Web服务容错技术研究

刘玲霞 武兆雪 钱 渊 夏靖波
(空军工程大学电讯工程学院 西安 710077)

摘 要 随着Web服务的迅猛发展,Web服务的高可用性已成为业界关注的一个新的焦点。作为一个新型的分布计算模型,Web服务有很多新的特点,所有这些新的特点对传统容错技术带来新的挑战。以如何提高Web服务的可用性为目标,对Web服务容错关键技术(包括体系结构、复制算法、失效检测以及日志和恢复)进行了详细分析。在此基础上针对Web服务的特点提出了需要解决的问题并给出解决思路。

关键词 容错,网络服务,可用性

Fault-tolerant Web Services

LIU Ling-xia WU Zhao-xue QIAN Yuan XIA Jing-bo
(The Telecommunication Engineering Institute, Air Force Engineering University, Xi'an 710077, China)

Abstract With the rapid development of Web services, the high usability of the Web service is becoming a new focus for research. As a new application model for decentralized computing, Web services have many new characteristics. All the characteristics bring the new challenges to the traditional fault-tolerant technology. With the aim of improving the usability of the Web service, this paper anatomized the key techniques of the fault tolerant Web services, including system architecture, replication algorithm, fault detection and log and recovery. On this foundation, it put forward the problems that need to be solved and gave the thinking to the characteristics of Web services.

Keywords Fault-tolerant, Web services, Usability

1 引言

Web服务是一个崭新的分布式计算模型,是一系列包括XML,SOAP[1],UDDI[2],WSDL和WSFL等标准在内的综合,给电子商务和B2B(business-to-business)等应用带来了无限商机。Web服务是开发下一代分布式应用的重要技术之一。由于在电子商务、应用集成、业务流程管理等领域有很好的应用前景,Web服务得到了工业界和学术界的广泛关注。近年来,许多研究机构、组织和公司纷纷投入到Web服务的研究和应用当中。工业界主要注重Web服务规范和协议的标准化问题,学术界针对Web服务核心支撑技术(包括服务组合、服务发现、服务安全等)展开了大量的研究工作[3]。目前,Web服务研究仍有许多关键问题尚待解决。其中,Web服务的可用性是非常重要的一个方面[4,5]。服务提供者需要考虑自己所提供服务的服务质量,以便确保它们在承诺的范围之内。

满足服务的可用性需求可以由不同层次的技术实现来支持:第一种是利用服务的服务实现所基于平台的容错机制来实现对服务可用性的支持,采用这种方式可以很好地保证服务的可用性,但并非所有服务实现所基于的平台都具备容错机制。服务的具体实现类型多样,若将可用性要求高的服务的具体实现都移植到具有容错机制的平台中,即为每种服务实现类型都提供相应的容错平台支持,代价是昂贵的,也是不可行的;第二种是通过服务发现等手段找寻可用服务来替换不可用的服务,保证用户能访问到可用的服务。用户一旦发现某个服务不可用,可以通过服务发现等手段快速查找和使用其它可用的Web服务。但这种方式并不能保证一定能为不可用的服务找到可替换的服务,此外某些服务是无法被替换的。因此采用这种方式只能在某种程度上保证用户能访问到可用服务,而不能从根本上保证服务的可用性;第三种是在Web服务平台中提供通用的容错机制来保证服务可用。作为Web服务在发布、部署、运行监控和管理等生命周期阶段的支撑平台,Web平台亦可如已有中间件平台一样,为Web服务应用提供容错机制,满足Web服务应用的可用性需求。

Web服务平台作为面向Web计算环境下的中间件平台,与传统的中间件平台一样,也需为Web服务应用提供可靠性、安全性、可伸缩性、高可用性、高效性等服务保证。而Internet的开放性和异构性使得大多数传统解决方案几乎不可重用。为此,Web服务的可靠性、安全、事务、可伸缩性、高可用性等诸多问题需要进一步加以研究[6]。目前已有一系列规范支持服务发现、事务、安全等技术,但现有技术还不支持如何保证服务的高可用性。消息可靠传输协议[7]仅保证消息能可靠到达服务,并没有为保证服务高可用提供技术解决方案。

作为一个新型的分布计算模型,Web服务有很多新的特点:Web服务的互操作性要求服务请求者无需感知访问的服务是否容错,容错服务与非容错服务能使用相同的客户端访

到稿日期:2008-02-29 本文受陕西省自然科学基金项目(No. 2007F01)及空军工程大学电讯工程学院博士启动基金(No. KDYBSJJ303)资助。
刘玲霞 博士,讲师,CCF会员,E-mail:lingxia-liu@tom.com;武兆雪 硕士,讲师;钱 渊 博士,讲师;夏靖波 博士,教授,博士生导师。

问;Web 服务支持多种后端应用;它面向的广域环境更对容错技术带来挑战。在广域环境中,服务应答慢等同于不可用,而且也无法保证可靠通讯条件。所有这些新的特点对传统容错技术带来新的挑战。

2 相关概念

2.1 Web 服务架构

Web 服务的基本架构如图 1 所示。

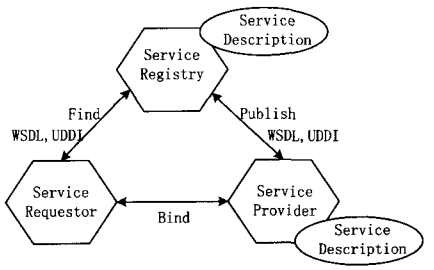

图 1 Web 服务的基本架构

该架构由 3 个角色(分别是服务提供者、服务注册中心和服务请求者)和 3 个基本操作(分别为发布、查找和绑定)组成。

服务提供者将其服务发布到服务注册中心。服务请求者通过查找操作从服务注册中心检索到所需服务的服务描述,接着使用服务描述与服务提供者进行绑定并访问 Web 服务。

在该架构中有一系列的标准(WSDL、UDDI)和协议(SOAP)。用 WSDL 来描述服务,用 UDDI 来发布查找服务,用 SOAP 来进行交互。在此需要说明的是,当服务提供者发布服务的时候,仅提供服务描述的 URL,服务请求者需要根据发布的 URL 到相关位置真正获得服务描述。

2.2 Web 服务的调用流程

Web 服务和客户之间、Web 服务之间都是通过 SOAP 消息交互。

Web 服务的一个简单的调用流程如图 2 所示。

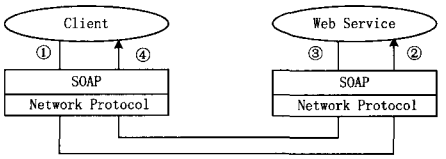

图 2 Web 服务的调用流程

(1)服务调用者发起一个请求,在该请求中将根据请求服务的服务描述提供所请求服务的地址、方法以及参数类型和值。调用端 SOAP 引擎将请求消息打包成 SOAP 消息后发送给所请求的 Web 服务。

(2)服务端 SOAP 引擎接收到请求的 SOAP 消息后,根据解析该消息后得到请求的方法、参数调用 Web 服务。

(3)服务端 SOAP 引擎将执行结果打包成 SOAP 消息后发送给服务调用者。

(4)调用端 SOAP 引擎接收到返回的 SOAP 消息后,解析该消息,得到返回结果。

2.3 可靠性与可用性

容错是提供可靠性和可用性的关键机制[8],使得系统在有失效发生时仍然能够继续正确执行。任何一个关键计算领域的应用开发者都可能面临容错问题,它是计算机科学中一个重要的研究领域。

容错的目的是满足应用关于可靠性、可用性等系统性能的要求。

• 可靠性[8]:系统在一个完整的时间间隔之内正确操作的概率。可以描述为一个时间函数 $R(t)$,它定义为系统在某个时间段$[t_0,t]$正确执行的条件概率(假设系统在 t_0 时刻正确执行)。系统无失效时间越长,可靠性越高。典型实例包括宇航飞行控制系统、军事系统以及某些工业控制器等;

• 可用性[8]:可以描述为一个时间函数 $A(t)$,它定义为系统在时刻 t 正确操作并且可以执行其功能的概率。可用性主要依赖于系统快速修复的能力。典型实例包括银行等时间共享系统和某些事务应用,例如飞机订票系统。

3 研究现状

SOA 是 Web 服务的基本架构。目前已有一些研究者开始分别从 SOA 架构中的 3 个角色出发对可用技术展开研究,如图 3 所示。

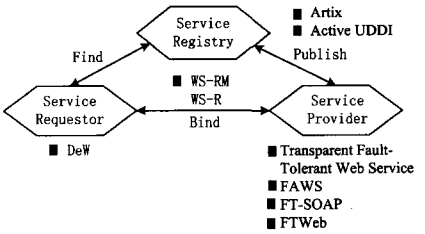

图 3 研究现状

• 可靠消息传递

可靠消息传送对 Web 服务而言是至关重要的。如果参与者不能确定消息交换的完成,很多业务问题就不可能得以解决。Web 服务中的可靠消息传递的目的在于允许应用程序间以简单、可靠、有效的方式发送和接收消息,甚至在应用程序、操作系统平台或网络连接失败的情况下亦能如此。

WS-ReliableMessaging (WS-RM)[9]和 WS-Reliability (WS-R)[10]这两个规范的目标都是通过定义一些机制来确保 Web 服务在不可靠的通信网络上确保消息的传递。它们的基本思想都是消息的客户端(或者发送方)连续地重新发送消息,直到它收到证实(或者确认)服务器(或者目的地)已经成功地获得了消息。消息可靠传递协议仅确保消息能在服务之间可靠传递,而无法保证服务可用。

• Artix

IONA 公司的 Web 服务产品 Artix[11]通过 Artix Locator 提供名字服务。服务的多个实例可以在 Artix Locator 中以相同的名字注册,Artix Locator 根据负载平衡算法从实例池中选择服务实例。它的主要目标是负载平衡,在某种程度上为用户提供了高可用服务。

• Active UDDI

在 Active UDDI[12]中,将 UDDI 视为一个 Web 服务,通过复制方式来保证 UDDI 服务可用。它扩展了 UDDI,通过代理(介于客户和 UDDI 注册中心)维护一个可用 UDDI 服务列表来提供可用的主动 UDDI 服务。

Active UDDI 为用户提供了一个可用的 UDDI 服务,使用户可以始终在注册中心进行发布查找操作。作为 SOA 架

构中服务提供者和服务请求者之间的桥梁，服务注册中心可为用户能访问到可用服务提供保证。

• DeW

DeW[13]设计的目标是保证用户能访问到可用服务（其服务实现部署在不同位置），它通过代理为用户屏蔽服务访问细节。代理分为宏（macro）代理和微（micro）代理两个层次，微代理负责应用相关细节，宏代理负责访问的异常处理。一个简单的访问流程如下：用户先下载服务的代理（代理和该服务的某个实现对应）；微代理向该服务的某个实现发起一个访问请求；如果该服务实现不可用，则由宏代理捕获到服务访问异常，并通过 DeW 提供的机制下载该服务其它实现的微代理，再由该微代理重新发起一个请求，直至该请求被成功处理。

通过其访问流程可以看出，DeW 框架虽然为用户屏蔽了服务访问细节，但请求者需要使用特定的方式来访问服务，这不符合 Web 服务松散耦合的特点。此外，该框架从服务请求者角度出发，仅仅关注如何将请求转发到可用的服务实现，没有在服务提供者端为服务提供失效检测（仅靠服务实现处理请求超时来判断其失效）及失效恢复等相应的容错机制。该框架仅保证用户能访问到可用服务，并未提供机制来保证服务可用。

• Transparent Fault-Tolerant Web Service

Transparent Fault-Tolerant Web Service[14]通过修改操作系统及 Web 服务器内核以提供对客户透明的服务访问机制。请求被接收后，首先转发给备份服务，由备份服务记录下该请求后，再转发给主服务。主服务处理完请求后将结果先传递给备份服务，再返回给客户。

它仅提供了消息转发机制来保证用户能透明地访问到可用服务，并未提供机制来保证服务可用。

• FAWS

FAWS[15]从服务提供者角度出发，在 Web 服务平台中提供了失效检测及日志等容错机制，为服务请求者提供透明的可用服务访问机制。接收到客户请求后，平台依次将请求转发给主副服务实现，直至该请求被处理。日志机制负责记录所有的请求消息，以此来保证该请求消息不会丢失。当请求服务的主服务实现不可用时，无需请求者再次发送请求消息，由平台转发该请求消息到其它备份服务实现，用户无需感知访问的服务的失效情况，访问过程对用户透明。此外，它还提供了失效检测机制，及时发现不可用的服务。

FAWS 从服务提供者角度出发，提供一些容错机制以提供透明的可用服务访问机制。它虽然提供了失效检测机制，并没有提供失效恢复机制，此外仅支持被动复制模式。与 DeW 类似，它仅保证用户能透明地访问到可用服务，并未提供机制来保证服务可用。

• FT-SOAP

FT-SOAP[16]是 National Taiwan Ocean University 研制的 Web 服务容错平台，基于服务模型，主要支持被动复制模式。它借鉴容错 CORBA[17]的思想，通过扩展 WSDL（引入服务组标签〈WSG/〉，记录服务组主副成员的版本、位置信息），引入服务组的概念来实现容错功能。用户首先访问主服务，如果主服务失效，则客户端 SOAP 引擎将根据 WSDL 中注册的服务组信息依次访问服务组中的成员，直到访问到可用服务为止。

从它的服务访问过程可以看出，它在有部分组成员失效尤其是主成员失效的情况下效率不高。服务访问者需要使用定制的客户端 SOAP 引擎，失效对客户端不透明，这破坏了 Web 服务松散耦合的特点。冗余副本的管理借鉴容错 CORBA 中的副本管理模式，没有考虑到服务实现的自治性。此外，失效检测机制通过简单的超时机制来判断失效。

• FTWeb

FTWeb[18]是由巴西的 Giuliana Teixeira Santos 等研制的 Web 服务容错平台，基于服务模型，主要支持主动复制模式。它同样借鉴容错 CORBA 的思想，通过扩展基本的 Web 服务运行环境来实现容错功能。它引入服务域[19]（service domain）的概念，将冗余的服务实现组织成单个服务。冗余的服务实现只被容错平台感知，对客户端透明。

但它使用客户端组件作为服务端的失效检测器，也破坏了 Web 服务的松散耦合的特点。与 FT-SOAP 类似，它也没有针对 Web 服务的特点研究容错管理技术，而是借鉴了容错 CORBA 中的管理技术。此外，失效检测机制亦通过简单的超时机制来判断失效。

• 协议栈的扩展

文献[20]通过扩展现有的 Web Services 协议栈，为关键服务的可用性提供技术支持。它分析了实际应用中的可用性需求后，对协议栈进行了 3 层扩展：Fault Detection，Enhanced Communication 及 High Assurance。Fault Detection 层主要提供状态检测功能，负责服务的状态检测，一旦其状态改变，它将通知所有对此事件“感兴趣”的所有组件。Enhanced Communication 层主要提供组通讯功能。High Assurance 层主要提供可用相关的服务，包括数据分发（Data dissemination）、监测和分布管理（Monitoring and distributed control）及事件通告（Event notification）。

随着 Web Services 的广泛应用，必将在协议栈层提供标准化协议，以增强服务可用性。

此外，还有一些工作着重针对如何为组合服务提供容错能力展开研究[21,22]。一个组合服务可以看作由基本服务（elementary service，文中的服务皆指基本服务）按照业务逻辑组成的工作流，保证容错的基本方式是有失效发生时采用补偿或替换来保证组合服务的正确执行。它们的区别在于：提供一个容错服务是从服务提供者的角度出发的（主要关注如何使自己提供的服务高可用），为组合服务提供容错能力是从用户的角度出发的（主要关注如何为用户提供一个高可用的复杂服务）；此外，作为组合服务的基本组成单位，有些关键基本服务在失效后是无法（或不能）补偿替换的。它们又是互补的：容错基本服务是容错组合服务的基础。一个组合服务如果是由很多不可用的服务组合而成，即使在理论上可以利用失效处理机制来保证组合服务仍然可用，其实这样的组合服务已经失去了实用性；服务组合时更倾向于选择一个可用性很高的基本服务。容错基本服务的使用会减少组合服务失效处理的时间和开销，提高组合服务的效率，为组合服务的可用性奠定基础。

4 关键技术

任何容错系统不外乎采用冗余手段，在有失效时能有机

制检测到并进行有效的恢复，以达到容错的目的。我们将对Web服务容错关键技术进行分析，提出需要解决的问题并给出解决思路。

4.1 体系结构

体系结构主要关注怎样将容错机制引入系统。目前主要有3种典型方式[23]，即系统法、服务法和反射法。这3种方法都有各自的优缺点。系统法将容错机制集成到底层的运行系统中，使得容错机制透明易用，但容错机制不易定制，并且也不易增加新的容错机制。服务法将基本的容错机制作为服务提供给用户，用户在此基础上定制自己的容错机制，其最大的优点就是灵活，用户可以利用系统提供的基本服务灵活定制自己的容错机制。但应用开发者需要在应用中正确地使用这些服务才能准确实现一个容错机制。反射法将应用的功能和非功能实现分开，可以根据需要将它们绑定，容错机制对应用开发者透明，而且可以灵活增加新的容错机制。它最大的缺点是性能。

而将3种典型的体系结构运用到Web服务框架中也有各自的缺陷。系统法将容错机制集成到底层的运行系统中，无法满足Web服务的后端服务类型的多样性等特点。服务法的优点是容错机制易定制，但它需要应用开发者在应用中正确地使用服务才能准确实现一个容错机制，使得应用中包含太多容错信息，从而无法动态配置某些容错属性，也无法全部满足Web服务的特点。反射法将应用的功能和非功能实现分开，使得应用中无需包含容错信息，从而可以动态配置容错属性。但目前基于反射法的体系结构几乎都是基于反射语言绑定的，无法适应Web服务框架中后端服务类型多样性等特点。

因此，需要设计一个新的体系结构（譬如结合反射法和服务法），一方面能适合Web服务的特点，另一方面又使得系统容错机制透明、易用、易定制，并且能最大限度地减轻应用开发者的负担。

4.2 复制算法

对分布式容错算法的研究由来已久，主要分成两种类型[24]：一种使用回滚恢复；一种使用模块冗余。

使用回滚恢复的算法可以分成两类[25]：基于检查点方式以及基于日志方式。检查点协议分为3种：无协作检查点（uncoordinated checkpointing）、协作式检查点（coordinated checkpointing）和消息引导检查点（communication-induced checkpointing）。根据记录日志事件信息的时机，消息日志可分为3种：悲观消息日志（Pessimistic Log）、乐观消息日志（Optimistic Log）及因果消息日志（Causal Log）。这种算法一般使用在多个服务协同的情况下。

模块冗余模式又可以分为两类[26]：模块多版本和模块复制。模块多版本可以从一定程度上避免软件错误，但其实现复杂，所以这种模式在分布容错系统中并没有得到广泛应用。模块复制是目前分布系统实现容错功能比较常用的模式。

主动复制[27]和被动复制[28]是两个经典的复制算法。主动复制算法中，所有冗余副本同时响应客户请求，当有副本失效时，其余副本在失效副本缺席的情况下继续正常工作。它失效恢复时间短，但耗费系统资源，同时要求成员的状态是确定的（deterministic），而且会带来重复嵌套呼叫问题。被动复制算法中，冗余副本分成主从两种，只有主副本接收和处理请求，主副本还负责更新从副本的状态。当主副本失效时，将从副本中的一个升级为主副本，继续提供服务。根据从副本（backup）的状态落后于主副本（primary）的程度，被动复制又分为冷备（cold backup）和温备（warm backup）等模式[29]。它节约系统资源，不要求服务的状态是否确定，但失效恢复时间长，而且还带来了主副本状态一致性问题。尤其是当主副本在处理请求的过程中突然失效，这时副本的状态一致性很难维护，处理不当会带来严重后果。

对于Web服务这种面向广域环境的分布计算，服务应答慢等同于不可用。它在不改变可用性的前提下对算法的性能提出了很高的要求。容错算法在很大程度上影响着服务的性能。然而目前的容错算法大多只是对主动或被动复制算法进行了某些方面的改进，很少致力于提高算法的性能。例如半主动复制算法（Semi Active）[30]解决了主动复制算法中状态确定性问题，异步主动算法（Asynchronous Active Replication）[31]使得可以在异步环境中使用主动复制算法。

经过分析发现，性能较优的主动复制算法中的结果协调（Consensus）过程对算法性能带来很大的影响，主要体现在：（1）结果协调过程必须等所有副本都处理完请求后才能进行，应答速度由处理速度最慢的副本决定；（2）在大部分副本没有失效的情况下需要一个至少是ES（Eventually Strong）级的失效检测器[32]的支持，而这样的失效检测器在分布计算环境尤其是异步环境下是很难实现的；（3）结果协调过程本身是耗费时间的。因此，改进Consensus过程能有效改善算法的性能。

4.3 失效检测

失效一般可以分为两类[33]，即Fail-stop和Byzantine（拜占庭）失效。在失效检测研究中，目前主要研究如何检测Fail-stop类型的失效。

容错系统一般采用超时机制实现失效检测，即通过被检测实体在规定时间内有无应答来判断失效，有轮询和心跳两种模式。

- 轮询（polling）：检测机制每隔一定的时间间隔检查被监控实体是否失效，相当于检测机制从实体“拉”失效事件，所以又称为“拉”模式。
- 心跳（heartbeat）：被监控实体定期向检测机制周期地发送心跳消息以表明自己的存活，相当于被监控实体向检测机制“推”失效事件，所以又称为“推”模式。著名的网格项目Globus就是利用心跳模式来检测运行进程的失效。

这两种监控模式各有优势。一方面，在pull模式中，失效检测参数（例如超时时间值，可能需要动态调整）只需驻留在失效检测器，无需分布在所有被监控实体上。但push模式中大量被监控对象所产生的心跳消息，很有可能突发性地阻塞网络。另一方面，在push模式中，只有one-way消息在发送，所以通信更加高效，而且如果存在若干个监控者同时监听心跳消息则使用组播设施（如IP组播）可以减少所产生消息的数目。总之，监测实体和被监控实体之间所采用的交互类型取决于应用本身，以及可容许的对失效信息的反应时间。

超时时限的设置对利用超时机制判断失效的失效检测器的性能有很大影响。时限过大，则使得失效检测器不能迅速发现失效；而时限过小，则使得失效检测器误将速度“慢”的被监测实体判断为失效。因此，简单地利用超时机制来判断一个实体是否失效会影响失效检测器的性能。

Tushar Deepak Chandra 等在 1996 年提出一个不可靠失效检测器(Unreliable failure detector)[32],它最初的目的是为了解决分布计算领域中的 agreement 问题。它的基本思想是:失效检测器维护一个队列,在一定时间内没有接收到某个被监控对象的消息就将它加入队列,接收到这个被监控对象的消息后就将它从队列中移除。它定义了一个失效检测器必须具有 2 个属性,即完备性(Completeness)和精确性(Accuracy),并且根据这两个属性的强弱将失效检测器划分为不同的级别。不同级别的失效检测器能解决不同层次的 agreement 问题。

虽然不可靠失效检测器最初的目的是为了解决分布计算领域中的 agreement 问题,但它实际上为失效检测器的设计提供了更好的思路。文献[34]基于不可靠检测检测器的思想和心跳模式,提出了自适应失效检测器(Adaptive failure detector)。自适应失效检测器利用不可靠检测检测器的思想,改进了传统的失效检测器。超时时限的自适应调整使得失效检测器检测失效的能力受超时时限的影响变小。Stelling 等基于不可靠检测检测器的思想为著名的网格项目 Globus toolkit 设计了一个失效检测服务[35]。

目前几乎所有的失效检测器的实现都基于一个假设,即被检测实体和失效检测器之间可靠通讯。这样简化了失效检测器的设计,但在 Web 服务面向的广域环境中,这样的假设无法得到保证。对于在没有可靠通讯假设的前提下,失效检测器在时限前没有接收到被检测实体的消息,意味着被检测实体失效、连接通道失效以及消息丢失。我们不能把这 3 种失效情况简单地判断为被检测实体失效,这样可能会增加系统开销或降低服务性能等。譬如对于采用被动复制的服务来说,当某个从副本发送给失效监测器的心跳消息丢失,若简单地认定此副本已经失效,则需要在一个新主机上启动一个新的副本,还需要对此副本进行状态恢复,在状态恢复的过程中可能还需要暂停主副本处理请求,这样会带来系统开销的无限增加和服务性能下降。因此,在广域环境中,需要一种能分辨这 3 种不同失效的失效检测器。

4.4 日志和恢复

在有失效发生时,容错系统需要恢复行为来消除失效对系统的影响。失效恢复是指即使在有失效发生时系统也能够通过重新配置维持可操作能力[36],例如在 primary 失效后让 backup 接管其工作。

所有的恢复都必须解决 3 个问题:(1) 实体的状态如何保存起来(即日志);(2) 如何从保存的状态恢复新的实体;(3) 如何有效地保证状态一致性。

消息日志是用于分布式系统中状态恢复的一种方法。消息日志法的基本思想是实体在失效之后退回到一个无错的检查点状态,在该状态下重新处理先前处理过的消息(称为重演),从而实现实体的状态恢复。

目前主要有 3 类消息日志,即悲观消息日志、乐观消息日志及因果消息日志。对于悲观消息日志,消息在处理之前必须保证已经被记录到稳定的存储器(stable storage)中。消息的记录和消息的处理是同步进行的,因此又被称为同步消息日志(synchronous log)。这种方式认为失效随时可能发生,所以称它为悲观日志。它必须在消息处理前先保存消息,因此它在系统即使无错时也会影响系统性能。对于乐观消息日志,消息先被缓存起来,再周期性地刷新到稳定的存储器(stable storage)中。消息的记录和消息的处理是异步进行的,因此又被称为异步消息日志(asynchronous log)。消息的处理和记录异步进行,因此这种方式对系统性能的影响不大,但这种方式增加了失效恢复的复杂性。因果消息日志是基于发送方的消息日志(一种乐观消息日志)的一种扩展,它综合了乐观和悲观消息日志的优点。

分析消息日志方式发现,目前缺乏对消息日志管理的研究。不管哪种方式都假定消息一定会保存在稳定的存储器中,恢复时可以直接从稳定的存储器中取得原先保存的消息来进行状态恢复。广域环境对消息日志的管理提出新的问题。如何保证各种消息一定会保存到稳定的存储器中(how)? 消息采用什么方式存储(集中式还是分布式抑或两者结合,where)才利于副本状态恢复? 系统中传递着各种 SOAP 消息,应该记录哪些消息以及具体记录些什么内容(what)? 因此我们需要针对 Web 服务的特点设计一个消息日志管理框架来解决这些问题。

结束语 随着 Web 服务的迅猛发展,Web 服务的高可用性已成为业界关注的一个新的焦点。作为一个新型的分布计算模型,Web 服务有很多新的特点,所有这些新的特点对传统容错技术带来新的挑战。

本文以如何提高 Web 服务的可用性为目标,重点针对 Web 服务的特点,对 Web 服务容错关键技术包括体系结构、复制算法、失效检测以及日志和恢复进行了仔细分析,提出需要解决的问题并给出解决思路。

当然,我们针对 Web 服务的特点所提出的需要研究的关键技术肯定不能完全覆盖 Web 服务容错技术研究的所有领域。将容错 Web 服务平台从理论研究推向实用还需要做大量研究工作。

参 考 文 献

[1] World Wide Web Consortium (W3C). Simple Object Access Protocol (SOAP) Version 1.2. 2003

[2] Organization for the Advancement of Structured Information Standards (OASIS). UDDI Version 3.0.1. 2003

[3] 岳昆,王晓玲,周傲英. Web 服务核心支撑技术:研究综述. 软件学报,2004,15(4):428-442

[4] W3C. Web service architecture requirements. 2004

[5] Leavitt N. Are Web services finally ready to deliver? IEEE Computer,2004,37(11):14-18

[6] Chaudhary A S, Saleem M A, Bukhari H Z. Web services in distributed applications advantages and problems. 2002

[7] OASIS. Web service reliable messaging (WS-Reliability 1.1) Specification. 2004

[8] Dhiraj K P. Fault-tolerant Computer System Design. New Jersey: Prentice Hall PTR, 1996

[9] WS-ReliableMessaging (WS-RM). http://www-128.ibm.com/developerworks/library/specification/ws-rm/, 2005

[10] OASIS. Web service reliable messaging (WS-Reliability 1.1) Specification. 2004

[11] IONA. Artix Technical Brief. http://www.iona.com/artix, 2004

[12] Jeckle M, Zengler B. Active UDDI-an extension to UDDI for dy-

(下转第 50 页)

COM. 2002

[3] Alarcon-Aquino V, Barria J A. Anomaly Detection in Communication Networks Using Wavelets. IEEE Proc-Commun, 2001, 148(6)

[4] Barford P, Kline J, Plonka D, et al. A Signal Analysis of Network Traffic Anomalies // Proc. of ACM SIGCOMM Internet Measurement Workshop. Marseilles, France, November 2002: 412-423

[5] Gao Jun, Hu Guangmin, Yao Xingmiao. Anomaly Detection of Network Traffic Based on Wavelet Packet // APCC'06. Asia-Pacific Conference on Communications. 2006

[6] Wenke L, Xiang D. Information-Theoretic Measures for Anomaly Detection // Proc. of IEEE Symposium on Security and Privacy. Oakland, CA, May 2001: 130-143

[7] Lakhina A, Crovella M, Diot C. Mining Anomalies Using Traffic Feature Distributions // Proc. of ACM SIGCOMM 2005. Philadelphia, Pennsylvania, USA, August 2005: 9-20

[8] http://www.apng.org/9thcamp/matbdfs.ppt

[9] 杨岳湘，王海龙，卢锡城. 基于信息熵的大规模网络流量异常分类. 计算机工程与科学，2007，29(2)：40-43

[10] Noble C C, Cook D J. Graph-based Anomaly Detection // SIGKDD '03. Washington, DC, USA, August 2003

[11] Han Jiawei, Kamber M. Data Mining-Concepts and Techniques. Morgan Kaufmann Publishers, 2000

[12] [EB/OL]. http://www.internet2.edu/network/

[13] Dainotti A, Pescape A, Ventre G. Wavelet-based Detection of DoS Attacks // Proceedings of IEEE GLOBECOM. 2006

（上接第 28 页）

namic and fault-Tolerant service invocation // The 2nd Annual International Workshop of the Working Group on Web and Databases of the German Informatics Society. Germany, 2002

[13] Alwagait E, Ghandeharizadeh S. DeW: A dependable Web services framework // The 14th International Workshop on Research Issues on Data Engineering. USA, 2004

[14] Aghdaie N, Tamir Y. Implementation and Evaluation of Transparent Fault-Tolerant Web Service with Kernel-level Support // The IEEE International Conference on Computer Communications and Networks. Miami, Florida, 2002

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

[16] Deron L, Fang C, Chen C, et al. Fault-tolerant Web service // The 10th Asia-Pacific Software Engineering Conference. Thailand, 2003

[17] Object Management Group. The Common Object Request Broker: Architecture and Specification, Chapter25: Fault Tolerant CORBA Specification. 2002

[18] Santos G T, Lung L C, Montez C. FTWeb: A Fault Tolerant Infrastructure for Web Services // The 2005 Ninth IEEE International EDOC Enterprise Computing Conference (EDOC'05). Enschede, the Netherlands, 2005

[19] Tan S, Vellanki V, Xing J, et al. Service Domains. IBM System Journal, 2004, 43(4): 734-755

[20] Birman K, van Renesse R, Vogels W. Adding High Availability and Autonomic Behavior to Web Services // The 26th International Conference on Software Engineering. Edinburgh, Scotland, United Kingdom, 2004

[21] 徐伟，金蓓弘，李京，等. 一种基于移动 Agent 的复合 Web 服务容错模型. 计算机学报，2005，28(4)：558-567

[22] Dialani V, Miles S, Moreau L, et al. Transparent fault tolerance for Web services based architecturesEighth International Europar Conference (EUROPAR '02). Germany, 2002

[23] Fabre J C, Perennou T. A metaobject architecture for fault-tolerant distributed systems: the FRIENDS approach. IEEE Transactions on Computers, 1998, 47(1): 78-95

[24] Beedubail G, Karmarkar A, Gurijala A, et al. An algorithm for supporting fault tolerant objects in distributed object-oriented operating systems // Fourth International Workshop on Object-oriented Operating Systems. Sweden, 1995

[25] Elnozahy M, Alvisi L, Wang Yi-Min, et al. A survey of rollback recovery protocols in message passing systems. ACM Computing Surveys, 2002, 33(3): 375-408

[26] Cristian F. Understanding fault - tolerant distributed systems. Communications of ACM, 1991, 34(2): 57-58

[27] Schneider F B. Implementing fault-tolerance services using the state machine approach. ACM Computing Surveys, 1990, 22(4): 299-320

[28] Budhiraja N, Maraullo K, Schneider F B, et al. Optimal primary-backup protocols // Sixth International Workshop on Distributed Algorithm. Haifa, Israel, 1992

[29] Chen I R, Bastani F B. Warm standby in hierarchically structured process-control programs. Trans. on Software Engineering, 1994, 20(8): 658-663

[30] Veríssimo P, Rodrigues L, Rufino J. Delta4: a generic architecture for dependable distributed computing. Powell D. ESPRIT Research Reports. Berlin: Springer, 1991: 295-305

[31] Baldoni R, Marchetti C, Piergiovanni S T. Asynchronous active replication in three-tier distributed systems // The 2002 Pacific Rim International Symposium on Dependable Computing. Japan, 2002

[32] Chandra T D, Toueg S. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, 1996, 43(2): 225-267

[33] Birman K P. Building secure and reliable network applications. Greenwich: Manning Publications, 1996

[34] Bertier M, Marin O, Sens P. Implementation and performance evaluation of an adaptable failure detector // IEEE Conference on Dependable Systems and Networks (DSN'02). Washington D. C., 2002

[35] Stelling P, Foster I, Kesselman C, et al. A Fault Detection Service for Wide Area Distributed Computations // The 7th IEEE Symp. on High Performance Distributed Computing. Chicago, USA, 1998

[36] Tanenbaum A S, Van Renesse R. Distributed operating systems. ACM Computing Surveys, 1985, 17(4): 419-470