Web 应用服务器的适应性失效检测*

陈宁江[1,2,3+], 魏 峻[1], 杨 波[1,2,3], 黄 涛[1,2]

[1](中国科学院 软件研究所 软件工程技术研究开发中心,北京 100080)

[2](中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

[3](中国科学院 研究生院,北京 100049)

Adaptive Failure Detection in Web Application Server

CHEN Ning-Jiang[1,2,3+], WEI Jun[1], YANG Bo[1,2,3], HUANG Tao[1,2]

[1](Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Key Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[3](Graduate School, The Chinese Academy of Sciences, Beijing 100049, China)

+ Corresponding author: Phn: +86-10-62630989 ext 214, E-mail: river@otcaix.iscas.ac.cn, http://www.ios.ac.cn

Received 2004-07-19; Accepted 2005-01-07

Chen NJ, Wei J, Yang B, Huang T. Adaptive failure detection in Web application server. *Journal of Software*, 2005,16(11):1929−1938. DOI: 10.1360/jos161929

Abstract: Failure detection provides the ability of timely detecting the liveliness of runtime systems and is the basic reliability technology in distributed systems. Providing good failure detection is important to Web application server (WAS) that is the leading middleware in Web computing environment. Adaptive failure detection requires that failure detectors can dynamically adjust the detecting quality according to the requirements of applications and runtime environments. In this paper, the concepts and qualities of failure detectors are firstly discussed, and a multi-level model of failure detection in WAS is presented. Based on the QoS (quality of service) specification of failure detectors, an algorithm for adaptive failure detection is given, and an adaptive failure detection framework of WAS is designed, which can satisfy the requirements of dynamically adjusting qualities and flexible integration of failure detectors. The work has been implemented in OnceAS application server, and the experimental results are given in the end.

Key words: Web application server; failure detection; quality of service; adaptability

摘 要: 失效检测是分布式系统的基本可靠性保障技术,它对运行时系统的存活状态进行及时检测.作为网络分布计算环境中的主流中间件,Web 应用服务器(Web application server,简称 WAS)需要提供良好的检测机制,并

* Supported by the National High-Tech Research and Development Plan of China under Grant Nos.2004AA112010, 2003AA413010 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.2002CB312005 (国家重点基础研究发展计划(973))

作者简介: 陈宁江(1975−),男,广西南宁人,博士生,讲师,主要研究领域为软件工程,中间件,网络分布式计算;魏峻(1970−),男,博士,副研究员,主要研究领域为软件工程,网络分布式计算;杨波(1979−),男,硕士,主要研究领域为网络分布式计算;黄涛(1965−),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件工程,网络分布式计算.

且要能满足适应性的需求.适应性失效检测要求失效检测器能够根据应用需求和系统环境的变化而动态地改变检测的质量.首先给出了 WAS 的多层失效检测模型,然后基于失效检测器的服务质量规约,提出了适应性失效检测算法,并设计了一个 WAS 的适应性失效检测框架.它能够满足动态调整失效检测质量和灵活集成失效检测器的要求.该工作在 OnceAS 应用服务器中进行了实现,并给出了 OnceAS 平台上的实验及数据.

关键词: Web 应用服务器;失效检测;服务质量;适应性

中图法分类号: TP393 文献标识码: A

失效检测(failure detection)是软件系统可靠性保障的基本技术,它对运行时系统的存活状态进行检测.提供失效检测功能的组件称为失效检测器[1](failure detector,简称 FD).在分布式系统中,失效检测器是失效恢复、动态重启、可靠性通信、集群管理等功能的基础.

Web 应用服务器(Web application server,简称 WAS)是目前 Web 环境下的主流中间件[2],是网络分布式系统的关键基础设施,可靠性可谓是其灵魂.在 WAS 中,需要提供良好的失效检测机制来及时检测服务器以及其中服务组件和应用组件的失效情况,从而提高系统的可靠性和可用性.随着应用需求与技术的发展,WAS 被要求具有良好的可配置性和适应性[2],能够提供面向不同运行环境和应用需求的多级别可靠性保障,这相应地要求 WAS 中的失效检测器能够支持这些需求.例如,在某一时段网络负载过大时,在不违反应用可靠性需求的条件下,能够动态地调整失效检测消息的发送周期,避免"伪失效"情况的出现.本文主要研究 WAS 中失效检测的体系结构和引入新的失效检测能力的机制;基于 FD 的 QoS 规约,为 WAS 提供具有适应性的失效检测能力.本文的工作基于中国科学院软件研究所研制的 OnceAS 应用服务器[3].它是遵循 SUN J2EE 规范实现的 Web 应用服务器.

本文第 1 节介绍分布式系统中失效检测器的基本概念.第 2 节给出 WAS 的多层失效检测模型.第 3 节阐述基于 QoS 规约的适应性失效检测算法,并提出一个 WAS 的适应性失效检测框架.第 4 节介绍在 OnceAS 平台上对本文工作所进行的实验.最后是相关工作介绍和本文工作的总结.

1 失效检测器的基本概念

在分布式系统中的组件集合$\Sigma=\{\sigma_1,\sigma_2,\ldots,\sigma_n\}$,各个$\sigma$之间存在消息传输的信道.$\sigma$中拥有失效检测器模块 FD.设 T 为时间集,一个 FD 可定义如下:

定义 **1**. 失效检测器函数 FD.设 $p\in\Sigma,t\in T$,函数 $FD_p(t):\Sigma\times T\to 2^{\Sigma}$.

Fig.1 State transitions of failure detection

图 1 失效检测状态转换

对于 $q\in\Sigma$, 如果 $q\in FD_p(t)$,则表示 p 的失效检测器在 t 时刻认为 q 失效.FD 的输出结果是被检测为失效的对象集合 $Failed=\bigcup_{t\in T} FD_p(t)$.对于一个被监控对象 q,存在两个基本状态:UP 表示 q 正常运行;DOWN 表示 q 发生失效.通常 FD 基于最终性(eventual)假设,即被怀疑的对象必将最终进入失效状态.但是,这样做没有考虑实际系统中的时间特性、伪失效等情况,因此,有必要引入 SUSPECTED 状态,表示 q 被怀疑失效但没有确认失效,即它有可能被误判;运行期间被怀疑对象的集合为 $Suspect$,且 $Failed\subseteq Suspect$.如图 1 所示,q 存在 4 种状态转换:UD 转换(UD_transition)、US 转换(US_transition)、SU 转换(SU_transition)和 SD 转换(SD_transition).SUSPECTED 属于中间状态,一般对 FD 的外部透明,是否执行 US 转换取决于系统的失效检测策略.分布式系统中的失效检测有两种基本策略[4],即 Push 策略和 Pull 策略,它们各有特点,适用于不同的应用需求和环境.

2 WAS 的多层失效检测模型

一个 WAS 的主要组成部分包括:为应用组件提供生命期管理和运行时环境的容器(container);为容器及应用提供运行时支持的各种服务组件;用于管理服务组件的管理内核.通常若干 WAS 可以构成集群(cluster),以提供可伸缩性、高可靠性和高性能的保障.WAS 在应用组件、服务组件、容器、服务器进程、机器节点、集群等多个层次上都可能发生失效.因此,我们采用面向多种粒度失效的层次型失效检测体系结构.采取层次型结构有如下好处:1) 一个层次的 FD 只针对该层次粒度的对象,便于采取和定制适合该粒度的检测算法;2) 获得较高的失效类型覆盖度,并使 WAS 的失效检测能力具有可扩展性;3) 由于失效处理可以局限于一定范围内,从而提高了检测的快速性;4) 提高可靠性,某一层次的 FD 失效不影响系统对其他对象的监控.在实际环境中,容器、服务组件和 WAS 进程的失效较为常见,常用的检测手段主要有 heartbeat,ping,事件/消息机制等.

WAS 的多层失效检测结构如图 2 所示,其中包括 3 个层次的 FD:监控 WAS 中容器和服务组件的服务级 FD(service level FD,简称 SLFD);监控 WAS 进程的进程级 FD(process level FD,简称 PLFD);监控集群及其成员的集群级 FD(cluster level FD,简称 CLFD).它们的定义如下:

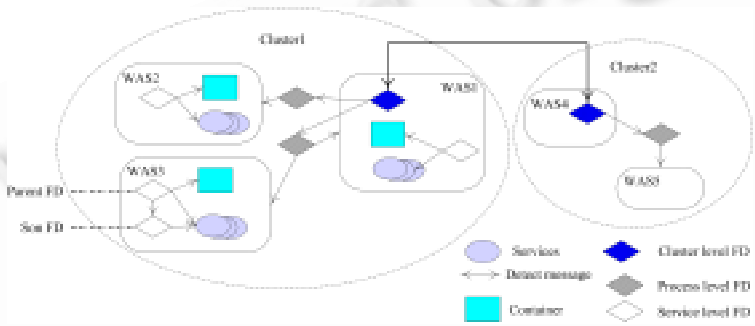

Fig.2 Multi layer failure detector architecture

图 2 多层失效检测体系结构

定义 **2**. $SLFD=\langle Id,AL,S\rangle$;$PLFD=\langle Id,AL,P,SFD\rangle$;$CLFD=\langle Id,CId,AL,PFD,CFD\rangle$.其中:(1) *Id* 是 FD 的唯一标识;(2) *AL* 是 FD 所使用的失效检测算法;(3) *S* 是 SLFD 所监测的服务集合;(4) *P* 是 PLFD 所监测的 WAS 进程集合;(5) *SFD* 是 PLFD 所监测的 SLFD 集合;(6) *CId* 是 CLFD 所处集群的标识;(7) *PFD* 是 CLFD 所监测的 PLFD 集合;(7) *CFD* 是一个集群的 CLFD 所监测的属于其他集群的 CLFD 的集合.

为了提高检测的可靠性和效率,同一层的 FD 之间可以构建父子关系.

定义 **3**. FD 的父子关系.设两个失效检测器 FD_1 和 FD_2,D_{FD_1} 和 D_{FD_2} 分别为它们所监测对象的集合,$D_{FD_2}\subseteq D_{FD_1}$ 且 $D_{FD_1}\not\subset D_{FD_2}$,如果 FD_2 将其检测结果传递给 FD_1,则称 FD_1 是 FD_2 的父亲,记为 $FD_1=P_a(FD_2)$.

定义 **4**. FD 检测深度.对于 $FD_1,FD_2,\ldots,FD_n$,如果它们按照父子关系构成一棵树,则 FD_i 在树中的高度称为它的检测深度,记为 $Dd(FD_i)$.

例如,在图 2 中,Cluster1 的 WAS3 中的两个服务级 FD 构成了父子关系,它们分别检测不同集合的对象,父 FD 接收子 FD 输出的检测结果.这样,既可以降低由于子 FD 失效所带来的不良影响,也可以缩小一个 FD 的监测范围,以提高检测效率.

3 WAS 的适应性失效检测机制

FD 的设计目的是能够在系统中长期运行以监控系统的状态,但实际系统必然由于环境的变化而存在着不稳定和稳定状态之间的变化.例如,随着 WAS 上应用负载的增加,在网络流量高峰期,检测消息可能会有较高的丢失率,这时,WAS 应该能够重新配置检测参数(即采取更高的检测延迟值),以避免因误判失效而影响应用系统的正常运行.所谓适应性的失效检测,即要求 FD 能够根据应用 QoS 需求和系统环境的变化而改变检测质量,动态地调整 FD 的运行时参数和行为.WAS 首先需要为各层次 FD 提供 QoS 描述机制,然后为应用组件提供透明

的服务质量调整机制,并能够方便地集成多种失效检测器的实现.

3.1 FD的QoS规约

对于 FD 的服务质量,人们已经进行了不少相关研究.Chandra 和 Toueg[1]研究了分布式系统中 FD 的性质和理论基础,指出 FD 具有两个主要性质:完整性(completeness)和准确性(accuracy).完整性反映了 FD 最终检测到确实发生组件失效的能力;准确性则是对 FD 允许发生的错误进行约束.在此基础上,Chen 等人[5]给出检测时间 T_D、错误重现时间 T_{MR}、错误持续期 T_M 等指标,其中 T_D 衡量 FD 的检测快速性,T_{MR} 和 T_M 衡量 FD 的准确性.这些研究成果为以后许多 FD 的设计和实现提供了有益的指导.考虑到 WAS 的实际需求,我们从两方面对 FD 的质量描述进行加强.(1) 时间性(timeliness),给出检测的时间约束,主要包括参数:发射周期 T_I,表示检测消息("心跳"和 Ping 消息)的发送时间间隔;怀疑等待延迟 T_W,表示从开始怀疑到确认失效所经历的时间.(2) 可伸缩性(scalability),表示满足系统规模可扩展和 FD 自身可靠度的要求,主要引入参数:检测深度 *Dd*;备份数目 *RN*,是一个 FD 将其输出结果备份的副本数目,用以避免单点失败.

FD 质量规约(FD QoS specification,简称 FDQS)给出应用对失效检测的需求描述,表示应用与 FD 之间的服务契约.它是 FD 实现适应性的基础.我们使用如下框架来描述 FD 的 QoS 规约.

定义 5. *FDQS*=⟨*Metric*,*Policy*⟩.其中,*Metric*={*Property*},*Property*={*Parameter*},*Policy*=⟨*Level*,*Actions*⟩.

Metric(度量)是衡量 FD 质量的属性类.一个 Metric 由若干 Property(属性)组成;Property 表示对某一度量的指标,每个 Property 又可由若干参数(parameter)来表示.策略(policy)指明 WAS 和 FD 基于服务质量的行为,它包括两部分:Level 为应用被授予的服务质量级别,它被映射为特定的 QoS 配置参数和操作;Actions 定义一组对应于 Level 的 WAS 和 FD 所要执行的行为.此规约能够满足现有的大多数 FD 的质量描述需求.我们能够方便地使用 XML 形式将其表达出来,这也有助于满足 WAS 的兼容性要求.图 3 给出了基于 XML 形式描述的 FDQS 的结构.

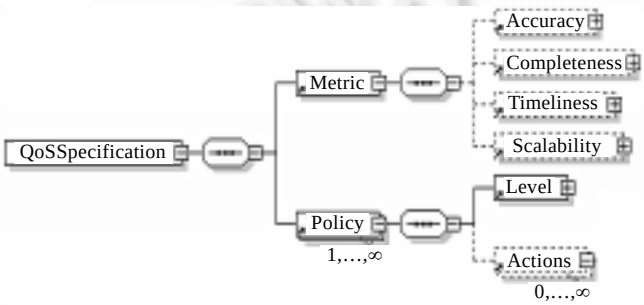

Fig.3 QoS specification of FD

图 3 FD 的 QoS 规约

3.2 基于QoS的适应性失效检测算法

基于 FD 的 QoS 规约,我们给出一个适应性失效检测算法.它的主要步骤如下:

Module FD;
Input: *FD_QoSSpecification*;
Output: *Failed*;
BEGIN
(1) **Initialization**:
Suspect← ∅ ; *Failed*← ∅ ;
For every parameter and policy in *FD_QoSSpecification*;
{Initialize the values, including T_D, T_I, T_W, *Dd*, *RN*, *TIMEOUT*, etc;}
(2) **Detect**:
at every T_I, send/receive detecting message m_d for $p \in \Sigma$;
t_p←*p*'s response time of m_d;
IF (policy is NOT_WAIT AND t_p>*TIMEOUT*)
execute UD_transition, *Failed*= *Failed*∪{*p*};
IF (policy is NEED_WAIT AND t_p > *TIMEOUT*)
{ execute US_transition, *Suspect*=*Suspect*∪{*p*};
IF (t_p>T_W) execute SD_transition, *Suspect*=*Suspect*−{*p*}, *Failed*= *Failed*∪{*p*};

ELSE execute SU_transition, *Suspect* = *Suspect*−{*p*};
}
IF (policy is NEED_BACKUP)
backup *Failed* to $P_a(p)$ and replicas of *p*;
calculate the value of QoS parameters such as T_M and T_{MR};
execute **probe**;
execute **negotiation**;
(3) **Probe**:
systemstatus←get runtime system information of WAS;
(4) **Negotiation**:
get *RuntimeQoSSpecification* from requests at runtime;
NEW_FD_QoSSpecification←*RuntimeQoSSpecification*∩*FD_QoSSpecification*;
Δ←ComputeMargin(*NEW_FD_QoSSpecification*, *systemstatus*); //Δ is the set of QoS margins
For every parameter $param_i$ in *FD_QoSSpecification*
{ $param_i$←$param_i$+ Δ_i;} //Δ_i is the margin of $param_i$
END.

在以上算法中:① detect是检测主过程.它根据FDQS以T_I为基本周期,分别执行各种失效检测状态转换(其中是否考虑 SUSPECTED 状态,由检测策略决定),进而如果需要备份,则执行对 *Failed* 集合的备份操作($O(Dd+RN)$次);在检测过程的末期通过计算和探测(probe)获取与检测质量相关的系统状态,然后调用协商(negotiation)过程.② probe 过程主要探测和获得运行时状态(如网络层延迟、内存资源等),生成系统状态集*systemstatus*.③ 协商过程的主要输入为运行时生成的 FDQS(*NEW_FD_QoSSpecification*)和 *systemstatus*,然后*ComputeMargin* 计算可行的 QoS 余度Δ,它表示在不违反 QoS 规约的条件下允许调整的 QoS 参数差值;协商的结果是形成新的 QoS 参数和策略.其中,协商是适应性调整的关键步骤,计算规则由 FDQS 中的 Policy 指定.

3.3 一个WAS的失效检测框架

基于以上适应性失效检测的原理,我们设计了一个适应性失效检测框架.它使WAS能够方便地集成各种失效检测器,并实施动态调整 FD 质量参数的行为.该框架的结构如图 4 所示.

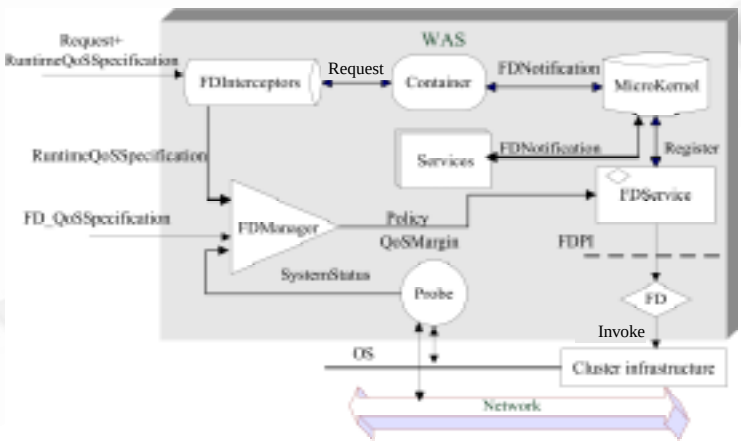

Fig.4 A framework of adaptive failure detection

图4 适应性失效检测框架

3.3.1 服务组件/容器失效检测

WAS 的服务组件和容器都可以使用基于微内核(microkernel)结构进行管理,如 Sun JMX[6],OnceAS MService[7]等都是微内核思想的体现.所有服务均通过统一的 Service 接口注册到微内核中,由后者进行运行时的服务管理、配置和调度.各个服务之间通过微内核进行通信,微内核基于通知模型(notification model)[6]提供了感知服务状态改变或者特定事件的机制.

为了支持服务级失效检测,我们引入新的失效通知类型 FDNotification,失效检测消息和结果都封装进入 FDNotification 对象,FDNotification 在微内核与组件之间传递.微内核注册为被监测 Service 的监听者(FDListerner),接收包含了失效检测结果或者“心跳”消息的 FDNotification;或者,微内核向各个 Service 广播 ping 消息(也为 FDNotification 类型),这时 Service 注册为该消息的监听者.需要指出的是,微内核中存在服务依赖关系管理(依赖关系是指一个服务的运行必须依赖于其他服务的存在),当一个服务发生失效时,微内核需要通知存在依赖关系的其他服务,以提供失效检测的完备性.

3.3.2 进程/集群级失效检测

对于同一节点的 WAS 实例进程,我们使用一个单独的伙伴进程对其实施检测.在集群环境中,集群中的每个 WAS 进程拥有集群级 FD,它们基于底层的集群基础设施(cluster infrastructure)实现对本集群中的成员进程和其他集群的检测.集群基础设施包含了一组协议,用于完成集群管理、通信、数据复制等任务,失效检测协议是集群基础设施中最基础的协议之一,其中实现了特定的失效检测算法.失效检测协议通过使用集群基础设施中的其他协议(如多播协议)来提供不同的检测质量,例如,对于强调可靠性的应用需求,失效检测协议通常会引入重复发送、事务性通信、收发校验、保证消息顺序等其他协议.这些失效检测所依赖的协议都可以通过 FDQS 规约中的 Policy 来定义.

3.3.3 适应层组件

根据前文的适应性检测算法的要求,FD 框架引入如下适应层组件:

· 失效检测拦截器(FDInterceptor)是基于 Interceptor 模式实现的组件,用于拦截用户请求消息.Interceptor 是支持适应性、可重配性和服务质量等应用的常见机制[8].FDInterceptor 主要负责两方面工作:一方面是取出请求消息中包含的失效检测策略及其参数,传递给失效检测管理器(FDManager)处理;另一方面是在返回结果中封装 FD 的输出结果,返回给客户.

· 资源探测器(Probe)用于实时监测操作系统和网络层的运行时资源和状态,它的输出(*SystemStatus*)包含运行时的网络延迟、内存资源、CPU 资源等信息,供 FD 的适应性算法使用.

· 失效检测管理器(FDManager)解析 FD 的 QoS 规约,转换成标准的 QoS 参数;根据输入(包括应用的可靠性规约、*SystemStatus* 等),动态确定失效检测的策略和 QoS 余度.

· FD 服务组件(FDService)是面向 FD 的特定服务组件,它通过 Service 接口注册到微内核中,充当具体的失效检测器实现与 WAS 之间的接口,从而将具体的失效检测器实现集成到 WAS 中.FDService 通过微内核的通知机制获取所需的 FD 执行结果,然后反馈给用户;接收 FDManager 的输出,据此调整失效检测器的配置和操作.

3.3.4 FDPI 协议

作为中间件,WAS 自身固然需要提供失效检测功能,但更为重要的是提供 FD 实现的集成和定制能力.遵循 Service Provider Interface 的思想[9],根据以上框架的内容,我们定义了 FD 提供者接口(failure detector provider interface,简称 FDPI)协议.FDPI 定义了 WAS 和 FD 提供者之间的契约,提供良好定义的 FD 类型和接口,使 FD 可替换、可定制和可扩展;提供了 FD 与 WAS 集成的接口和设施.FDPI 的结构,如图 5 所示,其中主要组件及其职责见表 1.

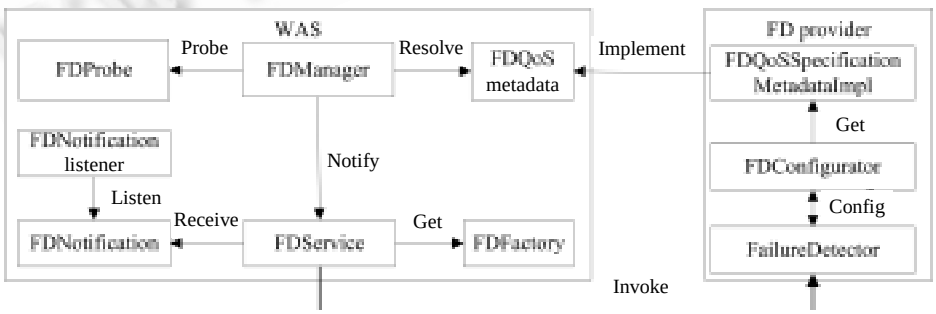

Fig.5 Structure of FDPI

图 5 FDPI 的结构

Table 1 The Components of FDPI

表 **1** FDPI 的主要组件

Component	Responsibility
FDManager	The abstract class of the FDManager component
FDFactory	Generating specific instance of FD
FDNotification	The interface of FDNotification
FDNotificationListener	The interface of listeners of FDNotification
FDService	The service interface of FD
FDProbe	The abstract class of the Probe component
FailureDetector	The interface for implementations of FD
FDConfigurator	The interface for configurations of FD
FDQoSMetadata	The interface of metadata of FDQS. FDQoSSpecificationMetadataImpl is the implementation at FD provider side

4 实现和测试

我们在 OnceAS 应用服务器中实现了上述适应性失效检测框架.为了评估适应性失效检测的效果,本节使用我们设计的一个简单的适应性 FD 算法 SimpleAdaptive 进行实验,它用于集群级失效检测之中.SimpleAdaptive 中的协商过程计算 FD 收到"心跳"消息的时间间隔,据此动态修改检测超时值 *Tr*(即 FD 经过 *Tr* 时间后,如果没有收到某个 WAS 的心跳消息,则认为后者发生失效).这样,当集群负载增加时,FD 能够随着网络延迟值的变化而延长失效确认时间,从而提高检测准确性.*Tr* 的计算方法如下:

Initialization:

set *Tr*, *T*max, *T*min, *Tr*, *K* from FDQS;

initialize *Tr*;

/* *Tr*′ is the interval between FD's receiving heartbeat message;

K is a constant which means the number of the most recent arriving messages */

Negotiation:

Tr′ ← the value returned by probe

$Avg(Tr)=\left(\sum_{i=1}^{K}Tr'\right)/K$; // average value of *Tr*′

IF ($Avg(Tr)>T\max$) $Tr=T\max$;

IF ($Avg(Tr)<T\min$) $Tr=T\min$;

$Tr=Avg(Tr)$.

实验设计为 5 台 PC 机组成一个集群,由一个成员作为主服务器,其他成员作为从服务器.从服务器上的 WAS 进程定期向主服务器的 WAS 进程发送"心跳",主服务器维护全局的可用服务器视图.这里假设主服务器不发生失效,如果它发生失效,则需要重构集群,这方面的工作不属于本文讨论的范围.在各节点上放置一个负载驱动程序,它们以一定概率向集群中发送一定数量的消息,用以模拟产生网络负载,并使负载发生变化.主服务器的 Probe 组件定期检测和计算网络延迟值 T_{delay},它反映了网络负载的变化情况.在实验中,每个节点的 WAS 进程分别崩溃一次,然后再重启,WAS 进程的失效时间在 90s~110s 之间,设定 *Tr*=10s.在负载驱动程序使用相同的负载产生参数的条件下,我们的实验分两种情况:(1) 使用 SimpleAdaptive 算法;(2) 不使用适应性算法.在非适应性情况中,*Tr* 值固定为 10s;在适应性情况中,动态地计算 *Tr* 值(取 *K*=10),为了避免颠簸,设置 *Tr* 的下限也为 10s.

此外,我们还对不同的怀疑等待时间 *Tw* 的情况进行了测试.对于每组测试,我们分别进行 10 次测试.图 6 和图 7 给出了其中两次实验的记录,其中,图 6(a)和图 7(a)反映了每个 WAS 进程实际失效情况和 FD 检测结果的比较,图 6(b)和图 7(b)给出了 T_{delay} 和 *Tr* 的变化情况.

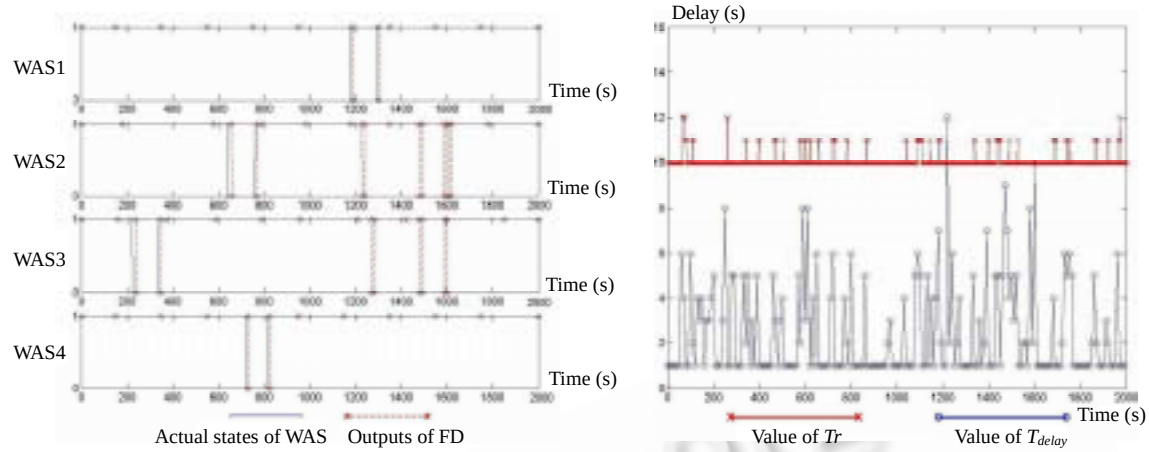

Fig.6 Results of one test of adaptive case

图 6 采用适应性机制的一次实验结果

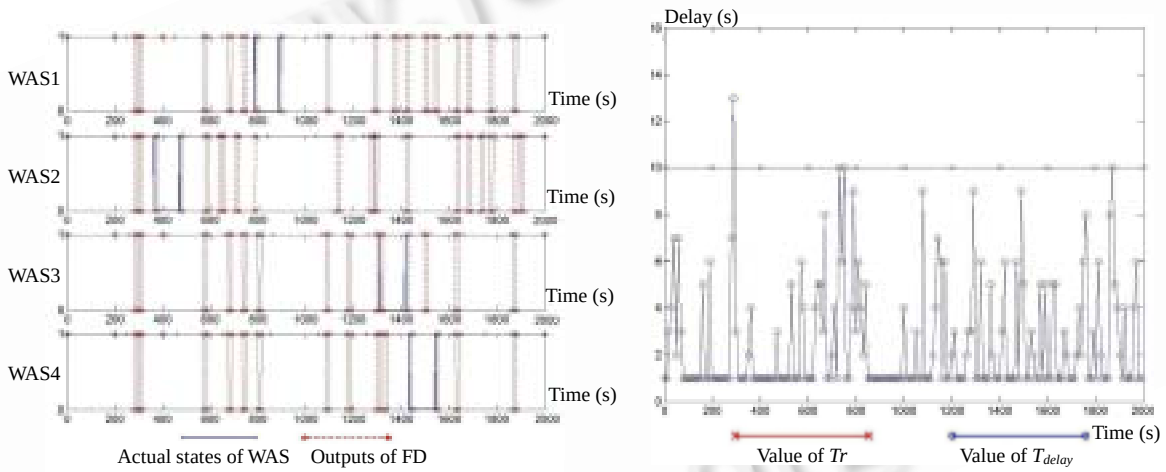

Fig.7 Results of one test of non-adaptive case

图 7 非适应性机制的一次实验结果

我们主要对两个 QoS 指标进行度量:一个是错误平均率λ_M[5],是指 FD 发生错误的比率;另一个是检测时间T_D.我们对每组测试的结果计算这些指标的平均值,见表 2.

Table 2 Experimental results of metrics of failure detection

表 2 失效检测质量指标值

Case	*Tw* (s)	Average errors	λ_M (%)	Average detection time (s)
Adaptive	3	10	1.38	13.98
	2	18	2.3	13.67
Non-Adaptive	3	41	6.66	10.21
	2	90	10.81	7.02

从实验结果可以看到,采用适应性算法能够较好地适应运行环境的变化,在检测错误数和错误率上比非适应性算法要好,达到了改善检测质量的目标.从检测的性能方面来说,在 T_w 分别为 3s 和 2s 的情况下,适应性情况的平均检测时间分别为非适应性情况的 1.37 倍和 1.95 倍,这是因为适应性处理带来了一定的延迟,所以检测快速性不够.但是,对于 WAS 这种长期运行的系统来说,由于 FD 错误所导致的服务器中断的代价比较高,所以λ_M比 T_D 显得更为重要.

5 相关工作

关于分布式系统中 FD 的研究,以往的 FD 实现大多假设分布式系统网络模型是弱不可靠性(weak unreliability)语义,它们忽略了网络分布式计算环境的实际因素.因此,不少研究针对不同的应用环境或者约束(如实时、大规模环境等),提出了多种 FD 实现[4,10−12].本文基于类似的时间性和可伸缩性考虑,提出了面向 WAS 的多层失效检测结构和适应性检测算法.在 FD 的 QoS 方面,除了 Chen 等人的具有代表性的工作以外,其他人也提出了一些 QoS 度量和参数,如文献[11,12].但是,它们并没有形成一个系统性的 QoS 规约.本文提出了一个 QoS 规约,能够基本上满足已有大多数 FD 实现的质量描述需求,有助于在 WAS 中集成第三方的 FD 实现.

在现有的主要 WAS 产品中,如 WebLogic[13],WebSphere[14],Oracle 9i/10gAS[15],JBoss[16],PKUAS[17]等,都提供了一定的失效检测机制,但检测粒度大多比较粗,主要面向节点级和进程级,缺乏足够的面向服务组件的检测能力.例如,WebLogic 利用 Ping 特定的端口或发送“心跳”信息来检测服务器是否可用;Orcale 应用服务器通过一个组件 OPMN 来完成服务器进程监控和重启,OPMN 提供了 ping、反向 ping 和进程 ID 等方法;JBoss 基于底层的 JavaGroups 组通信工具完成集群中的进程监控.这些 WAS 的 FD 检测的配置基本上都是静态的,并没有很好地集成第三方检测机制的能力.

其他一些中间件系统对可靠性保障以及失效检测能力进行了研究,如 Chameleon[18],2KQ+[19],StarBus[20]等,它们对 WAS 的适应性可靠性保障能力都具有很好的参考价值.但是,与本文的工作相比,它们没有给出一个良好的失效检测框架和方便集成 FD 实现的能力.

6 结束语

在目前 WAS 的研究和实现中,仍然缺乏面向中间件特征的可靠性理论和技术研究,在分布式系统等其他领域中的相关成果没有很好地应用到 WAS.本文针对 WAS 的特征,在分布式系统中失效检测器研究的基础上,对 WAS 的失效检测机制进行了研究.首先给出失效检测的 QoS 规约,为实现失效检测的动态适应性提供了基础;其次,提出了面向多粒度的多层失效检测结构以及基于 QoS 的适应性算法;然后,设计了一个 WAS 的适应性失效检测框架,实现有效的失效检测集成能力,使 WAS 的失效检测功能具有较好的可扩展性和可配置性,从而实现改善检测质量、保证应用的检测需求的目标.本文的工作在作者自主研发的 OnceAS 应用服务器中进行了实现.文中给出了在 OnceAS 平台上的实验.实验表明适应性检测框架是有效的.我们下一步的工作将主要研究在满足失效检测的 QoS 需求的条件下,如何与失效检测的执行代价之间进行权衡,进一步提高失效检测的性能.

References:

[1] Chandra TD, Toueg S. Unreliable failure detectors for reliable distributed systems. Journal of ACM, 1996,43(2):225−267.

[2] Fan GC, Zhong H, Huang T, Feng YL. A survey on web application servers. Journal of Software, 2003,14(10):1728−1739 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/1728.htm

[3] Fan GC. Research on some key technologies for web application server [Ph.D. Thesis]. Beijing: Institute of Software, The Chinese Academy of Sciences, 2004 (in Chinese with English abstract).

[4] Hayashibara N, Cherif A. Failure detectors for large-scale distributed systems. In: Kikuno T, ed. Proc. of the 21st IEEE Symp. on Reliable Distributed Systems (SRDS 2002). Washington: IEEE Computer Society, 2002. 404−409.

[5] Chen W, Toueg S, Aguilera MK. On the quality of service of failure detectors. IEEE Trans. on Computers, 2002,51(5):561−580.

[6] Sun Microsystems, Inc. Java management extensions instrumentation and Agent specification, v1.0. 2000.

[7] Fan GC, Lin SB, Dong WC, Hu JH, Feng YL. WebFrame: An extensible multi-layer web application server. Chinese Journal of Computers, 2004,27(4):451−460 (in Chinese with English abstract).

[8] Schmidt D, Stal M, Rohnert H, Buschmann F. Pattern-Oriented Software Architecture: Patterns for Concurrency and Distributed Objects, Volume 2. New York: John Wiley & Sons, 2000.

[9] Sun Microsystems, Inc. Java 2 platform enterprise edition specification (version 1.4). 2003.

[10] Fetzer C. Perfect failure detection in timed asynchronous systems. IEEE Trans. on Computers, 2003,52(2):99−112.

[11] Gupta I, Chandra TD, Goldszmidt GS. On scalable and efficient distributed failure detectors. In: Kshemkalyani A, Shavit N, eds. Proc.of the 20th Symp. on Principles of Distributed Computing (PODC 2001). New York: ACM Press, 2001. 170−179.

[12] Bretier M, Marin O, Sens P. Performance analysis of a hierarchical failure detector. In: Weinstock CB, ed. Proc. of the 2003 Int'l Conf. on Dependable Systems and Networks. Washington: IEEE Computer Society, 2003. 635−644.

[13] BEA Systems, Inc. Using WebLogic server clusters, version 8.1. 2003.

[14] Roehm B, Csepregi-Horvath B, Gao P, Hikade T, Holecy M, Hyland T, Rana R, Satoh N, Wang H. IBM WebSphere v5.1 performance, scalability, and high availability. 2003.

[15] Oracle. Oracle application server 10g high availability guide. 2004.

[16] Labourey S, Burke B. The JBoss Group. JBoss clustering. 2002.

[17] Huang G, Wang QX, Mei H, Yang FQ. Research on architecture-based reflective middleware. Journal of Software, 2003,14(11):1819−1826 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/1819.htm

[18] Bagchi S, Srinivasan B, Whisnant K, Kalbarczyk Z, Iyer RK. Hierarchical error detection in a software implemented fault tolerance (SIFT) environment. IEEE Trans. on Knowledge and Data Engineering, 2000,12(2):203−224.

[19] Wichadakul D, Nahrstedt K, Gu XH, Xu DY. $2K^{Q+}$: An integrated approach of QoS compilation and reconfigurable, component-based run-time middleware for the unified QoS management framework. In: Guerraoui R, ed. Middleware 2001. New York: Springer-Verlag, 2001. 373−394.

[20] Zhou MH, Wu QY, Jia Y, Wang HM. A management framework for replicated objects based on CORBA. Journal of Computer Research and Development, 2002,39(6):743−748 (in Chinese with English abstract).

附中文参考文献:

[2] 范国闯,钟华,黄涛,冯玉琳.Web 应用服务器研究综述.软件学报,2003,14(10):1728−1739. http://www.jos.org.cn/1000-9825/14/1728.htm

[3] 范国闯.Web 应用服务器关键技术研究[博士学位论文].北京:中国科学院软件研究所,2004.

[7] 范国闯,林世彪,董伟川,胡建华,冯玉琳.WebFrame:一种多层次可扩展的 Web 应用服务器.计算机学报,2004,27(4):451−460.

[17] 黄罡,王千祥,梅宏,杨芙清.基于软件体系结构的反射式中间件研究.软件学报,2003,14(11):1819−1826. http://www.jos.org.cn/1000-9825/14/1819.htm

[20] 周明辉,吴泉源,贾焰,王怀民.一个基于 CORBA 的复制对象管理框架.计算机研究与发展,2002,39(6):743−748.