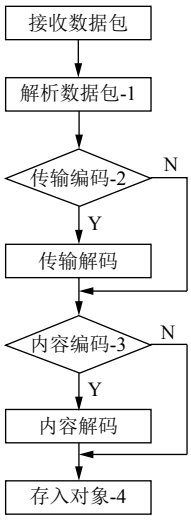


图 2 数据包预处理流程

(1) 提取出分发器新增的 UUID (作为异构执行体标识)、安全系数、异构度向量等字段, 以及数据包原有 HTTP 码和响应头部字段: Transfer-Encoding、Content-Type、Content-MD5、Content-Encoding.

(2) 若检查到 Transfer-Encoding 字段, 表示数据包经过传输编码, 依照字段值对数据包解码 (常见的如 chunked, 表明数据包采用分块传输, 须先对其剔除分块边界非关键信息并进行排序、拼接).

(3) 若检查到 Content-Encoding 字段, 表示数据包内容被压缩, 依照字段值对数据内容进行解码.

(4) 最终将 UUID、安全系数、异构度向量、Content-Type、Content-MD5 以及解码后的数据内容存入新的待比对数据对象.

### 2.3 内容比对

根据待比对数据对象中的 HTTP 码值、数据格式(结构化、非结构化) 等特征, 有相应的比对要求. 本文分别采取专用的流程和算法对待比对数据进行比对工作, 比对出每两份数据是否一致.

对特定格式的数据还要进一步定位出其内容不一致部分, 用于后续的数据清洗和攻击行为分析等工作. 预处理后的数据之间进行一致性比对的流程如图 3.

(1) 待比对数据为预处理后的响应数据对象.

(2) 比较两个对象中的成员属性 HTTP 码, 若一致

则进行下一步, 若不一致则直接生成结果.

(3) 根据成员属性 Content-Type 的值, 判断该数据是否属于结构化数据 (HTML、XML、JSON)/文本, 若属于则继续判别, 若不属于 (如图像、视频、音频等) 则一律按照二进制格式处理.

(4) 对于二进制流数据, 计算其 MD5 码值.

(5) 对于 MD5 码摘要值比对, 没有必要定位出其差异部分, 所以直接使用字符串比对算法, 完全一致则通过.

(6) 若数据属于 HTML/XML 格式, 则将其转换为 JSON 格式.

(7) 若数据为 JSON 格式, 则将其解析为 JSON 内存对象, 将 JSON 内存对象生成格式化内存文本, 再按照文本格式处理.

(8) 在结构化数据中可能会出现一些非关键信息.例如为了抵御重放攻击, 在数据中加入的流水号、随机数以及时间戳等信息, 这些字段会影响数据的比对——两份相同的数据会由于这些非关键信息而被判定为不一致. 因此在进行比对前要先剔除包含这些信息的结构化字段.

(9) 对文本格式数据以及转换为 JSON 文本的结构化数据调用 diff 算法比较文本是否一致, 并定位出出错部分.

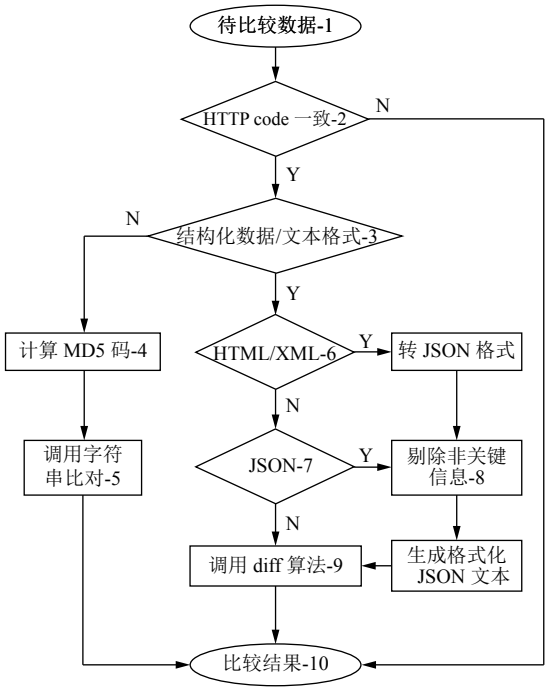


图 3 内容比对流程

(10) 比对结果包括两部分, 一是将两份数据是否一致的信息记录在公共数组中. 二是对每个数据对象的修改: 对于直接判错的数据, 将其单独归为一个集合, 不再进行内容比对, 表决时, 其置信度置 0; 记录 diff 算法定位出出错位置.

比对结束后, 记录所以数据的一致性情况, 等待所有比对完成后进行表决.

## 3 表决算法

对多份数据进行表决时, 最直接的方式是两两比对法, 对所有数据两两比对后, 将比对一致的数据归入一个集合. 当 $N$=3 时, 这种方法很适用, 但当 $N$>3 时, 两两比对法会产生很多多余的工作, 拖慢表决速度. 因此本文使用聚类比对法进行改进.

### 3.1 聚类法

当 $A$ 数据与 $B$ 数据比较一致后, 即可将两份数据归入一个一致性集合 (不必像两两比对法那样等到全部比对完才划分), 可设定为 $A$ 集合 (或 $B$ 集合), 当 $C$ 数据与 $A$ 数据比对一致后也归于 $A$ 集合, 若不一致则另建 $C$ 集合. 无论一致与否, $C$ 都无须再与 $B$ 比对, 节省时间开销. 之后的数据依次与各组的一份数据比对, 相同则归入该集合; 若全部不同, 则新建一个集合.

在聚类过程中, 同步检查各集合的元素数, 一旦有某组满足表决通过条件或整体满足表决不通过条件, 则立即向分发模块返回表决结果. 返回结果后继续完成所有数据的比对工作, 所有数据比对结束后将找出的所有出错数据对应的执行体 UUID 或表决不通过的信息返回给分发模块.

### 3.2 并行化改进

聚类法的效率显然较两两比对法有所提升, 然而在执行体数量较大时, 会出现很多数据排队等待比对的情况. 对此本文提出并行聚类法进行改进.

对于 $N$ 份待比对数据, 表决器分配$\lfloor N/2 \rfloor$个线程, 由主线程 (0 号线程) 控制数据分配. 对 $N$ 份数据按照到达先后顺序编号, 范围为[0, $N-1$].

算法流程如下:

(1) 建立一个分类集合数组: 记录现有的分类集合标记 (初始为[0, $N-1$]), 初始长度为 $N$.

(2) 对每份数据建立一个未比较数组: 记录尚未与之比较过的所有其他数据的编号.

(3) 建立一个各集合元素数量数组: 实时记录当前

各集合元素数.

(4) 设当前有 $a$ 份待比较数据, 按分配规则 (见下文分配规则) 对⌊$a$/2⌋个线程分配本轮需要进行比较的数据编号.

(5) 每次比较后, 将结果传入主线程, 更新所有数组 (见下文更新数组), 判断表决是否完成 (同聚类法). 若满足条件则向分发模块返回表决结果. 然后继续完成所有比对.

分配规则为: 主线程根据“分类集合数组”为⌊$a$/2⌋个线程顺序分配两个数据, 对某一线程选择分配的第一个数据时检查其“未比较”数组, 在选择另一数据时跳过比较完成的数据.

更新数组方式为: (1) 分类集合数组更新: 每对数据比对后, 若结果一致则将其归入同一个集合, 集合编号取较小的编号, 在数组中删去较大的集合编号; 若不一致, 则保留两份数据各自的编号进入下一轮比对; (2) 未比较数组更新: 对于每份数据而言, 其初始未比较数组含有其他所有数据的编号, 共计 $N$–1 个; 每次比较后, 对参与比较的数据在其对应数组中删去与本轮与之比较的数据编号.

## 4 改进的表决通过规则

### 4.1 大数表决规则的局限性

实验发现, 在大数表决规则下, 系统经常出现表决不通过, 甚至表决出错误结果的情况 (当实验中设置的错误数据较多时). 经过分析, 发现简单的大数表决规则存在以下问题:

(1) 简单大数表决机制默认每个异构执行体的置信度是相同的, 并且异构执行体之间遭受攻击是相互独立的. 而事实上各异构执行体使用的硬件、系统、容器、应用等软硬件设施的安全性必然是有所不同的; 另外异构执行体之间不可能完全异构, 这就导致执行体之间的一些同构性可能会导致它们遭受到同一次攻击的影响.

(2) 由于上文的同构性问题, 可能出现某几个异构体的响应数据遭到相同的篡改. 因此 $N$ 值越大, 表决机制可信度越高, 而当 $N$ 值超过 3 后可能导致出现两个 (或多个) 数据集合元素个数相等的情况. 例如 4 个执行体响应数据, 2 个等于 $A$, 2 个等于 $B$. 简单大数表决缺乏对这种情况的表决能力.

(3) 当 $N$=3 时, 其中一个执行体在某次表决中被判定出错, 该执行体必须进行数据清洗等处理, 在替换新的执行体之前, 只剩下两个置信度相等的执行体无法进行表决, 会导致持续表决过程阻塞.

针对上述问题, 本文提出了基于历史表现和异构度修正的大数表决机制, 即在大数表决无法通过的情况下, 加入额外参数进行二次表决, 提高表决鲁棒性.

### 4.2 基于历史表现的执行体安全性系数

不同的执行体的安全性显然不同, 因此在表决中,应该赋予各执行体专属的置信度, 这将由执行体的安全性决定.

各异构执行体的安全性可以由其历史表现来评价, 根据执行体的在线运行记录, 统计出其抗攻击置信度, 在表决中加入这些信息, 能够体现基于时间迭代的表决效果.

设执行体执行次数为 $t$, 执行结果正确次数为 $r$, 则某执行体的安全系数$s = r/t$. 每次表决过后, 外部表决模块将参与表决的执行体的表现返回给分发模块, 再由分发模块上传到管理模块, 由管理模块记录各执行体的安全系数.

设执行体 $a$ 第 $i$+1 次执行的输出为$\varphi_a(i+1)$, 本次表决的正确输出为$\varphi_{\text{correct}}$, 则本次表决结束后异构体 $a$ 的安全系数修改为:

$$S_a(i+1)=\begin{cases}\dfrac{r_a(i+1)}{t_a(i+1)}=\dfrac{r_a(i)+1}{t_a(i)+1}, & \varphi_a(i+1)=\varphi_{\text{correct}}\\ \dfrac{r'_a(i+1)}{t'_a(i+1)}=\dfrac{r_a(i)}{t_a(i)+1}, & \varphi_a(i+1)\neq\varphi_{\text{correct}}\end{cases}$$

初始值$r_0$, $t_0$值设置为 10, 以防止某些执行体在统计初期出错导致其安全系数变化剧烈, 进而影响表决结果.

### 4.3 基于异构度的执行体集合置信度

异构冗余系统要求执行体完全异构以期大幅增加攻击难度. 然而, 实际上不存完全异构的执行体, 执行体之间必然存在一定的同构性, 这些同构属性可能会导致多个执行体遭受到相同的攻击. 因此在表决过程中, 比对一致的数据集合, 若其对应执行体集合互相异构性较低, 则该集合为正确输出的置信度必然较低.

本文评价执行体数据集合置信度的方式是引入集合异构度系数, 用于表决中调整集合的置信权重.

异构执行体集合异构度计算如下方法:

(1) 异构属性设置: 用一个多维向量来标记执行体的异构属性, 如表 1 所示, 以 CPU 架构和 Web 容器为

例, 若执行体 $a$ 的 CPU 为 X86 架构, Web 容器为 IIS, 则其异构属性向量为 [1, 2] (随着异构属性的增加, 多维向量可扩展). 两种属性的权重设置为 [2, 4] (本小结参数均为示例, 非实际数据).

表 1 异构属性标记示例表

| 属性 | CPU架构 | | Web容器 | |
|---|---|---|---|---|
| 种类 | X86 | 1 | Tomcat | 1 |
| | ARM | 2 | IIS | 2 |
| | MIPS | 3 | PWS | 3 |
| | Atom | 4 | Jboss | 4 |
| | PowerPC | 5 | Weblogic | 5 |
| 权重 | 2 | | 4 | |

(2) 执行体相互异构度计算: 假设异构体 $b$ 的异构属性向量$v_b$ = [2, 3, 1, 2], 异构体 $c$ 的异构属性向量 $v_c$ = [4, 3, 1, 1], 属性权重向量$v_{\text{weight}}$ = [2, 2, 3, 1]; 将$v_b$和$v_c$按位异或得$v_{bc}$ = [1, 0, 0, 1]; $b$、$c$ 相互异构度 $I_{bc}$ = $v_{bc}\cdot v_{\text{weight}}^{\text{T}}$ =3; 在本例中两个完全异构的执行体的异构度$I_{\max}$ = $v_{\max}\cdot v_{\text{weight}}^{\text{T}}$ = 8, $v_{\max}$为全 1 多维向量; 因此归一化后的 $b$、$c$ 异构度$I_{bcn}$ = $I_{bc}/I_{\max}$ = 0.375.

(3) 集合异构度计算: 假设某个 $n$ 元一致性数据集合$\theta_x$ = [$a$, $b$, $c$, $d$,⋯], 其集合异构度为其中所有元素相互异构度的平均值, 计算公式为:

$$I_{\theta_x}=\frac{\sum_{i,j\in\theta_x,i\neq j} I_{ijn}}{C_n^2}$$

### 4.4 最终规则

设某次比对完成后, 存在若干个元素数相等的一致性集合$\theta_1$ = [$a$, $b$, $c$,⋯], $\theta_2$ = [$d$, $e$, $f$,⋯],⋯

每个集合对应的置信度为:

$$w_x=\alpha\sum_{i\in\theta_x}S_i+\beta I_{\theta_x}n_x$$

公式由两部分组成, 加号左侧为集合$\theta_x$中元素安全系数之和, 加号右侧为集合$\theta_x$异构度与其中元素个数$n_x$之积; $\alpha$和$\beta$为折扣因子. 其中置信度最大的集合设为$\theta_{\max}$, 则表决最终输出为$\varphi_{\text{correct}}=\varphi_i, i\in\theta_{\max}$.

在这样的机制下, 表决输出结果的置信度更高, 且为出现若干集合元素数相等而无法表决出结果的情况提供了鲁棒性机制, 提高了拟态系统的容错性.

## 5 实验验证

设计实验验证: (1) 本文改进的基于历史表现和异构度的大数表决规则是否总是能够产生表决结果; (2) 本文提出的并行聚类算法是否能够提升比对速度.

### 5.1 实验环境

操作系统: CentOS; 处理器: Intel(R) Core(TM) i7-3740QM CPU @ 2.70 GHz, 4 核; 内存: 4 GB.

### 5.2 实验方案

准备不同数据格式的 HTTP 数据包, 并手动添加 UUID 等字段, 模拟执行体响应数据; 编程实现数据发送模块, 模拟分发器向外部表决模块发送数据; 按照本文提出的表决策略对 $N$ 份数据采用不同的表决算法进行表决, 获得表决结果并测量运行时间和 CPU 占用情况等性能指标.

手动修改数据内容, 模拟受攻击篡改的异构执行体响应数据; 随机设定执行体基于历史表现的安全性系数和异构向量两项参数; $N$ 值分别取 3、5、7、9, 以研究优化的表决算法在异构体数量增加时的加速比变化情况; 分别采用第 4 节中介绍的两两比对法、串行聚类比对法和本文提出的并行聚类比对法进行实验并比较, 研究并行聚类法的运行效果.

模拟两种情况: (1) 最好情况即所有数据一致; (2) 较差情况即 50%–70%(根据执行体数调整) 的数据出错. 针对两种模式各迭代 1000 次, 计算表决通过率、运行耗时和 CPU 占用情况.

### 5.3 实验结果

实验结果如表 2 所示. 当执行体数量为 3 时, 表决通过率未达 100%. 这是因为此时, 当超过两份数据出错, 就会导致 3 份数据比对全部不一致, 一旦安全系数较高的 2 份 (或 3 份) 数据的安全系数刚好相等, 则无法表决出结果. 因为在此情况下不存在异构置信度系数的调节, 所以存在一定的可能性导致无法产生表决结果. 而当异构体数目较大时, 这种可能性几乎不存在, 且在实际应用中, 错误率远低于本文的实验设定, 表决不通过的可能性进一步降低.

由图 4、图 5 可以看出, 随着执行体数目增加, 并行聚类法的加速效果越明显, 尤其在较差情况下, 两两比对法和聚类法时间消耗值呈现指数增长, 而并行聚类法趋向于线性增长, 使得运行效率显著提升.

### 5.4 结果评价

在最好情况下, 本文提出的并行化的聚类表决算法在异构执行体数较少时, 由于不能明显地减少比对的轮数, 加上并行化引起的通信和管理开销增加, 导致效率不仅较普通聚类表决法没有提升反而有略微的下

降. 但随着执行体数目的增加, 并行聚类法对两两比对法和串行聚类法的加速比不断提高, 显著地缩短了表决运行时间, 虽然 CPU 占用率成倍提高, 但占用数值在实验中最高只在 3% 左右, 可以忽略不计.

表 2 表决算法测试结果

| 执行体数 | 算法 | 表决通过率 (%) | 最好情况耗时 (ms) | 较差情况耗时 (ms) | CPU占用 |
|---|---|---|---|---|---|
| 3 | 两两比对法 | 98.6 | 5957 | 8552 | 几乎无 |
| | 聚类法 | 98.9 | 5979 | 8869 | 几乎无 |
| | 并行聚类法 | 98.7 | 6215 | 9134 | 几乎无 |
| 5 | 两两比对法 | 100.0 | 12065 | 20349 | 几乎无 |
| | 聚类法 | 100.0 | 9212 | 15040 | 几乎无 |
| | 并行聚类法 | 100.0 | 9636 | 12002 | 几乎无 |
| 7 | 两两比对法 | 100.0 | 18820 | 40905 | 几乎无 |
| | 聚类法 | 100.0 | 12855 | 25879 | 几乎无 |
| | 并行聚类法 | 100.0 | 9807 | 18417 | 稍有 |
| 9 | 两两比对法 | 100.0 | 24014 | 76709 | 几乎无 |
| | 聚类法 | 100.0 | 15946 | 43047 | 几乎无 |
| | 并行聚类法 | 100.0 | 12241 | 26290 | 稍有 |

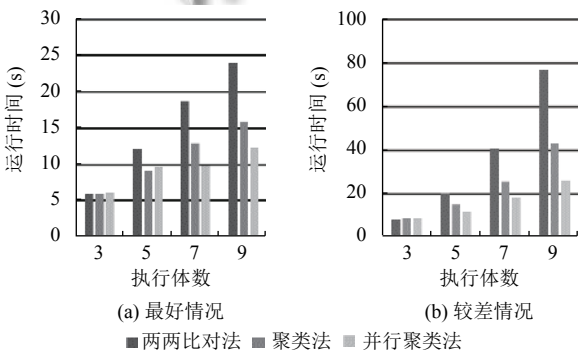


图 4 两种情况下表决算法运行时间比较

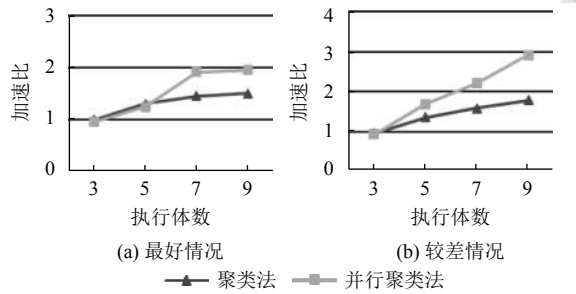


图 5 两种情况下聚类法和并行聚类法加速比变化趋势

在较差情况下由于很多数据不一致, 很多数据无法通过归类而跳过比较, 所以串行聚类法缩短运行时间的幅度有限. 而并行聚类法由于几乎没有数据被闲置的情况, 依然可以获得很明显的加速效果.

在实际系统运行中, 数据出错概率较低, 本文实验较差情况是一种极端状况. 因此聚类法能够在绝大部分情况中有效地提升运行效率, 而并行化改进后运行速度得到进一步提高, 在较差情况下也能大幅缩短运行时间, 在执行体数量较多时, 优势尤其明显.

## 6 结束语

本文基于历史安全性系数和集合异构度修正的大数表决机制实现了拟态通用运行环境外部表决的要求. 与现行的外部表决机制相比, 本文改进的大数表决机制使得表决准确度更高, 系统容错能力更强. 此外, 并行化改进的聚类表决算法能够解决串行表决中的数据排队问题, 大幅提升了表决速度. 而针对不同格式数据的比对流程也使得数据比对、表决更加可靠.

这 3 项改进或设计能够充分发挥拟态信息系统动态、异构、冗余的特点. 具有一定的理论价值和实际应用价值.

未来研究方向: (1) 对执行体异构属性进行扩展, 并加入新的态势信息参与表决, 进一步提高表决机制的可靠性、容错性; (2) 针对图像、视频、音频等非结构化数据制定专用的比对流程以定位出错误部分; (3) 针对其他协议下的大文件、视频流传输等情况, 研究持续表决机制, 即针对同一文件进行分块表决.

## 参考文献

1 王伟, 曾俊杰, 李光松, 等. 动态异构冗余系统的安全性分析. 计算机工程, 2018, 44(10): 42–45, 50.
2 张振峰, 任卫红, 朱建平. 重要信息系统安全威胁及防范对策分析. 信息网络安全, 2010, (7): 25–28. [doi: 10.3969/j.issn.1671-1122.2010.07.012]
3 Jajodia S, Ghosh AK, Coyan V, *et al*. Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats. New York: Springer, 2011, 54.
4 钱春沁. 建立企业重要信息系统应急响应和灾难恢复机制的若干构想. 保密科学技术, 2016, (7): 35–38.
5 邬江兴. 网络空间拟态防御导论. 北京: 科学出版社, 2017.
6 斯雪明, 王伟, 曾俊杰, 等. 拟态防御基础理论研究综述. 中国工程科学, 2016, 18(6): 62–68.
7 霍立田, 邵培南, 徐李定, 等. 拟态通用运行环境的资源管理与调度技术. 计算机工程, 2020, 46(2): 159–169.
8 付琳, 邵培南, 应飞, 等. 拟态通用运行环境的框架设计. 计算机工程, 2020, 46(3): 24–33.