云服务可靠性优化方法研究

梁员宁 陈建良 叶 笠
(中国人民解放军78179部队 成都 611830)

摘 要 针对云服务的冗余特性和可靠性保障的需求,探讨了提高云服务可靠性的有效途径。基于云服务的可靠性体系框架和管理模式,提出了信任冗余的云服务可靠性增强总体框架;在服务准备时的冗余设计阶段,基于选举协议的云服务轮询检测机制,设计了信任感知的容错服务选择算法,给出了最小容错服务个数的求解方法;基于服务组合运行时的容错处理框架,提出了保证服务响应时间的基于失效规则的云服务调用策略。实验结果表明,提出的容错服务选择算法和云服务调用策略具有较好的实用性和有效性。

关键词 云服务,可靠性设计,信任冗余,失效规则
中图法分类号 TN393 **文献标识码** A

Research on Reliability Optimal Method of Cloud Service

LIANG Yuan-ning CHEN Jian-liang YE Li
(Army Unit 78179 of PLA, Chengdu 611830, China)

Abstract In allusion to the redundant features and reliability guarantee requirements, this thesis tried to explore the effective ways for improving cloud service reliability. A trust redundancy reliability enhancement framework of cloud service was put forward based on the reliability architectue and management model. On the redundant design phase of the service being ready, based on the failure polling detection algorithm of voting protocol, a trust-aware fault-tolerant service selection algorithm was designed, and the calculation methods of min-number of fault-tolerant services were given. Based on the running fault-tolerant management framework of services combination, a cloud service invoking policy of ensuring service response time based on failure rules was presented. Experiments show that the fault-tolerant service selection algorithm and the cloud service invoking police have more practicality and effctiventness.

Keywords Cloud service, Reliability design, Trust redundancy, Failure rule

目前,基于可靠性属性的服务质量保证研究往往只涉及服务生命周期管理的某一阶段,大多数研究也只停留在理论探索阶段,与实际的应用存在较大的差距。容错是提供可靠性和可用性的关键机制,它使得系统在部分构件失效发生时仍然能够正常运行。在传统的软件开发过程中,软件容错技术的实现代价很高,只有极少数的关键系统采用软件容错技术增强系统可靠性。当前,针对服务组合的容错算法研究主要体现两方面在:一是通过扩展服务的标准协议,引入服务异常处理机制进行多级异常处理来保障可靠性;二是研究开发支持容错的服务运行平台。文献[1]在SOA概念的基础上提出了利用容错的思想,通过服务平台的失效检测、服务复制和日志的方法实现服务的容错机制;文献[2]通过扩展标准的服务描述协议WSDL,引入服务组的概念,利用服务被动复制的容错模式保障可靠性;文献[3]通过修改服务运行的服务器操作系统和内核信息,建立用户透明的服务错误容忍访问机制。

云服务作为一种新兴的分布式服务计算模式,面向的是广域的服务运行环境,服务的超大规模性、高度复杂性、分布自治特性和动态网络变化特性对在云服务组合中应用容错技术带来了挑战。在服务组合执行过程中,服务的可靠性可能会受到网络环境的动态不确定性、服务交互通信可靠性的改变、服务遭遇恶意攻击拒绝、服务基础设施故障、服务交互协议的低性能等因素的影响,使得服务组合流程的制定和服务质量难以得到保障,因此需要对服务组合的运行提出有效的容错保证机制,如何保证云服务的可靠性已成为组合云服务能否成功应用的关键。

1 云服务可靠性增强总体框架

1.1 可靠性管理体系框架

云计算服务环境下的信息处理系统以可共享与可集成的自治元服务资源为基本构造单元,强调服务的松耦合、可重用、易组合和虚拟动态优化,本质上是将分布在自治软硬件基础设施资源上的元服务,通过相关协议实现服务的发布、交互、整合与重组,并协调云服务提供者的行为,为云服务使用者提供增值的服务。云服务环境的虚拟动态性和服务的随机性,使得服务系统必须能够动态适应和处理不可预知的变化,为海量的云服务用户提供可靠的信息服务质量。为此,需要

到稿日期:2012-10-26 返修日期:2013-02-07 本文受国防预研基金项目(9140A26010306JB5201)资助。
梁员宁(1986—),男,主要研究方向为可靠性与信息安全,E-mail:lyn577@163.com;陈建良(1968—),男,主要研究方向为计算机网络安全;叶 笠(1972—),男,博士,主要研究方向为计算机网络安全。

对具有虚拟动态变化特性的云服务系统的运行状态进行可靠性分析评估，通过构建服务系统的可靠性模型，评估环境变化对服务系统运行可靠性的影响，判定服务能否满足用户的可靠性需求，进而开展服务可靠性保障机制和方法的研究，增强服务的可靠运行能力。云服务是一种基于 SOA 架构的运行模式，根据服务运行的生命周期可分为服务需求分析阶段、服务准备阶段、服务提供与实现阶段和服务运行维护阶段，云服务的可靠性体系架构如图 1 所示。

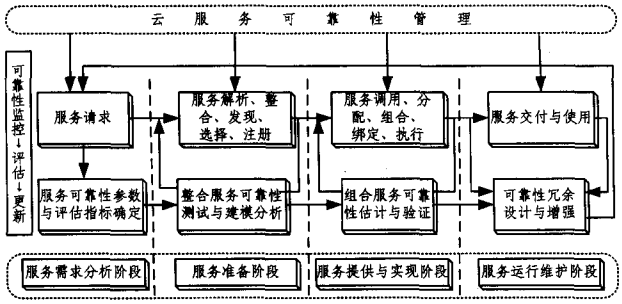

图 1　云服务的可靠性体系框架

在服务需求分析阶段，需根据用户提出的服务请求确定服务可靠性参数与评估指标；在服务准备阶段，需采用服务可靠性测试、建模分析技术和确定可能导致整合服务失效的隐患、薄弱环节，将可靠性设计进去；在服务提供与实现阶段，则需对服务可靠性进行有效评估、验证，以确定是否可以交付服务，是否满足服务请求者的要求；在服务运行维护阶段，依据服务的运行状态和需求进行可靠性冗余设计，从而实现服务的可靠性增强。

1.2　服务可靠性管理模式

在用户进行云服务资源访问操作过程中，对一个服务请求反馈回的结果往往是一类功能属性相同、非功能属性有差异的云服务[4]。为此，针对云服务的随机性和云服务环境的动态性特点，构建了云服务的可靠性管理模式图，如图 2 所示。通过对云服务监控器在线实时监测记录云服务的可靠性数据信息进行各可靠性指标评估预测，动态自适应更新云服务的可靠性，为用户提供服务可靠性的参考数值。该结构模型的可靠性指标可依据需要进行扩展，当有新的指标加进来时，不需要对建立在该模型上的服务选择技术进行修改。

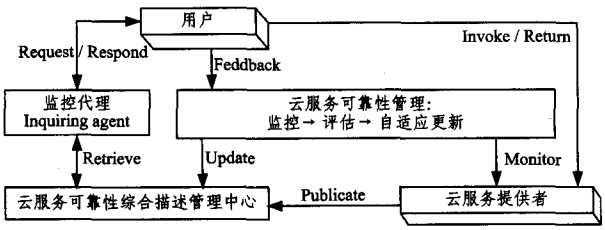

图 2　云服务的可靠性管理模式图

云服务的可靠性管理器首先对服务请求执行任务和云服务之间的通信数据通过云监控系统进行检测。云监控系统通过部署在物理机/虚拟机上的监控代理程序(agent)收集实时的、细粒度的监控数据，通过分析监控数据，发现应用的特点和模式，帮助用户定位性能瓶颈，支持高效的读写和循环利用存储空间，其具有实时、历史数据获取、基于 Web 数据可视化的特点。云监控系统中记录有与云服务的可靠性有关的数据信息，如单位时间内到达的任务数和处理完毕的任务数、故障发生时间和故障修复时间以及周期性监测到的一些实际执行信息等等，然后云服务可靠性管理器根据监控系统检测的结果对服务的各可靠性指标进行评估。

通过对整个云服务系统进行全面的在线监控，使得云服务系统管理员能够时刻掌控整个云服务资源的性能，确保满足用户对服务的可靠性需求。监控通常是通过监控软件来实现的，对重要系统资源进行监控，检测出瓶颈和潜在的可靠性问题并在严重的情况下进行自动恢复，才可以支撑云计算平台的灵活性和高可靠性。监控的对象一般包括系统硬件设备(服务器、存储、网络等)，也包括软件(应用程序、数据库、中间件等)。云服务系统通常会包括多种类型、多种结构以及多种品牌的硬件和软件，支撑云计算平台的监控软件需要较高的兼容性，能够同时监控异构的硬件设备和软件。

1.3　可靠性增强总体框架

在云服务环境下，由于存在着大量功能相同的元应用服务，云服务组合可以充分利用这些服务的冗余特性，在不太可靠的单个元应用服务的基础上构建可靠的组合服务。为了保证云服务组合的可靠性，针对云服务运行环境的特点，从服务的响应时间和可靠性保障方面对服务组合的设计和业务流程的管理进行考虑，本文基于云服务的可靠性评估模型提出了信任冗余的可靠性优化方法。基于信任冗余的云服务可靠性增强总体框架如图 3 所示，该框架主要由服务请求者、云服务提供者、云服务代理、云服务调用模块(cloud service transfer module)、冗余服务选择模块(redundancy service select module，RSSM)和可靠信任管理模块(reliable trust management module，RTMM) 6 部分组成。

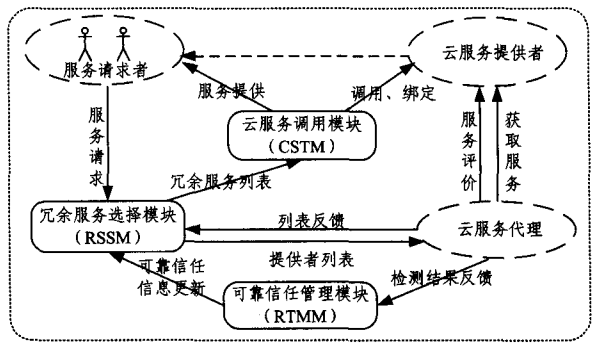

图 3　靠性增强总体框架

步骤 1　当服务请求者提出某种服务的请求时，首先询问冗余服务选择模块 RSSM，并提供所需服务的功能性需求 f 和非功能性能指标要求(如可靠性的置信度 λ)；

步骤 2　冗余服务选择模块 RSSM 根据请求者所需服务的功能特性 f 查询相应的服务领域，在满足服务功能 f 要求的基础上综合考虑置信度为 λ 和服务领域信任状况推选适量的云服务提者，并将提供者列表、检测列表一并发送给云服务代理；

步骤 3　云服务代理根据冗余服务选择模块 RSSM 提供的服务提供者列表，与相应的服务提供者进行交互以获取所需服务，在服务交互完成后，对服务的可靠信任情况进行评价，并将检测结果反馈给可靠信任管理模块 RTMM；

步骤 4　可靠信任管理模型对云服务提供者列表信息更新后返回至冗余服务选择模块；

步骤 5　冗余服务选择模块 RSSM 将满足要求的冗余服

务提供者的列表发送至云服务调用模块 CSTM；

步骤 6　云服务调用模块 CSTM 根据冗余服务列表按照设定的调用策略将对服务进行调用与绑定，为服务请求者提供可靠的服务。

在冗余服务的可靠性信息更新过程中，使用了轮询机制来保证冷僻和低可靠信任的服务提供者的可靠信任信息的新鲜性；通过轮询机制，使该服务领域所有服务提供者的可靠性信息的准确度得到了有效提高。

2　服务准备阶段时的冗余设计

在服务准备阶段，云服务代理和云服务综合管理中心需要共同完成服务使用者的服务执行请求的解析、接纳、请求服务的整合、发现与选择、对服务注册信息的登记与更新操作等，根据用户个性化的服务请求，通过整合分布式的不同软硬件元服务形成服务资源池，为用户提供满足需求的服务资源。为了保证服务在运行生命周期中出错时能快速恢复正常运行状态，需要在服务准备阶段设计服务失效检测机制和冗余服务选择算法。失效检测是对服务系统应用容错技术提高可靠性首先要解决的问题，只有检测到服务失效，才可能进行服务的重新定位和恢复。

2.1　服务失效检测机制

为了能够在服务准备阶段从服务提供者获得更多满足需要的有效服务，假设所有服务提供者以未知可靠状态、信任可靠状态和混合状态 3 种基本状态形式存在。未知可靠状态的服务提供者提供的服务需要进行检测才能作为选择的对象，可靠信任度较低，服务结果不能用作其可靠性的评估依据，其作为服务失效检测的待测者；信任可靠状态的服务提供者提供的服务能够长期保持很高的可靠信任度，服务结果能直接作为可靠性评估的依据，其作为服务失效检测的检测者；混合状态的服务提供者提供的服务其可靠信任度处于未知状态和可靠状态之间，兼备待测者和检测者的职能[5]。

对于服务的失效检测问题，主要是利用轮询和心跳两种模式对被测实体进行检测，采用超时机制来判断服务的失效情况。由于服务提供者的可靠信任状态会随着服务运行环境的变化而发生改变，长时间不检测可能会使得其值不够准确，因此需要利用轮询检测机制对服务提供者的状态进行失效检测，基于选举协议的云服务轮询检测机制的算法如下。

步骤 1　通过冗余服务选择模块 RSSM 将 m 个可靠信任的待测者添加到检测列表返回给云服务代理，m 为轮询待测者个数，根据服务领域的特点和服务请求的频率，m 的数学计算公式为：

$$m=\min(\mathrm{Max}(m),\left\lceil\frac{p+q}{Time_i Sqf_i}\right\rceil)$$

其中，$Time_i$ 是在服务领域 i 中提供者保持可靠信任状态的最久时间；Sqf_i 为服务领域 i 的服务查询频率；p 和 q 分别表示服务领域中处于可靠信任混合状态和未知状态的数量；$\mathrm{Max}(m)$ 表示最大的轮询待测者个数。

步骤 2　云服务代理和同一服务领域中的多个云服务提供者进行服务交互。

步骤 3　云服务代理在特定的时间 t 内获得的云服务提供者的有效服务数据信息集为 C，$C=\{c_1,c_2,\cdots,c_k\}$，$k\in\mathrm{N}^+$，$l\leqslant n$，其中 n 表示冗余服务提供者的数量，当询问返回的数据结构值超过 $n/2$ 满足要求时，云服务代理即获得正确的服务结果 $result$，则有：

$$\forall c_i,c_j\in C_d$$
$$\mathrm{if}(diff(c_i(value)-c_j(value))<\varepsilon \text{ and } |C_d|>n/2)$$
$$\mathrm{then}\ result=avg(C_d)$$

其中，ε 为允许出现的最大误差值。

步骤 4　一一比较服务的返回结果值 $result$ 和各云服务提供者的结果 $value$，并依据相似程度将其标记为具有不同等级程度的正确性结果。

步骤 5　云服务代理和所有服务领域的云服务提供者进行交互，并将结果标记值返回给可靠信任管理模块 RTMM。

步骤 6　可靠信任管理模块 RTMM 通过服务检测结果更新可靠信任状态待测者的可靠信任度，将其作为结果值。

2.2　容错服务选择算法

信任感知的容错服务选择(Trust Aware Fault Tolerant Services Seletion，TAFTSS)。在传统的容错算法中，容错服务的个数往往是某一固定值，对于大规模的服务系统，为了满足可靠性要求，其值往往取得较大。这种盲目设定固定值的方法，虽然保证了服务的可靠性，但其成本开销也在增加，同时大量的容错服务，使得在服务选择时增加了难度，且在实际应用中，往往并不需要太多的冗余服务个数就能提供满足要求的可靠性服务。TAFTSS 算法主要针对云服务环境的动态变化特性，考虑了服务领域服务提供者的可靠信任状态，通过数学的方法计算满足服务请求者的可靠性需求的最小服务容错个数，并为服务请求者提供满足其请求的容错服务提供者列表。在利用选举协议进行服务失效检测时，若获得的服务结果中有一半以上满足可靠性要求，其结果即可作为正确值。假设有 m 个服务提供者，即有：

$$C_d\geqslant\max(\left\lceil\frac{m+1}{2},2\right\rceil)$$

设在对 m 个服务提供者进行选举轮询时，有 i 个提供了满足可靠性要求的服务，其概率为 $C_m^i\overline{P}^i(1-\overline{P})^{m-i}$，$\overline{P}$ 表示 m 个服务提供者的可靠性平均值，依据多数原则，得到满足要求服务的概率为：

$$Rf(m,\overline{P})=\sum_{i=\max(\left\lceil\frac{m+1}{2},2\right\rceil)}^{m}C_m^i\overline{P}^i(1-\overline{P})^{m-i}$$

当 $Rf(m,\overline{P})$ 的值大于等于服务请求者提出的可靠性需求且同时满足 $Rf(m,\overline{P})\geqslant\lambda$ 和 $Rf(m-1,\overline{P})<\lambda$ 时，得到的 m 值即为服务的最小容错数。

基于信任感知的容错服务选择的基本思想是：在进行服务选择时，优先考虑从可靠信任的服务提供者中进行选择，若可靠信任提供者提供的服务数目达不到最小容错服务的要求，则可通过混合者提供的服务进行选择，直到满足服务请求者的可靠性要求为止。TAFTSS 算法的描述如下。

步骤 1　在可靠信任的服务提供者中进行服务选择，由于最小容错服务 m 的数量一般不会太大，可通过使用穷举法获得冗余数 m，使其满足下式：

$$\begin{cases}Rf(m,\overline{P}_a)\geqslant\lambda\\Rf(m-1,\overline{P}_a)<\lambda\end{cases}$$

步骤 2　当容错服务的个数 m 小于等于可靠信任提供者的个数 am 即 $m\leqslant am$ 时，可靠信任提供者能够满足服务请求者的可靠性需求，可以从可靠信任提供者中随机选取 m 个服务作为容错服务选择列表的候选服务，转到步骤 5。

步骤 3　当计算得到的容错服务个数 m 大于可靠信任提供者的个数 am 即 $m>am$ 时，需要调用处于可靠信任混合状态的服务提供者的服务，这在一定程度上降低了可靠性的标准，通过利用穷举法即可获得满足需要混合状态的服务提供者数量 cm，满足下式：

$$\begin{cases} i=\max\limits_{\left(\left\lceil\frac{an+cn}{2}\right\rceil,2\right)}^{an+cn}\ \sum\limits_{j=0,j\leqslant i-an}^{cn} C_{cn}^{j}\bar{P}_h(1-\bar{P}_h)^{cn-i}\bar{P}_a^{i-j}(1-\bar{P}_a)^{an+j-i}\geqslant\lambda \\ \sum\limits_{i=\max\left(\left\lceil\frac{an+cn+1}{2}\right\rceil,2\right)}^{an+cn-1}\ \sum\limits_{j=0,j\leqslant i-an}^{cn-1} C_{cn-1}^{j}C_{an}^{i-j}\bar{P}_{cn-1}^{j}(1-\bar{P}_{cn-1})^{cn-1-i}\bar{P}_a^{i-j}(1-\bar{P}_a)^{an+j-1-i}\geqslant\lambda \\ \bar{P}_{cn}=\dfrac{\sum\limits_{i=1}^{cn}PHt_i}{cn} \end{cases}$$

其中，an 和 cn 分别为可靠信任者和混合者的数量，PHt_i 为可靠信任度第 i 高的混合者。

步骤 4　将全部可靠信任者和可靠信任度最高的 cn 个混合状态服务提供者加入容错服务选择列表。

步骤 5　将容错服务选择列表反馈至云服务代理，完成容错服务选择。

3　服务组合运行时的容错处理

在组合服务的运行过程中，服务的失效一般有 3 种形式：一是节点崩溃故障，主要是指构成组合服务的服务组件节点发生了失效，导致该节点的服务功能消失；二是服务交互通信失效，主要是指由于服务之间交互信息的通信链路因网络故障等造成的消息丢失、服务提供者的暂时故障导致消息交互产生延迟等；三是误差失效，对于包含有循环结构、选择结构和分支结构的组合服务，由于在选择分支时采用的是概率统计经验值，而组合服务模型中对于分支的选择直到服务运行时刻才能确定执行路径，因此不可避免地会与服务实际运行时的可靠性质量产生一定的误差。

3.1　容错处理框架设计

为了保证服务组合运行时的可靠性，建立服务组合运行时的容错处理框架，如图 4 所示，该框架主要由最优化分析器、服务配置代理、组合服务重构模块、服务执行引擎、服务流程监测器、服务决策器和服务流程调节器等部分组成。在云服务环境中，由于服务的行为具有不可预测性，需要通过云服务组合流程的失效监测器来保证组合服务能够按照服务水平协议(Service Level Agreement，SLA)正确执行；为了更好地支持组合服务运行时容错，需要利用最优化分析器对抽象的工作流进行预先优化处理，在服务流程进行重构的过程中，通过服务流程调节器自动选择优化策略完成容错处理。

服务的容错处理需要根据服务组合生命周期各阶段的特点，采用基于可靠性感知的不同的服务优化选择算法。在服务组合流程的执行阶段，对候选服务的选择、服务的动态调用与绑定、服务的重试、服务替换和服务流程重构的操作主要由云服务代理通过调用服务执行引擎完成；在对服务流程进行重构后，最优化分析器针对不同服务组合的业务流程特点，采用自适应的可靠性动态优化算法，对于具有线性关系的服务组合模型，一般采用基于启发式 0-1 整数规划法对其进行在线实时优化设计，对于具有非线性关系的模型，则选择遗传算法对其进行可靠性优化。在服务组合执行过程中产生的可靠性数据信息将通过服务流程监测器和云服务代理反馈至服务存储库中，形成可靠性属性数据统计列表，为评估服务的可靠性提供数据依据。

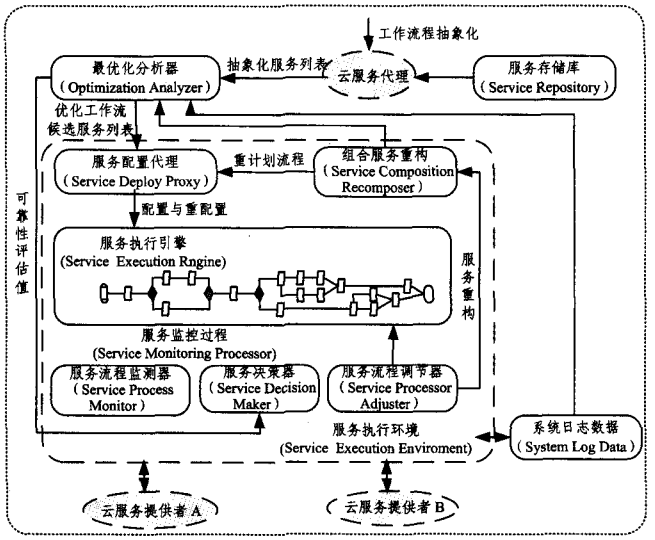

图 4　服务组合运行时的容错处理框架

3.2　云服务的调用策略

在云服务环境下，不同的用户对服务资源的需求模式不同，用户动态申请的服务资源往往呈现出一种波动性，不同的组合服务模型具有不同的服务故障特征。为了保证组合服务始终保持在高可靠运行状态，又不以牺牲服务系统整体服务资源的使用率和服务的可靠性质量为代价，在综合考虑服务资源需求和服务失效规则的基础上，提出一种保证服务响应时间的基于失效规则的云服务调用策略(Cloud Service Invoking Policy Based on Failure Rules，CSIPBFR)。CSIPBFR 策略不涉及云服务平台的硬性修改，能够使冗余服务列表中提供的服务保持在最可靠状态，从而为整个组合服务的可靠性运行提供机制保障。为了考虑动态调用节点服务的可靠性问题，将组合云服务的节点服务资源或虚拟节点服务资源当作基本的资源单位，用节点服务崩溃和节点服务的无计划重起失效来衡量节点服务资源的可靠性状况。在云服务环境下，根据服务资源的使用情况和表现形式，可将各种服务大致划分为数据计算密集型的高性能计算(High Performance Computers，HPC)服务和交互密集型的网络(Web)服务[6]。

对于节点服务资源的系统失效规则，国内外学者做出了大量的相关研究：文献[7]通过对服务资源的运行系统的日志进行失效数据分析，发现节点资源失效往往呈现出很强烈的时间局限性，同时将节点服务的无计划重起失效相隔时间当作一个满足参数 $shape$ 小于 1 的 $weibull(scale,shape)$ 分布的随机过程；文献[8]对经典的 HPC 计算服务和 Web 服务混合机群服务系统进行失效检测分析，同时指出：节点服务的无计划重起失效的可能性随着服务系统的运行而呈现出先降低再增加的趋势；文献[9]通过对服务软件运行系统进行性能分析，发现服务软件系统随着服务运行时间的不断提高，系统的性能呈现下降的趋势，而软件的失效率则缓慢上升，到一定的时间，软件往往表现出老化的现象；等等。上述的节点服务资源的失效规律说明，对于刚刚发生失效的节点资源服务，其再次发生失效的概率比较大，但随着服务运行时间的增加，该节点将会表现得越来越稳定，因此，对刚刚发生失效的节点资源

服务进行可靠性检测分析对保证该节点服务的可靠性是非常有必要的;对于表现出软件老化的服务软件系统,其产生失效的原因主要是在软件内部累积了过多的内部程序错误,对于这类问题的解决,可以通过软件再生技术[10]实现,即对服务软件系统内部累积的所有错误进行清除,重新恢复服务系统至初始正常运行状态,通过有效的服务软件再生策略可以降低服务软件系统的性能损失。

在云服务环境下,节点服务资源的动态调用策略是一个On-Demand策略,传统的冗余服务调用策略往往不考虑节点服务的失效规则,而是单独维护一个节点服务资源池,当用户发出的服务请求到达时,服务决策器调用的服务策略随机地从服务节点资源池中选择空闲的服务提供给服务请求者,当服务调用处理完毕后,将服务随机地放回节点服务资源池中。这种方式在一定程度上满足用户的可靠性需求,但不能保证服务的响应时间和服务的高可靠运行。CSIPBFR策略是在综合考虑节点服务资源失效在时间上的规律的基础上,以服务运行时间为驱动方式,用于保证服务响应时间的可靠服务保证策略。CSIPBFR策略的算法描述如下:

```
    CSIP B FR_Arithmetic_Structure;
  Web_Node_List,网络 Web 服务的节点服务资源列表;
  HPC_Node_List,HPC 计算服务的节点服务资源列表;
  Node_pool_List:空闲资源池里维护的节点服务资源列表;
  HPC_Job_Running_List,处于运行状态的 HPC 计算服务的列表;
Input:
  Web_Nodel_Failure_Event,Web 服务节点资源服务的失效事件;
  HPC_Node_Failure_Event,HPC 计算服务节点资源服务的失效事件;
  pool_Node_Failure_Event,资源池节点资源服务的失效事件;
  Time_Schedule_Event,时钟事件;
Start:
  getevent;
  switch(event_type){
    casePool_Node_Failure_Event;
      putFailureNode. uptime=CurrentTime;
      Node_Pool_List. head=Operator. Head();
      putFailureNodetoNode_Pool_List. head;
      break;
    caseWeb_Node_Failure_Event;
      putFailureNode. uptime=CurrentTime;
      Node_Pool_List. head=Operator. Head();
      putFailureNodetoNode_Pool_List. head;
      break;
  case HPC_Node_Failure_Event;
    put FailureNode. uptime=CurrentTime;
    Node_Pool_List. head=Operator. Head;
    put FailureNode to Node_Pool_List. head;
    put the Job to HPC_Job_Waiting_List. tail;
    put rest nodes of Job to Node_Pool_List. tail;
    break;
  case Time_Schedule_Event;
    For(Web_Node_List){
      ++ Node. ServiceAging;
          if(node. ServiceAging >Re juvenationThreshold){
        put Node_ServiceAging=0;
          Node_Poll_List. tail=Operator. Tail();
          Put Node to Node_Pool_List. tail;
        }
    }
    get Web_Re quest_Node_Re source;
    if(RequestedNodes. size < Web_Node_List. size){
      toReleaseNode. size=WebNodeList. size-
      RequestedNodes. size;
      take toReleaseNode from Web_Node_List;
      Node_Pool_List. tail=Operator. Tail();
      release Nodes to Node_Pool_List. tail;
    } else {
      toRequestNode. size-RequestedNodes. size
      -WebNodeList. size;
      Node_Pool_List,head=Operator. Head();
      take toRequestNode From Node_Pool_List. head;
      put these Node to Web_Node_List;
  }
    For(HPC_Job_Running_List){
      if(Job. executeTime+Job. StartTime > CurrentTime){
        put Job. status=finished;
        Node_Pool_List. tail=Operator. Tail();
        release Job's Nodes to Node_Pool_List. tail;
        put Job to HPC Job Finished List;
      }
  }
  For(HPC_Job_Waiting_List){
    if(Job. Nodesize < Node_Pool_List. size){
      Node Pool List. tail=Operator. Tail();
          get Node from Node_Pool_List. tail;
          put job. status=running;
          put Job to HPC_job_Running_List;
        }
      }
      break;
    }
End
```

CSIPBFR策略维护的是按前次失效恢复时间(*uptime*)有序排列的节点服务资源池,在空闲节点服务资源池的节点服务列表的首部和尾部通过设置操作*Operator. Head*()和操作*Operator. Tail*()来对调用的服务进行合适的选择,实现节点服务资源的调用提供和回收,保证节点资源服务的高可靠。由节点资源服务的失效规则可知,对于失效服从参数*shape*小于1的*weibull*(*scale*,*shape*)分布的服务节点,可将刚刚失效的服务节点放在空闲节点服务资源池列表的*Operator. Head*()位置,当有服务请求到达时,则从空闲节点服务资源池列表的*Operator. Tail*()位置获取节点服务,保证该服务资源节点提供的服务是空闲节点服务资源池中响应时间最快、可靠性最高的节点服务;对于服务器整合的服务软件系统平台,需要考虑软件的老化问题,假定网络Web服务的失效率服从一个特定参数缓慢上升的分布,在服务软件系统运行时间到达门限值的资源服务节点时,利用软件主动再生将资源服务节点放回空闲节点服务资源池服务列表的*Operator.*

Tail()位置;对于 HPC 计算服务,节点资源服务的运行时间越长,其失效的概率越大,往往表现出服务软件老化的现象。对具有超大规模性、高度复杂性、动态变化特性和开放式的云服务环境,整个服务系统的服务资源往往会随着运行环境的改变而呈现出不同的失效规律,资源服务的失效到达也会服从不同的分布。因此,可根据服务资源的不同失效规则,按照 CSIPBFR 策略维护各自独立的空闲节点服务资源池服务列表,列表中的服务资源按照可靠性高低依次由位置 *Operator. Head*()排到位置 *Operator. Tail*(),当有服务资源请求时,可通过各自的空闲节点服务资源池服务列表选择响应时间最快、可靠性最高的服务资源,服务资源的回收则根据服务所属关系选择相应的空闲节点服务资源池服务列表。

4 实验验证及结果分析

为了能够在更加逼近真实的云服务环境中应用论文提出的评估方法对动态变化的云服务可靠性进行分析,本节依托某实验室的云计算平台搭建实验环境对容错服务选择算法和云服务调用策略的实用性和有效性进行仿真验证。

在服务准备阶段,利用服务失效检测机制发现和排除服务资源池中的假冒提供者,即在与服务请求者交互过程中提供虚假的反馈结果或是恶意的虚假服务,以达到诋毁正常服务提供者的目的。本文通过实验测试一般容错方法和信任感知的容错方法对假冒提供者攻击的抵抗能力。假设假冒提供者的比率为[0,90%],每增加 10%进行 100 次实验,计算平均的服务容错数和服务成功率。一般容错方法和信任感知的容错方法所需的容错服务数如表 1 所列,前者是通过理论的方法计算获得。当假冒提供者比率缓慢增加时,信任感知的容错方法的平均容错数增速比较平缓,基本保持不变,这体现了其对于服务提供者环境具有较好的平稳性能。而一般容错方法在即使没有假冒提供者的运行环境中都至少需要 5 个服务提供者,且其容错数量随着提供者比率的增加而呈现快速增长趋势,当假冒服务提供者比率超过 30%时,其值变得相当大,已不具有实现性。

表 1 容错服务数量和假冒提供者比率的相互关系

假冒提供者比率	信任感知的容错方法	一般容错方法
0	4.35	5
10%	4.62	8
20%	4.94	19
30%	5.23	45
40%	5.41	127
50%	5.97	200+
60%	6.23	200+
70%	6.56	200+
80%	6.74	200+
90%	6.83	200+

使用一般容错方法和信任感知的容错方法的提供者比率和成功率的关系如图 5 所示。随着假冒提供者比率的增加,一般容错方法的服务成功率呈下降趋势,幅度比较大,不能有效地抵御假冒服务提供者的攻击,而信任感知的容错方法在所有情况下都能达到高于 0.96 的成功率,有效地屏蔽了假冒服务提供者攻击。由此可看出,信任感知的容错方法无论是在容错服务数量还是抵御假冒服务提供者攻击等环境适应性方面都比一般容错方法具有更好的实用性和现实性。

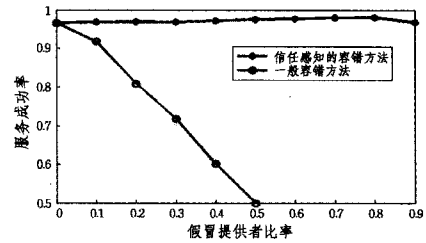

图 5 冒提供者比率与服务成功率之间的关系

在服务组合运行过程中,利用异构服务整合环境下高性能计算服务资源和网络服务资源的应用特点和失效规则,模拟云服务资源的动态调用情况。对于动态调用策略提供服务的可靠性方面,可以通过平均完成作业量、平均响应时间、平均执行时间来进行评价,完成作业量越多,作业平均执行时间越少,响应时间越短,说明服务提供策略的有效性越好。高性能计算服务资源和网络服务资源在负载特征、服务资源使用特征、服务性能指标和服务管理时间粒度等方面往往存在着较大差异。在实验过程中,网络服务资源和高性能计算服务资源的混合比例为 6∶1 和 12∶1(分别记为 *whc*6 和 *whc*12),失效分布 *zipf* 为 0.5,服务节点资源总数从 150 按步长逐渐增加到 1350 个。如图 6、图 7 所示为一般调用策略和基于失效规则的调用策略的性能比较,横坐标是服务节点资源总数,纵坐标分别是平均完成作业量和平均响应时间。

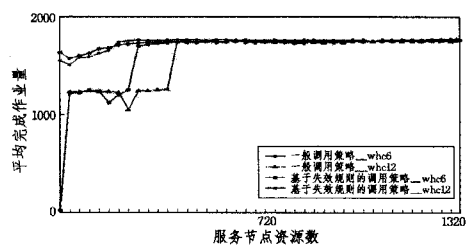

图 6 两种调用策略的平均完成作业量对比情况

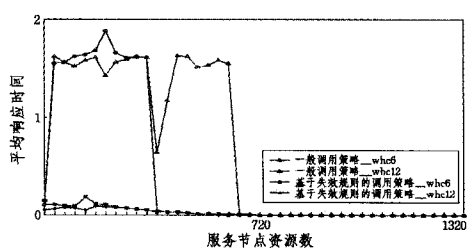

图 7 两种调用策略的平均响应时间对比情况

从图中可以看出,与一般服务调用策略相比,在服务失效规则非一致分布的情况下,基于失效规则的服务调用策略能够更好地优先选择高可靠的服务资源来执行服务系统负载,有效地保障了服务系统在高负载运行情况下完成的作业量、执行时间和服务响应时间,进而在很大程度上缓解了服务系统资源的瓶颈,有效保证了服务的运行性能。

结束语 本文针对云服务的冗余特性和可靠性保障需求,设计了基于信任冗余的云服务可靠性增强总体框架,为云服务运行的服务需求分析阶段、服务准备阶段、服务提供与实现阶段和服务运行维护阶段全生命周期的可靠性优化提供了方法支持;在服务准备时的冗余设计阶段,基于选举协议的云服务轮询检测机制设计的信任感知的容错服务选择算法,保证了满足服务可靠性的最小容错服务个数;基于服务组合运行时的容错处理框架提出的基于失效规则的云服务调用策

略，为服务的响应时间提供了有力的保障。

云服务通过动态协作的网络资源以一种柔性可演化、连续反应式、多目标适应的组合新形态实现了服务资源的有效动态配置和共享使用，为跨企业、跨组织的分布式应用系统的构建提供了支持。云服务网络环境的开放性、平台的异构性以及通信的异步性和服务构件的自治性等不确定因素使得云服务组合的可靠性受到了巨大影响，这使得云服务的可靠性优化必须考虑每个时刻组成云应用服务的实时可靠性状况。下一步的工作重点是基于现有的云计算服务应用平台，从不同角度考虑影响云服务可靠性的因素，并进行应用实例分析，进一步优化可靠性设计方法，真正达到为云服务用户提供持续可靠的运行服务的目的。

参 考 文 献

[1] Jayasinghe D. FAWS-a client transparent fault tolerance system for SOAP-based Web services[EB/OL]. http://www-128.ibm.com/developerworks/webservices/library/ws-faws/,2005

[2] Liang D, Fang C L, Chen C, et al. Fault-tolerant Web service [A]// Proceedings of the 10th Asia2 Pacific Software Engineering Conference(APSEC'03)[C]. ChiangMai, Thailand, 2003:310-319

[3] Santos G T, Lung L C, Montez C. FTWeb: a fault tolerant infrastructure for Web services[A]// The 2005 Ninth IEEE International EDOC Enterprise Computing Conference(EDOC'05)[C]. Enscheda, Netherlands, 2005:95-105

[4] Lin M, Chang M, Chen D. Distributed-program reliability analysis: complexity and efficient algorithms[J]. IEEE Transactions on Reliability, 1999, 48(1):87-95

[5] Yang M, Wang L N. Research of Web service reliability enhancement method based on trust fault tolerant[J]. Journal on Communications, 2010, 31(9):131-138

[6] Tian G H, Meng D, Zhan J F. Reliable Resource Provicion Policy for Cloud Computing[J]. Chinese Journal of Computers, 2010, 33(10):1859-1872

[7] Heath T, Martin R P, Nguyen T D. Improving cluster availability using workstation validation[A]// Proceedings of the ACM SIGMETRICS[C]. Marina Del Rey, California, USA, 2002:217-227

[8] Sahoo R K, Sivasubramaniam A, Squillante M S, et al. Failure data analysis of a large-scale heterogeneous server environment [A]// Proceedings of the DSN 2004[C]. Florence, Italy, 2004:772-784

[9] Shereshevsky M, Crowell J, Cukic B, et al. Software aging and multi-fractality of memory resources [A]// Proceedings of the 33rd DSN 2003[C]. San Francisco, Ca, USA, 2003:721-730

[10] Huang Y, Kintala C, Kolettis N, et al. Software rejuvenation: Analysis, module and applications[A]// Proceedings of the 25th Symposium on Fault Tolerant Computer Systems[C]. Pasadena, California, 1995:381-390

(上接第 123 页)

表 2 给出了 Coverity Prevent、Flawfinder 和本文中的方法测试漏洞的结果比较。从表 2 的测试结果来看，Coverity Prevent 的正确率较好，false_positives 的 3 个代码包的检测结果均保持在 96%左右，但存在一定的误报，Flawfinder 的结果略低。但 Encapsulation 类的漏洞的语法解析树的漏洞发现方法可以 100%发现漏洞，当然有一部分原因是在测试前已知此漏洞的存在，在契约规则的编写过程中会有一定的影响。由于特殊环境要求，例如代码包 B，能够发现在自定义传输数据包中发现的漏洞并归类到 Errors 类。由此可以得到 Coverity Prevent 和 Flawfinder 发现常规漏洞的效果明显较好，但在处理特殊环境和特殊参数情况下，类似自定义类和自定义传输协议数据包情况下，语法解析树的漏洞发现方法的契约规则更灵活，如果测试人员能够提前了解代码的功能，则对契约规则的设置帮助非常大。在实际工作中，对源代码漏洞检测可将不同的工具结合使用。另外虽然 Flawfinder 的漏洞发现要略低于 Coverity Prevent，但其由于属于开源软件，费用更低些。

结束语 针对源代码漏洞检测是一项相当复杂又繁琐的工作，需要程序员细心地了解每个调用函数或者对象方法的内部代码，利用改进的语法解析树解析函数调用前后形成的契约规则来判断函数是否存在漏洞，尤其改造了契约规则文法以支持对象之间存在继承而形成的契约规则。这种方法针对对象类继承关系下的漏洞检测和排查效果明显，并易于扩展；将其与已有的静态漏洞排查软件结合，以增加发现概率。

参 考 文 献

[1] Coverity[CP/OL]. http://www.coverity.com,2012

[2] CodeCheck[CP/OL]. http://www.abraxas-software.com/,2012

[3] FlawFinder HomePage[OL]. http://www.dwheeler.comlflawfinder/

[4] Bloch V J, Kohno J T T, McGraw G. ITS4: A Static Vulnerability Scanner for C and C++ Code[C]// Proc. 16th Computer Security Applications Conferences. New Orleans, LA, 2000:257-266

[5] Bauer T, Lips H P, Thiele G, et al. Operational tests on HVDC thyristor modules in a synthetic test circuit for the sylmar east restoration project[J]. IEEE Transactions on Power Delivery, 1997

[6] 张晓琳，王国仁. 用继承扩展 XML-RL[J]. 小型微型计算机系统，2005，26(2)：243-247

[7] 阳小奇，刘坚. 一种基于契约的跨过程安全分析方法[J]. 西安电子科技大学学报：自然科学版，2006，33(3)：390-394

[8] 陈海明，董韫美. 上下文无关语言分析树的一种表示形式[J]. 计算机研究与发展，2000，37(10)：1181-1184

[9] 陈再良，徐德智，陈学工，等. 基于链式结构 XML 文档的生成方法[J]. 计算机工程，2006，32(10)：59-61

[10] 肖袁. 一种高效的 XML 多分支路径查询算法[J]. 计算机应用与软件，2010，27(7)：153-155

[11] Swiler L P, Phillips C, Ellis D. Computer-attack Graph Generation Tool[C]// Proceedings of the 2nd DARPA Information Survivability Conference & Exposition. Los Alamitos, California, USA: IEEE Computer Society, 2001:307-321

[12] ISC. Internet Domain Survey[OL]. http://www.isc.org/ds/

[13] SCAP. Security Content Automation Protocol [OL]. http://scap.nist.gov/

[14] CVE. Common Vulnerabilities and Exposures. http://cve.mitre.org/

[15] CERT/CC. CERT/CC Statistics[OL]. http://www.cert.org/stats/

[16] 雷富兴，张来顺. 基于 Hoare 逻辑的过程调用的形式化方法[J]. 计算机工程与设计，2011，32(1)：197-201