

一种融合负载感知和检测点的 Web 服务适应性失效检测机制

陈宁江, 李昌武, 俞闽敏

(广西大学 计算机与电子信息学院, 广西 南宁 530004)

摘 要: 针对 Web 服务可靠性保障的需求, 探讨了在动态多变的网络环境下的 Web 服务适应性失效检测机制. 基于一个层次型服务失效检测框架, 融合了消息发送频率、丢包率等因素, 设计了一种可感知网络负载的、基于检测点的服务失效检测算法, 以增强对网络情境的适应性, 改善失效检测的服务质量. 模拟实验表明该算法能够有效保障在动态环境下的 Web 服务失效检测的准确性, 同时可取得较好的快速性权衡.

关键词: Web 服务; 失效检测; 适应性; 检测点; 负载感知

中图分类号: TP393

文献标识码: A

文章编号: 1000-7180(2010)08-0061-05

A Mechanism of Adaptive Failure Detection for Web Services Based on Combination of Load Awareness and Checkpoint

CHEN Ning-jiang, LI Chang-wu, YU Min-min

(College of Computer, Electronic, and Information, Guangxi University, Nanning 530004, China)

Abstract: To satisfy the requirements of reliable Web services, the mechanism of adaptive failure detection for Web Services in the dynamical network environment is studied. A framework of hierarchical failure detection is presented firstly, then an algorithm based on load awareness and checkpoint is designed. It considers the factors including the frequency of messaging delivering and packet loss so as to enhance the adaptability to the network context and to improve the quality of detection. Simulation results show that the algorithm can guarantee the correctness of detection as well as obtain the trade-off of rapidness.

Key words: Web services; failure detection; adaptability; checkpoint; load awareness

1 引言

随着 Web 服务应用的发展, Web 服务可靠性^[1]的提高成为人们关注的焦点. 可靠性保障涉及错误的检测、预测、容忍和预防等多方面, 而失效检测^[2] (Failure detection) 是其中的基础机制. 为了提高失效检测的准确性和精度, 近年研究者们较多地关注分布式系统的适应性失效检测问题, 提出了多种实现算法^[3-5], 它们的检测粒度主要面向节点级和进程级. 而 Web 服务是一种松耦合的软件系统, 由分布于网络环境中的若干服务复合而成, 有必要进一

步监测和判断服务级运行时状态. 文献[2]对 Web 服务的适应性失效检测机制进行了初步探讨, 能基于网络延迟动态调整检测相关的时间参数, 但是考虑的参数还比较简单. 本文在已有工作基础上, 进一步探讨了在层次型 Web 服务失效检测体系中的适应性失效检测算法, 考虑了更为丰富的影响因素 (如安全边界值、丢包率等), 使失效检测器的运行时参数能够根据网络情境的变化而动态调整, 进一步改善失效检测的服务质量.

收稿日期: 2010-06-09; 修回日期: 2010-07-09

基金项目: 国家“八六三”计划项目(2007AA01Z134); 国家科技支撑计划课题(2009BAH53B03); 广西高校人才小高地建设创新团队计划(桂教人[2007]71号)

2 层次型适应性 Web 服务失效检测框架

本文的工作基础是一个层次型服务失效检测框架,如图 1 所示.它主要的特点是可检测多粒度的对象状态,能够动态调整失效检测器的参数和行为.

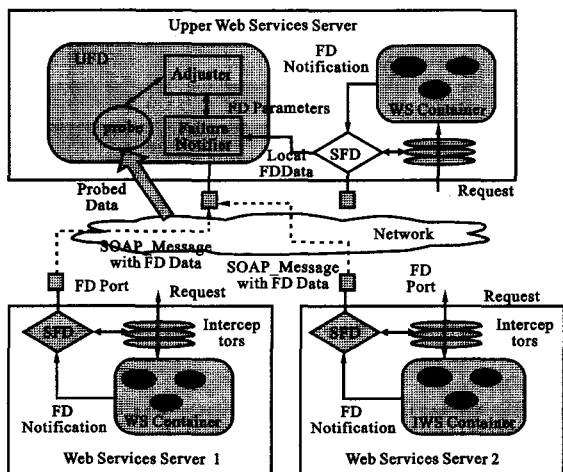


图 1 层次型适应性失效检测框架

在框架中,失效检测器分为服务级失效检测器 (Service Failure Detector, SFD) 和高层失效检测器 (Upper Failure Detector, UFD), 它们构成层次型结构 SFD 部署于 Web 服务节点上,负责通过与 Web 服务容器 (WS container) 的消息交互 (FD_Notification) 检测其中的 Web 服务实例的状态. UFD 一方面可以检测远程主机级的失效,另一方面收集远程主机上多个下层 SFD 的检测结果,获取相关联服务的状态.

为了避免消息被突发性阻塞和考虑到 Web 服务运行的特点,本框架采用的是 Pull 模式的检测消息发送模式. 执行失效检测的基本过程为: UFD 以一定时间间隔向已注册的 SFD 发送检测用 SOAP 消息,专用的 FD 端口 (FD Port) 负责接收并传给 SFD; SFD 收到检测消息后给 UFD 发送应答消息,应答信息包括 SFD 探测到的所有已注册的服务的最新存活状态. UFD 中的失效通告器模块 (Failure Notifier, FN) 通过 FD Port 汇集下层 SFD 结果并负责发布失效检测结果. 运行时探测器 (Probe) 用于实时监控网络层运行时的延时和丢包等状态. 判定器 (Adjuster) 根据探测器发来的运行时情境数据,执行调整计算来修改失效检测器的心跳消息发送频率、等待时间等检测参数值,从而动态影响失效检测行为.

针对失效检测消息在网络传递过程中可能存在的延时、丢失等问题,我们着重设计了适应性服务失

效检测机制,主要关注涉及网络环境中失效检测消息的问题. Web 服务往往需要中长期运行,服务运行的网络环境容易多变,特别是服务应用负载的增加,若网络负载处在高峰期时,心跳消息可能出现延时,丢包率也可能会成倍增加. 所以 UFD 应该能感知系统环境的变化而调整参数,以减少以 SFD 为中介对受监控服务状态的误判. 因此,我们为框架设计了一个改进的基于检测点和负载感知的适应性失效检测算法 CLAFD (Checkpoint-based and Load-Awared Failure Detection algorithm), 如下一小节所述.

3 基于检测点和负载感知的失效检测算法

因为传统的失效检测算法存在过早超时等缺陷,所以 Chen 等人^[6]等人提出了基于检测点的失效检测算法,其思想是将心跳报文发送时间转换为检测点. 本文的 CLAFD 算法在基于检测点的基础上进行改进. 因为检测点算法的失效判断不依赖于先于当前消息的消息和不依赖于个别消息的高延时或丢失,所以其对 Web 服务动态多变的网络环境具有适用性. 在 CLAFD 算法中,除了能够感知常见的网络延迟等时间因素外,我们增强了对丢包率的考虑,主要原因在于:其一,对于金融、政府、电信等部门关键业务需求,Web 服务的业务持续性问题需要得到重视,对消息丢失率有一定要求,如果丢包达到特定阈值,则可认为服务失效;其二,丢包率过高,检测消息重发的概率就会加大,加大网络的负担;第三,丢包率过高也会影响收集状态信息的准确性. CLAFD 算法的主要思想是:根据探测器 (Probe) 获得的网络情境数据,包括网络的延时和丢包率等状态,调整失效检测器的超时值 ξ 和消息发送的时间间隔 η (即发送频率). 调整超时值 ξ 的目的是针对失效检测的准确性,而调整消息发送频率 η 是因为 Web 服务运行环境如果处于良好的运行状态,意味着当前的可靠度比较高,出于性能考虑可适当调整检测强度.

在一般的分布式系统中,相邻的两条心跳消息延时变化不大,因此可以用心跳消息传输平均时间作为 ξ 值.

定义 1 心跳消息传输平均时间. $Avg(A_k) = \left(\sum_{i=1}^k p_i \right) / k$, 其中 k 为在给定周期内检测消息发送的个数, p_i 为第 i 个检测消息传递所用的时间.

定义2 丢包率 P_L . $P_L = \sum_{i=1}^k L_i / \sum_{i=1}^k R_i$, 其中, R_i 为给定的周期内已发出的 k 个检测消息中的第 i 个消息已发出, 并记 $R_i = 1$; L_i 为已发出的 k 个数消息中的第 i 个消息已丢失, 并记 $L_i = 1$.

但考虑到 Web 服务运行环境的复杂性, 相邻两条消息的延时可能会成倍的变化, 为了提高失效检测的准确性, 我们必须把这种消息激变的情况考虑在内. 我们引入了安全边界值 α 的计算^[7], α 是对估计下一条消息到达时间所产生误差的修正. 设 $error_k$ 表示第 k 条心跳消息的误差. $delay$ 表示预计的消息延迟, var 表示表示误差之间的变化幅度; γ 、 β 、 ϕ 为 3 个典型的参数, 其中 γ 表示与前面消息到达时间相比, 估计下一条消息到达时间的重要程度; β 和 ϕ 用于衡量消息到达时间的差异. 下面公式中的 k 均表示收到的第 k 条消息, $k+1$ 表示预计收到的下一条消息, 则消息错误值 $error_k$ 表示收到第 k 条消息实际时间值与预期值之差, 其计算如下式:

$$error_k = p_k - \text{Avg}(A_k) - delay_k \quad (1)$$

式中, p_k 为消息实际的传递时间; $\text{Avg}(A_k)$ 为心跳消息传输平均时间; $delay_k$ 表示消息延迟.

在上式的基础上, 可动态计算消息延时的值, 根据第 k 个消息 $error_k$ 计算出和 $delay_k$ 算出下一个消息延时值, 如公式(2)所示:

$$delay_{k+1} = delay_k + \gamma \cdot error_k, \quad (2)$$

下一个消息误差的变化幅度 var_{k+1} 可根据前一个 var 值和 $error_k$ 计算得到, 如下式:

$$var_{k+1} = var_k + \gamma \cdot (|error_k| - var_k) \quad (3)$$

安全边界值 $\alpha_{(k+1)}$ 表示根据 $delay_{k+1}$ 和 var_{k+1} 的值计算出下一条消息到达时间所产生误差的修正值, 其计算如下式:

$$\alpha_{(k+1)} = \beta \cdot delay_{k+1} + \phi \cdot var_{k+1} \quad (4)$$

由以上可动态计算失效检测超时值如下:

$$\xi = \text{Avg}(A_k) + \alpha_{(k+1)}, \quad (5)$$

综合以上丢包率、消息延时、失效检测超时等计算方法, CLAFD 算法的主要步骤描述如下:

Algorithm CLAFD ;

BEGIN

(1)初始化:

/* 设置初始的最大检测时间 $TD_{\max}$ 、消息发送频率的最大值 $\xi_{\max}$ 、允许的最大丢包率 P_L^U 、超时值 ξ , 和消息发送的时间间隔 η */

Initialize $TD_{\max}$, $\xi_{\max}$, P_L^U , ξ , η ;

/* 对受测 SFD 集合 S 中的每个对象初始化为怀疑状

态 */

For each $s \in S$, $s.state = \text{SUSPECTED}$;

new thread $_p$; // 创建网络状态探测线程;

new thread $_f$; // 创建服务失效检测线程;

(2) // thread $_p$ 线程计算平均延时、安全边界值和丢包率;

(2.1) /* UFD 以 η_i 为时间间隔, 给 S 中的 SFD 发送 (第 i 条) 检测消息 m_i ; */

UFD.send(m_i , η_i , S);

(2.2) /* 当接收到 m_i 时计算其传输的时间 p_i , 其中 q_i 表示接收到 m_i 时的时间, σ_i 表示 m_i 发出的时间; */

if (received(m_i) == ture)

then { $p_i = q_i - \sigma_i$; }

(2.3) $\xi = (\sum_{i=1}^k p_i) / k + \alpha_{(k+1)}$; // 计算超时值;

(2.4) $\alpha_{(k+1)} = \beta \cdot delay_{k+1} + \phi \cdot var_{k+1}$; // 计算 α 值

(2.5) $P_L = \sum_{i=1}^k L_i / \sum_{i=1}^k R_i$; // 计算丢包率

(3) // thread $_p$ 线程计算 ξ 、 η 的值

(3.1) get ξ ; // 获取消息超时值 ξ

(3.2) $\eta = \xi_{\max} - \xi$; // 计算消息发送时间间隔

(3.3) 重复以上 2 步;

(4) // thread $_f$ 线程执行失效检测过程

// 如果 SFD 未失效则进行失效判断

while (SFD! = FAILED) {

(4.1) // 构造检测点

/* 在 t 时间内未失效, 则获取最新 ξ 值, 重新构造检测点, 其中 T_{now} 为当前时间, t 为检测点, $s.state = \text{TRUSTED}$ 表示信任 SFD */

if ($T_{\text{now}} > t$ and $s.state = \text{TRUSTED}$)

then { get ξ ; $\tau_i = \sigma_i + \xi$;

$\tau_{i+1} = \sigma_{i+1} + \xi$; $t \in [\tau_i, \tau_{i+1})$; }

(4.2) // 超时失效判断

/* 在检测点 t 中, 如果没有从 $s \in S$ 收到 m_i 或者比 m_i 更新的消息, 则 s 状态设怀疑状态 */

if (notreceived(s , t , m_i) == ture) and $j \geq i$)

then $s.state = \text{SUSPECTED}$; ;

/* 如果收到 m_i 或者比 m_i 更新的消息, 则在 t 时间内信任 s */

if ($j \geq i$ and received(s , t , m_i) == ture)

then $s.state = \text{TRUSTED}$;

/* 如果 $T_{\text{now}} \geq t$ 且处在怀疑状态, 则判断此主机失效 */

if ($T_{\text{now}} \geq t$ and $s.state = \text{SUSPECTED}$);

then $s.state = \text{FAILED}$;

(4.3) if ($P_L^U >$) then $s.state = \text{FAILED}$;

// 若丢包率超过阈值, 则要求判断为失效

}

END.

在以上算法中, 线程 thread_p 用于实现了图 1 框架中 probe 模块的主要功能, 即实时监控网络层运行时的延时和丢包等状态; 线程 thread_f 实现了判定器模块的主要功能. CLAFD 算法中的 (1) 是对等待时间和丢包率允许的最大值等关键参数进行初始化; 算法中的 (3) 主要是根据 (2) 中的探测和计算结果算出下一个检测点的超时值、丢包率和消息发送的时间间隔 η , 其中 $\eta = \xi_{\max} - \xi$ 的理论依据是: ξ 变小说明服务运行环境正常, 所以可靠性比较高, 出于性能的考虑可适当降低检测强度, 即发送心跳消息的时间间隔 η 应该加大. (4) 主要是进行失效判断, (4.1) 是受测对象在 t 时间内未失效, 则需要构造新的检测点以便整个检测往下延续; (4.2) 是根据检测点算法进行失效判断, (4.3) 判断丢包率是否大于应用对丢包率的 QoS 要求, 如果超过值则认为发生失效. 本算法根据实时监测的数据所反映的网络延迟, 动态调整 ξ 值, 在计算 ξ 值时还特别考虑了由于 Web 服务运行环境的复杂性而产生消息激变的情形, 使失效检测对网络环境的负载变化具有一定的适应能力, 特别是避免由于网络忙而导致误判失效.

4 实验模拟

我们对 CLAFD 检测算法进行了实验模拟验证. 在实验中, 我们用 JBoss4.2.1 应用服务器作为 Web 服务的运行模拟平台, 模拟分布于网络中多台 PC 的一组 Web 服务实例 (规模为 50). 服务失效检测器模块作为可动态插拔的 JMX 服务插入 JBoss 服务器, 通过监测 Web 服务的消息收发进行失效检测; UFD 部署于一个独立的节点, 定期向部署于其它节点的服务失效检测器发送心跳消息和收集检测结果. 我们使用给定的延时程序来模拟环境负载的增减变化; 采用停止服务的方式模拟失效. 在此平台上我们的实验分别测试三种情形: (1) 无动态调整的普通失效检测算法 (标记为 non-Ada); (2) 以往我们只感知平均网络延迟的适应性失效检测算法^[2] (标记为 Avg); (3) 本文的 CLAFD 算法. 有关的初始检测参数设置如下:

对于 non-Ada 算法: $\xi_{\text{non-Ada}} = 2\ 250\ \text{ms}$, $\eta_{\text{non-Ada}} = 4\ 000\ \text{ms}$;

对于 Avg 和 CLAFD 算法: ξ 的初始值为 $2\ 250\ \text{ms}$, η 的初始值为 $4\ 000\ \text{ms}$, $P_L^U = 2.3\%$.

实验的统计周期为 30 分钟, 对以上三种算法分别进行实验, 对它们在运行期间的超时值 ξ 和消息

传递时间 T_w 的变化进行统计. 其中 T_w 指的是 UFD 给 SFD 发送检测消息再返回给 UFD 的整个过程所用的时间, T_w 受制于网络延迟, 并和网络延时同方向变化, 统计 T_w 的目的是为了根据网络延时的变化, 动态的改变超时值 ξ . 实验比较结果表明 CLAFD 算法能即时感知最新网络的情况, 即 ξ 能快速反应 T_w 的变化并即时调整其值, 所以 CLAFD 算法有助于提高失效检测的准确性.

我们在实验中还对失效检测的两个重要质量指标进行度量: 一个是错误平均率 λ_M , 是指失效检测器发生错误判断的比率; 另一个是检测时间 T_D , 指从受测对象发生失效到失效检测器检测到其失效所经历的时间.

试验结果表明在服务运行环境良好的情况下 non-Ada、Avg 和 CLAFD 这三种算法的错误平均率并没有什么区别, 但在高负载的运行环境中 CLAFD 算法减少了错误平均率, 从而提高了失效检测的准确性. 满足网络环境的复杂动态而多变, 服务执行消耗的系统资源较高的要求. 另一方面, CLAFD 算法由于采用了适应性策略而带来了额外的延时, 所以在高负载网络环境下平均检测时间明显增加. 但是, 在服务长期运行中, 服务处在高负载的时间不会持续很长, 所以服务在大部分时间里都是处在良好的网络环境, 从图 4 中可以看出网络在良好情况下 CLAFD 检测时间比较接近 Non-Adaptive 的检测时间, 在快速性方面比 Avg 算法要好, 从而在准确性和快速性方面能够取得较好的权衡.

5 结束语

Web 服务的失效检测是其可靠性保障的关键机制之一. 本文在分布式系统的失效检测工作基础上, 针对 Web 服务的运行情境特点, 考虑了网络环境的动态多变性, 对面向 Web 服务的适应性失效检测机制进行了研究. 首先, 本文给出了一个层次型适应性 Web 服务失效检测框架, 然后, 重点对其中的适应性检测算法开展工作, 通过引入丢包率、心跳消息发送间隔安全边界等因素, 提出了一种可感知网络负载的、基于检测点改进的 Web 服务失效检测算法, 使之对复杂的网络环境具有适应能力. 模拟实验结果表明, 该算法能够保障网络环境下失效检测的准确性, 有助于改善失效检测质量. 下一步工作将深入研究面向服务复合的失效检测机制、进一步丰富可感知的网络情境以及基于运行时探测信息的失效检测器灵活部署等问题, 以更好地满足检测的服务

质量需求.

参考文献:

- [1] Neal Leavitt. Are web services finally ready to deliver[J]. IEEE Computer, 2004, 37(11):14-18.
- [2] 陈宁江. 面向Web服务的层次型动态失效检测体系[J]. 计算机工程, 2009, 35(17):94-95.
- [3] Naohiro Hayashibara, Makoto Takizawa. Performance evaluation of the accrual failure detector[C]// Proceeding of the 26th IEEE international on Distributed Computing Systems Workshops (ICDCSW06). USA: Washington, 2006, 32(3):213-221.
- [4] 陈诚. 面向容灾的失效检测技术与实现[D]. 长沙: 国防科学技术大学, 2008.
- [5] 栾兰, 刘淑芬, 张欣佳. 基于检测点失效检测算法的研究与改进[J]. 吉林大学学报: 理学版, 2008, 46(4):681-

686.

- [6] Chen Wei Toueg S, Aguilera M K. On the quality of service of failure detectors[J]. IEEE Transactions on Computers, 2002, 51(5): 561-580.
- [7] Marin Bertier, Olivier Marin. Performance analysis of a hierarchical failure detector[C]// Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN'03). San Francisco, 2003, 21(3): 241-251.

作者简介:

陈宁江 男, (1975-), 博士, 副教授, CCF 会员. 研究方向为软件工程、网络分布计算等.

李昌武 男, (1982-), 硕士研究生. 研究方向为软件工程、中间件和服务计算等.

俞闽敏 女, (1986-), 硕士研究生. 研究方向为中间件、数据集成等.

(上接第60页)

由图4可知,随着任务最大关联度的增大,及任务之间的联系越复杂,本系统的执行效率较Condor和Pbs都有一定提高.

5 结束语

本文研究了网格环境下关联任务的迁移问题,提出了一种适合关联任务的迁移算法.该算法与传统集群系统的迁移机制相比,执行效率有一定的提高.算法侧重于待迁进程的选择和目的节点的确定,在迁移过程中要不断地访问数据库,提取关联任务的相关信息,尤其是关联任务数量较大时,对数据库的访问会占用一部分迁移的时间开销,因此,如何降低迁移的时间开销是下一步研究的重点.

参考文献:

- [1] 陈廷伟,张斌,郝宽文.基于任务-资源分配图优化选取的网格依赖任务调度[J].计算机研究与发展,2007,44(10):1741-1750.

- [2] 许丽娟.基于计算网络的集群系统负载平衡和进程迁移机制的研究与实现[D].上海:上海大学,2005.

- [3] Ferrai D, Zhou S. An empirical investigation of load indices for load balancing applications[M]. Englewood Cliffs, NJ: Prentice-Hall, 1988.

- [4] Kemal Efe, Margaret A Schaar. Performance of co-scheduling on a network of work stations[C]// USENIX Winter Conference. IEEE, 1993:525-531.

- [5] 陈廷伟,张斌,郝宽文.网格依赖任务调度系统的设计与实现[J].小型微型计算机系统,2008,29(3):455-460.

- [6] 杨永健,孙永雄,李秋秋,等.网格计算中一种负载均衡聚类匹配迁移算法[J].微电子学与计算机,2006,23(10):119-121.

- [7] 许封元,房至一,朱维平.进程迁移对负载平衡影响的实验[J].吉林大学学报:理学版,2006,44(6):925-930.

作者简介:

张千 女, (1982-), 硕士, 讲师, CCF 会员. 研究方向为云计算.