面向依赖关系的网格服务维护机制研究

吴 松[1],齐 力[2],金 海[1],石宣化[1],郑 然[1],陈 召[2]
(1. 华中科技大学计算机学院,湖北武汉 430074; 2. 国家开发银行营运中心,北京 100037)

摘 要: 在复杂的大规模分布式网格环境下,网格服务组件的维护是一个不容忽视的问题,不恰当的维护操作可能会带来不可预料的灾难. 本文关注由服务间的依赖关系引起的网格维护期间系统可用性问题,提出了一种网格服务维护机制,针对网格服务组件提供了多种粒度(服务级,容器级和节点级)的维护. 在使用了该机制后,网格管理员可以在运行时更安全地执行维护任务并保持较高的系统可用性. 测试结果显示,我们提出的面向依赖关系的服务维护机制使得网格的管理变得更加自动和高效.

关键词: 网格服务; 服务依赖; 服务维护
中图分类号: TP393 **文献标识码**: A **文章编号**: 0372-2112 (2009) 11-2359-08

Dependency-Aware Maintenance Model for Dynamic Grid Services

WU Song[1], QI Li[2], JIN Hai[1], SHI Xuan-hua[1], ZHENG Ran[1], CHEN Zhao[2]
(1. *School of Computer, Huazhong University of Science and Technology, Wuhan, Hubei* 430074, *China*;
2. *Operation Centre, China Development Bank, Beijing* 100037, *China*)

Abstract: Any mistaken maintenance for the complicated and distributed Grid can bring unpredictable disaster. Here we focus on the system availability problems caused by service dependencies during the maintenance in Grid. A novel mechanism is proposed in this paper. It provides multiple granularities (service-, container-, and node-level) maintenance for service components in Grid. By using the Cobweb Guardian, administrators of Grid can execute the maintaining task safely in runtime with high availability. The evaluating results show that our proposed dependency-aware maintenance can make the management for Grid more automatic and available.

Key words: grid service; service dependency; service maintenance

1 引言

随着网格[7]系统规模的不断发展,网格环境中存在大量提供各种功能的网格服务组件[16~20],如何高效维护在大规模复杂环境中的大量网格服务组件正逐渐成为网格系统不容忽视的重要问题. 对于网格管理员来说,维护任务贯穿于服务组件的整个生命周期. 网格服务组件的生命周期包括这样几个阶段:发布、部署、初始化、激活、销毁. 针对各个阶段,一次维护任务包括发布、部署、反部属、重部属、配置、激活和重激活. 网格服务组件维护中的一个难点是维护工作必须面对网格服务的依赖性,也就是说一些服务可能会存在内部的依赖关系. 对某一个服务组件的维护必须考虑这种依赖关系,才能保证使用该组件的服务继续正常运行,这在网格工作流或者组合服务的情况下尤其明显.

我们举例说明网格服务组件维护的重要性. 假设管理员在网格系统安装了执行系统,该执行系统由信息服务、计算终端服务和带数据冗余的数据中心组成,它为某些最终用户提供保证服务质量(QoS)的服务,一些关键服务在某天会因为更新要求而被重新部署. 这时管理员必须首先传送补丁或配置参数,然后远程调用维护脚本重新启动这些服务. 如果维护时间过长,大量的服务请求将被拒绝;另外,当重部署数据中心时,一些对服务质量要求较高的数据分段传输请求将会失败. 进一步,由于执行服务和数据中心之间很可能存在调用依赖,这些失败将导致相关作业请求失败. 更进一步,由于信息服务和数据中心可被部署于同一容器,对共同运行环境的重启动操作可能将同时影响两者,因此对信息服务的请求也可能失败. 而且,由于新的数据冗余服务和低版本的计算终端服务不匹配,对数据冗余服务的强制性重部署将很可能导致整个系统崩溃. 以上还没有考虑系统规模对维护工作的影响,如果在大规模网格系统中,成

收稿日期:2007-04-04;修回日期:2009-04-15
基金项目:国家自然科学基金(No. 60673174, No. 60603058, No. 60803006);国家 863 高技术研究发展计划(No. 2006AA01A115);教育部新世纪优秀人才支持计划(No. NCET-07-0334);教育部留学回国人员科研启动基金

百上千的网格服务会使维护工作涉及的问题更多.

上述问题都降低了整个系统的可用性,在这种情况下,一个高效的维护机制会从多方面给管理员带来方便.首先,维护机制应尽量减少管理员手工操作,不必每次都对每个服务执行维护;其次,如果维护机制具备分析依赖关系的能力,就能提供更高可用性的维护方案,并尽量避免依赖关系对维护的影响;第三,维护机制应支持多种粒度的维护,以此减少来自运行环境的影响.本文认识到大规模网格系统中分布式服务维护的重要性以及面临的挑战,因此所作的研究主要致力于回答如下问题:在服务间存在复杂的依赖关系的情况下,在对某个服务组件进行维护时,如何保证系统更高的可用性?

为提供动态网格环境下维护过程中系统的高可用性,本文提出了一种名为 Cobweb Guardian 新机制,该机制支持对服务组件多粒度(服务级、容器级和节点级)的维护.Cobweb Guardian 管理着由管理员初始化的各种服务依赖图,当一个维护请求到达时,它分析相关的依赖并生成一个减少依赖影响的最优维护方案.此外,Cobweb Guardian 为维护提供了会话管理机制以防止依赖关系可能带来的问题.

2 研究动机

2.1 概念

在进行进一步讨论前,首先说明一些基本概念:(1)在网格系统中,服务组件都是存在于某些特定容器里面,例如 Globus Toolkit 4 Java WS[3],这些服务组件公开提供了一系列的操作,使用这些操作可以组合出一些新的服务.服务组件之间的通信通常被包装在一个消息(Message)里面并通过简单对象访问协议(SOAP)传输;(2)本文中讨论的"依赖"指的是对服务组件的正确执行通常分别依赖于主机环境、被依赖的调用服务以及被依赖的部署服务;(3)对网格系统的一次维护包括对网格环境中一些特定服务组件的一系列操作(如部署,反部署……).通常,维护请求由管理员分发.

2.2 网格中的依赖关系

本节定义服务型网格服务的依赖模型,为后面的工作进行铺垫.假设有状态网格服务的集合:$S=\{s_0, s_1, \cdots, s_n\}$.

定义 1 服务依赖 R 是一种集合 S 上的二元关系,记做 $s_i \geq s_j$,它有以下性质:

(1)对于所有 $s_i \in S$, $s_i \geq s_i$ 存在,即任何服务依赖于它自己.所以依赖关系满足自反性.

(2)对于 $s_i, s_j, s_k \in S$,如果 $s_i \geq s_i$, $s_i \geq s_k$,就有 $s_i \geqq s_k$,表示依赖满足传递性,为了区别直接依赖和间接依赖,间接依赖记做"$\geqq$",特别地,如果 $s_k = s_i$,我们有 $s_i \geqq s_i$.

$S_{in}(s_i)$ 和 $S_{out}(s_i)$ 表示 s_i 的依赖和被依赖(包含直接和间接依赖)服务的集合,对于 S 中的任意两个服务,如果它们有相同的依赖和被依赖集合,它们就叫做同构的,同构的服务能并行部署或者放在目标节点的同一个包里.

定义 2 调用依赖中的"与"依赖、"或"依赖与"异或"依赖.设函数 $f(s_i, s_j)$ 为服务 s_i 与 s_j 调用关系.

(1)服务 s_i 的依赖集合中任取 $s_x \in S_{out}(s_i)$,均有 $f(s_i, s_x)=1$ 则表示 s_i 与 $S_{out}(s_i)$ 是与依赖的,称为 s_i 与依赖于 $S_{out}(s_i)$;

(2)服务 s_i 的依赖集合中存在 $s_x \in S_{out}(s_i)$,使得 $f(s_i, s_x)=1$ 则表示 s_i 与 $S_{out}(s_i)$ 是或依赖的,称为 s_i 或依赖于 $S_{out}(s_i)$;

(3)服务 s_i 的依赖集合中存在 $s_x \in S_{out}(s_i)$,使得 $f(s_i, s_x)=1$,而且任取 $s_y \in S_{out}(s_i) - \{s_x\}$,均有 $f(s_i, s_y)=0$ 则表示 s_i 与 $S_{out}(s_i)$ 是异或依赖的,称为 s_i 异或依赖于 $S_{out}(s_i)$.

定义 3 $\Sigma(s_i)$ 定义为服务 s_i 的直接依赖集,通过研究 $\Sigma(S)$,能够得到一个有向图 $DG<S, \Sigma(S)>$,它具有以下属性:

(1) S 中的每个 s_i 都是 DG 中的一个顶点.

(2) Σ 中的每一个依赖都是依赖有向边.

另外,定义 $R_k(s_i)$ $(k>0)$ 表示 s_i 的第 k 个传递依赖关系,$R_3(s_i)$ 表示与 s_i 的距离为 3,此外,$\Sigma_{closure}(s_i)$ 表示服务 s_i 的传递闭包,简单说,它表示 s_i 的所有依赖,公式里的符号 $\cup$ 表示相关集的并.

$$\Sigma_{closure}(s_i) = \bigcup_{k=1}^{n} R^k(s_i),\ n=|S|,\ n>0 \qquad (1)$$

定义 4 部署关键路径表示 $\Sigma(s_i)$ 的最小子集,其中的边可以连接所有 s_i 依赖的节点.通过探测路径上的所有节点,可以构造一个对于 s_i 的最小部署.

定义 5 定义一个 $n \times n$ 的矩阵——服务依赖矩阵(SDM),SDM 的每个元素 d_{ij} 表示 s_i 和 s_j 之间的直接依赖关系,如式(2)所示:如果 $s_i \geq s_i$,第 i 行和第 j 列就为 1,否则为 0.

$$d_{ij}=\begin{cases}1, & s_i > s_j\\ 0, & \text{else}\end{cases} \qquad (2)$$

此外,对于某个服务定义服务依赖度 $DGD(s_i)$,记录 s_i 依赖于多少其他服务,类似定义被依赖度 $DDD(s_i)$,表示有服务依赖于 s_i,DGD 和 DDD 的值在式(3)和式(4)中定义:

$$DGD(s_i)=\begin{cases}0, & n=1\\ \dfrac{1}{n-1}(\| SDM_i + SDM_i^2 + \cdots + SDM_i^n \| - 1), & n>1\end{cases}$$

(3)

式(3)中 SDM_i 表示 SDM 矩阵的第 i 行向量,SDM_i^k 表示第 k 个传递向量.此外,+ 符号定义了向量的二元或运算.

$$DGD(s_i)=\begin{cases}0 & ,n=1\\ \frac{1}{n-1}(\|T_i+T_i^2+\cdots+T_i^n\|-1) & ,n>1\end{cases}$$

$$,T=SDM^T \quad (4)$$

式(4)中,SDM^T 表示 SDM 矩阵的转置,为了部署,*DDD* 和 *DGD* 的值可以用引用的方式记录,他们有利于依赖策略评价和简化网格的部署,下面引出引理 1,引理 2.

引理 1　对于图 $DG<S,\Sigma(S)>$,定义 4 中给出了 $DDD(S)$ 和 $DGD(S)$:

(1) 如果 DG 中无环,对于 S 中的所有服务 s_i,$DDD(s_i)$ 和 $DGD(s_i)$ 的和小于 1.

(2) 如果 DG 中有环,则至少存在 2 个 S 中的服务 s_i、s_j,使得 *DDD* 和 *DGD* 的和大于 1.

(3) 特别地,这些代表服务的节点在环路上.

证明　第 1 和第 2 条容易从 *DDD* 和 *DGD* 的定义上证明,环上的服务必然属于环上其他服务的依赖和被依赖向量,自然会在 *DDD* 和 *DGD* 中被分别数两次,因此它们的和将大于 1.

对于第 3 条,采用反证法:假设存在不在任何环上的服务 s_i,它的 *DDD* 和 *DGD* 之和大于 1,根据前面的讨论,至少能找到一个服务 s_j 同时存在于 s_i 的依赖集和被依赖集中,也就是存在 $s_i \geqq s_j$ 并且 $s_j \geqq s_i$,根据定义 1,依赖传递性,就有 $s_i \geqq s_i$,表示 s_i 在一个环上,这与假设矛盾,所以第 3 条的结论成立.

引理 2　如果图 $DG<S,\Sigma(S)>$ 上有环,对于环上的每一个服务向量,他们的 *DDD* 相互相等(*DGD* 同理),也就是这些向量同构.

证明　根据定义 1 中同构的定义,定义 4 中 *DDD*,*DGD* 的定义,如 DG 中的任意两个服务 (s_i,s_j) 同构,他们就有同样地 *DDD* 和 *DGD*,因此只需证明 s_i 和 s_j 同构.

因为 s_i 和 s_j 在环上,有 $s_i \geqq s_j$,$s_j \geqq s_i$,对于 $S_{out}(s_i)$ 中的任意一个 s_k,易得 $s_i \geqq s_k$,$s_j \geqq s_k$,因此对于所有 $S_{out}(s_i)$ 中的服务,它也同样在 $S_{out}(s_j)$ 中,所以 s_i 和 s_j 的依赖集合相同,类似地,可以证明,它们的被依赖集也相同,所以 s_i 和 s_j 同构,引理 2 的结论正确.

2.3　维护方案和依赖关系

在这一节,我们将定量说明服务依赖对传统维护方案的影响程度.网格管理员通常采用串行(例如使用 Linux 的 Shell 脚本)或并行(例如采用 Java 或 C 线程)的方法来维护分布式系统.

基于语言的并行维护在这种解决方案下,如果不考虑依赖关系的影响,系统的可用性为:

$$A=1-(\max_{i=1}^{n}(t_r^i+t_p^i))/\sum_{i=0}^{n}t_i) \quad (5)$$

此外,总体维护时间是:

$$t=\max_{i}^{n}(t^i) \quad (6)$$

但是,当存在部署依赖时,系统的可用性将降为零,并且在维护后将变得不可恢复.这是由于在存在部署依赖的情况下,如果对某个服务进行部署的同时,部署这个服务所依赖的那个服务,则必定导致该服务部署的失败.例如,假设服务 A 部署依赖于服务 B,则实际部署过程应该是先部署 B 再部署 A,如果在部署 A 时同时部署 B,则必定导致部署的失败.此时,系统将永远不可用.

此外,这种方法同样不适用于调用依赖(如或依赖和异或依赖).因为无论先完成维护的是组合服务或服务组件,我们都牺牲了系统的可用性.事实上依次维护各个服务组件的方法所得出的系统可用性更高.

基于脚本的串行维护作为另外一种常用的日常维护方法,它提供如下的可用性:

$$A=1-(\sum_{i}^{m}(t_r^i+t_p^i))/\sum_{i=0}^{n}t_i) \quad (7)$$

在式(6)中,m 表示在 n 个服务上的维护步骤.这种情况下的可用性要低于并行的情况.此外,这种方法所花的总维护时间要长得多.(见式(7))

$$t=\sum_{i}^{m}t^i \quad (8)$$

虽然串行的维护方案可以解决部署依赖的问题,但由于它的低可用性和低有效性,大部分部署任务并没有采纳这种方案.

3　系统设计

3.1　维护粒度

维护网格系统中服务和主机环境之间复杂的依赖关系是一项大的工程任务.有效地减小维护粒度可以提高维护的有效性.在 Cobweb Guardian 中使用了三种粒度级别(服务级,容器级和节点级)来减少依赖关系对维护工作造成的影响.

(1) 服务级别　服务级的维护是指所有的维护任务都是针对服务组件的,重载和等待操作也是以服务为单位执行的.

(2) 容器级别　容器级的维护是指维护任务是针对整个容器的,也就是说,如果容器中的某个服务需要维护,那么该容器中的其它服务也将处于维护状态.这种级别在现有的一些软件如 GT4,MySQL server 和 HTTPD server 中得到广泛应用.虽然这种级别的维护粒度比服务级别要大,但是当一次维护某个容器中一系列服务组件时它可以花费更少的时间.

(3) 节点级别 节点级的维护是从全局的观点来看的.这个级别的维护任务是针对一系列的计算节点的.它可以平衡不同容器间的维护策略,通过分别与服务级和容器级维护工作的配合,可以提供更优化、更安全的维护方案.

3.2 体系结构

如图 1 所示,Cobweb Guardian 体系结构分成两个层次:Cobweb Guardian (CG) 和 Atomic Guardian (AG).一个 CG 和多个 AG 协同工作来执行网格环境下的节点级维护任务.

CG 由四个主要功能模块组成:会话控制、依赖优化、授权中心和策略控制. (1) 策略管理模块是为了让管理员能够按需执行维护任务. (2) 会话控制模块主要负责维护任务的进度,此外它也负责将维护任务传递到冗余的目标 AG. (3) 授权管理模块用来检测所有的维护请求,以避免不可预知的危险(如部署特洛伊木马的请求). (4) 依赖优化是 CG 的核心模块,它负责解析管理员的输入,同时将要求的和现有的依赖图进行匹配.

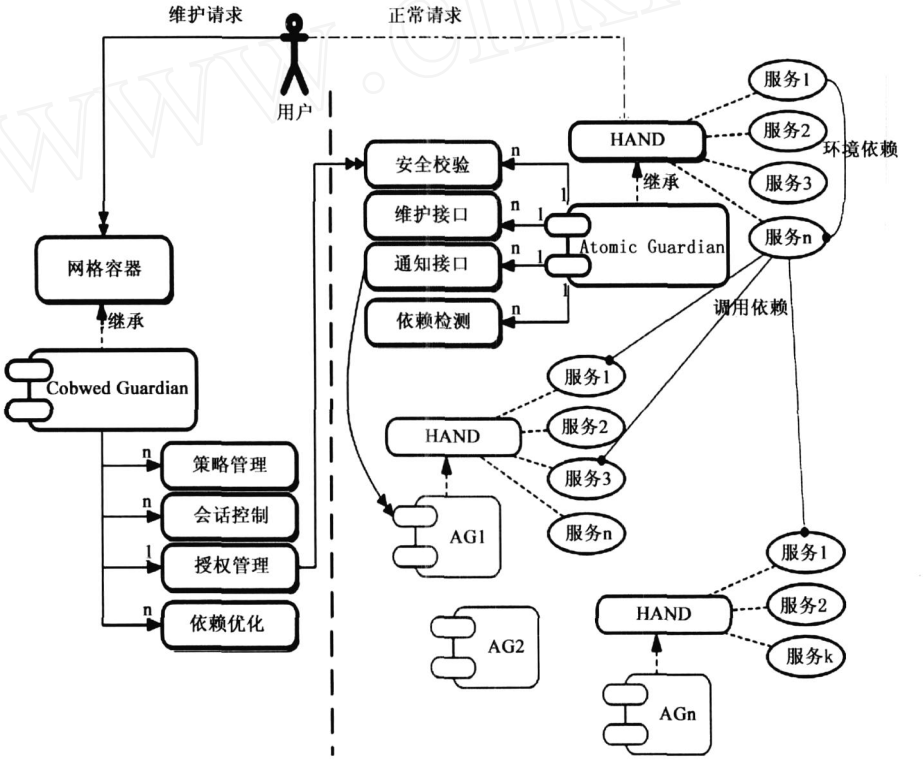

图1 Cobweb Guardian 的体系结构

AG 执行具体的维护操作,它的实现是基于我们以前的工作(HAND[11]).通过调用 AG,Cobweb Guardian 系统可以支持容器级别和服务级别的维护.AG 也是由四个部分组成:(1) 安全校验模块作为普通容器的防火墙,对到达的用户请求与维护请求进行分析,并检查对应的权限. (2) 维护接口接收来自会话控制的请求,它通过与主机容器管理模块的协作来完成部署、更新和激活任务. (3) 通知接口负责将维护状态通知有依赖关系的服务和 CG.通过这个模块,管理员可以掌握整个容器的维护进度并在第一时间检测到失败.通知将仅被发送到维护中依赖该服务的节点. (4) 依赖检测模块负责通过输入输出信息流来记录不同的调用依赖关系,任何被记录了的节点在进行节点维护时都会收到通知模块的通知,这样可以降低 CG 的工作负荷.

3.3 具有会话控制的维护传播

为了避免部署依赖带来的影响,CG 将在维护前检查依赖关系,同时生成一个关键维护路径来传递维护操作.也就是说,CG 提供了会话机制来保证维护过程.这种思想来源于并行的拓扑排序算法[8].为了确保维护操作能被正确执行,CG 首先将维护操作传递给它所依赖的服务.

3.4 依赖优化的分组维护

为了实现维护过程的高可用性,不同语义的调用应该被区别对待.根据 2.2 节的定义:(1) 对于与依赖,由于任何组件的不可用都会链式传播到整个系统,属于该类型依赖关系的服务维护都会减少系统的可用性.最好的解决方案是同时将全部维护传播到目标容器. (2) 对于异或依赖和或依赖,因为一般请求通常都是以某种语义(如循环策略,随机策略,负载均衡策略) 发送到一系列相关服务的,如果对于这些服务的维护分组执行,系统

可用性将会有一定程度的提高.

3.5　带反馈的分组维护

虽然分组方案牺牲了维护时间来换取可用性,但是它仍然会给系统带来一些不可预料的因素.例如,一些关键的调用请求可能由于维护原因而随机地被拒绝掉.CG为网格应用提供了反馈通知机制.由于CG定期的检查远程服务组件的运行状态,网格应用可以调用CG的回调APE而获得这些状态,从而避免可能的错误调度.使用这种方式,系统可用性将得到很大的提高,可用性的代价主要来自组合服务组件的维护.

4　测试

我们以中国教育科研网格系统ChinaGrid为测试平台验证了我们的想法和方案.实验所用的服务均在中国教育科研网格系统支撑平台[1]CGSP2.0.1版本上进行过测试.

我们进行测试的目标包括:

(1)证明与传统方法相比,Cobweb Guardian提高了维护过程中系统的可用性;(2)比较Cobweb Guardian的各种维护方案,了解在不同调用下进行维护时,哪种方案使系统具有较高的服务可用性和吞吐率;(3)证明在动态性方面,"依赖反馈"和"多粒度维护"是行之有效的.

4.1　测试环境

如未特别说明,本文的实验环境在两个Linux集群上进行.其中一个集群由16个节点组成,节点机的CPU为1 GHz的Pentium Ⅲ处理器,内存为512M,运行内核版本2.4.20-8的Red Hat Linux以及Sun实现的J2SDK 1.5.0-06-b05;另一个集群由20个双CPU(1.3 GHz Itanium2)服务器组成,运行内核版本为2.4.0-2的Red Hat Linux以及BEA实现的J2SDK 1.5.0_03-b07.集群内节点以100M以太网连接,两个集群间由100M带宽的网络连接.

缩写词　以下是本节将要用到的缩写词:(1)NonD:不考虑依赖关系的维护方案;(2)SRL:利用脚本实现的串行化维护方案;(3)CG-0:简单的传播维护方案,未对调用依赖关系进行优化;(4)CG-1:不同调用依赖关系下的分组维护方案;(5)CG-2:带反馈的分组维护方案;(6)REQ:对特定服务组件的恒定请求率.

评测程序　实验基于中国教育科研网格系统支撑平台(CGSP)提供的典型执行系统——通用运行服务.如图2所示,通用运行服务包含有6个服务组件:认证服务、信息服务、数据服务、缓存服务、查询服务以及副本服务.当一个请求到来时,通用运行服务首先对请求进行分析,然后调用认证服务组件检查有效性、调用信息服务组件获取作业执行信息、调用数据服务组件获取各阶段所需的数据.更进一步,数据服务组件获取作业执行信息时,会首先尝试调用副本服务,如果失败则再调用查询服务,查询服务会在后端的数据库中进行查询.另一方面,数据服务采用副本方式存取阶段性数据.信息服务与通用运行服务可以含有多个组成部分.有关CGSP执行系统的详细描述可参考文献[15].通过对这些服务的现场维护,我们可以观察Cobweb Guardian的效果.

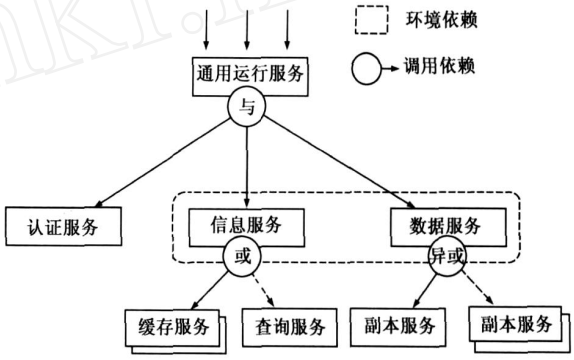

图2　CGSP执行系统的服务依赖关系

4.2　具有会话控制的维护评测

为研究部署依赖的影响,在这个实验中,我们每秒发出6个对通用运行服务的请求.实验共持续了140s,在第15s,我们开始对通用运行服务进行维护(包括认证服务组件、信息服务组件、数据服务组件以及通用运行服务本身的升级).维护开始后,服务会立即停止响应一段时间.本实验测试了3种方案:NonD、SRL和CG-0.

图3显示了三种维护方案在30s观察其内的吞吐率.还未开始维护时,所有的系统都工作良好.第15s维护开始,在很短的时间内,三种方案的吞吐率都迅速降低至零.NonD方案最早完成维护工作,但系统不能继续正常工作.NonD方案没有考虑服务组件间的部署依赖关系,导致通用运行服务组件升级失败.而SRL方案完成维护的耗时最长,因此显然CG-0方案的效果最好.在

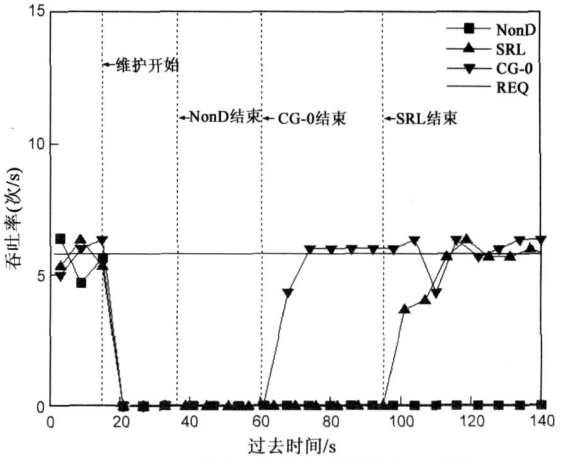

图3　不同方法在维护期间内的服务吞吐率

相关的服务组件(认证等)正在维护时,与之有"与依赖"关系的服务不能正常工作,因此 CG-0 丢失了 49.1 %的请求.

4.3 针对调用依赖的维护机制

在下面两个实验中,我们使用信息服务评估在不同调用依赖关系下,Cobweb Guardian 在提高服务可用性方面的效果.我们每秒发出 10 个对信息服务的请求.

或依赖 如图 4 所示,我们在第 15s 分别启动 CG-0 和 CG-1 两种维护方案.CG-0 以遗传方式将维护任务首先发到部署有副本服务和查询服务的容器,然后再对部署有信息服务的容器进行维护,整个过程耗时 62.97s.从表 1 我们得知,在 CG-0 方案执行过程中,吞吐率降低至接近于零.此外,在观察期内,请求丢失率为 57.9 %.

表 1 实验结果对比

方法	维护点	反映时间(ms)	吞吐率(req/s)	丢失率(%)
CG-0	之前	443.9	9.51	0
	进行时	N/A	0.86	57.9
CG-1	之前	463.8	9.92	0
	进行时	929.8	7.08	25.6

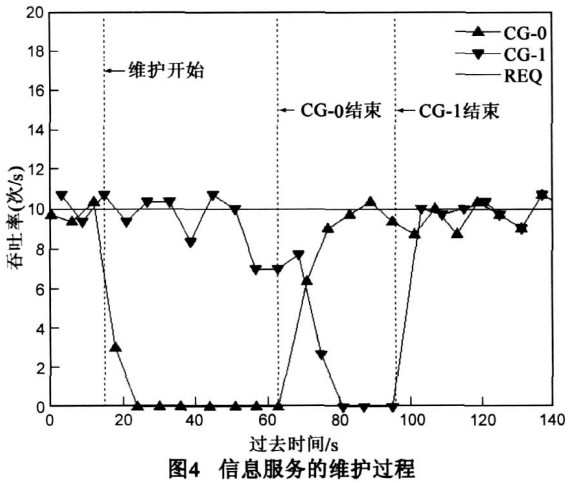

图4 信息服务的维护过程

相较 CG-0 方案,CG-1 方案对"或依赖"关系进行了优化.Cobweb Guardian 按照不同的优先级对组内容器依次进行维护,效果的改善相当明显:首先,维护期间内的吞吐率提高至每秒 7.08 次;第二,尽管 929.8ms 的平均响应时间大约是维护前的两倍,但请求丢失率从 57.9 %下降至 25.6 %.因为信息服务的维护是不可避免的,可以得知 25.6 %是所有方案所能达到的最低请求丢失率.从图中我们可以看到,CG-1 的吞吐率从 57s 开始下降至每秒 7.3 次,这是由于 Cobweb Guardian 开始维护副本服务时,信息服务对副本服务的调用尝试会失败,继而会尝试调用查询服务,这个过程将会延长响应时间.

图 5 显示了上述几个过程的响应时间.我们可以清楚地看到:当副本服务于第 57s 开始维护时,平均响应时间有所增加.对于 CG-0 方案来说,CG-1 方案的改进是令人满意的.美中不足的是 CG-1 方案的总维护时间比 CG-0 方案多出了 32.985s.

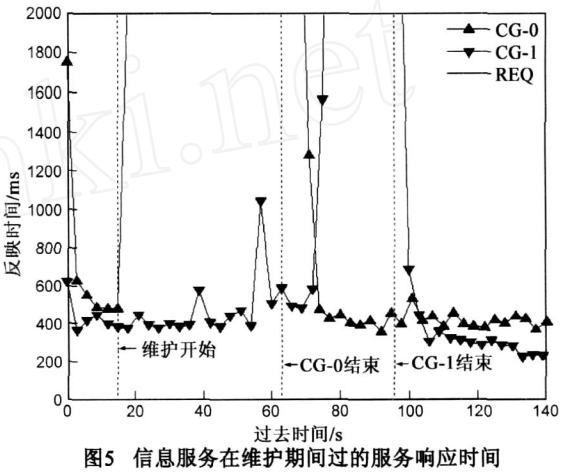

图5 信息服务在维护期间过的服务响应时间

异或依赖 我们对数据服务组件(异或依赖关系)重复该实验,这次新增了 CG-2 方案.CG-2 方案会将维护状态反馈给用户层,这使得正在被维护的服务不会被无谓地调用.同前面的实验一样,我们在第 15s 分别启动 3 种维护方案.

图 6 描述了整个维护过程:CG-0 方案和前一次实验类似,在维护期间所有对数据服务的请求全部处于阻塞状态.因此 CG-0 方案的吞吐率最低,CG-1 方案的效果也不佳.虽然 CG-1 方案的吞吐率达到了 4.71(CG-0 方案是 0.01),请求丢失率为 52.1 %,相较 CG-0 方案的 53.9 %略有降低.出现这种情况的主要原因是 CG-1 方案用了更长的时间进行维护,在这段时间内数据服务总是尝试请求副本服务,而这些请求显然会失败.CG-2 方案比前两种方案的效果要好得多,CG-2 方案维护用时与 CG-0 方案相同,但提供了更高的吞吐率(每秒 6.56 次)、更低的请求丢失率(25.9 %)以及对于正常请

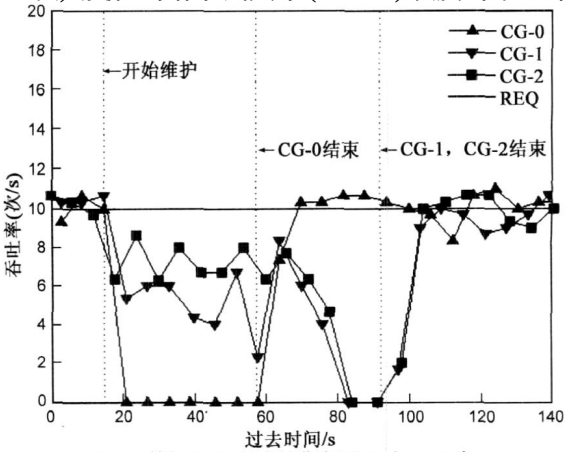

图6 数据服务在维护期间的服务吞吐率

求更短的响应时间(478ms).

表 2 分别列出了 3 种维护方案在不同阶段(维护前与维护后)的请求响应时间、吞吐率和请求丢失率.这张表反映了前述 3 种维护方案的差异.结果证明了 Cobweb Guardian 具备为不同的依赖关系寻找最佳维护方案的能力.

表 2　实验结果对比

方法	维护点	反映时间(ms)	吞吐率(req/s)	丢失率(%)
CG-0	之前	552.5	9.21	0
	进行时	N/A	0.01	53.9
CG-1	之前	405.6	9.43	0
	进行时	933.2	4.71	52.1
CG-2	之前	405.1	10.07	0
	进行时	478.5	6.56	25.9

4.4　动态评估

网格服务的动态变化是非常普遍的,本实验将研究 Cobweb Guardian 在动态变化情况下的维护效果.在对数据服务和副本服务进行正常维护时,我们将反部署其中一个服务的副本.

本实验采用 CG-1 方案.图 7 描述了本次维护吞吐率的变化.从图中我们可以看到,维护后的总吞吐率降低至维护前初始吞吐率的 2/3.这证明副本服务被正确地反部署.另外,Cobweb Guardian 阻塞对处于维护状态下的服务组件的请求过程,吞吐率分三个阶段逐渐降低.同时,维护期间的请求丢失率是 25.7%,是一个相对比较低的值.

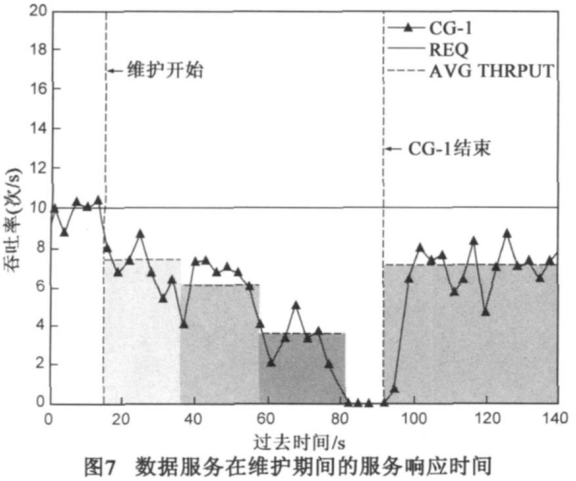

图7　数据服务在维护期间的服务响应时间

5　相关工作

通过探索软件组件之间依赖关系进行软件(服务)维护的方案在前人的研究中已经被广泛讨论过.

在面向对象编程的年代,Stephen[14]等人提出一种使用一阶逻辑进行知识表现来表示软件组件互联信息,以促进软件开发或修改期间各组件互联信息的正确性和完整性检查.他们使用组件互联图(CIG)来表示调用依赖.但是当从面向对象转换到面向服务架构后,CIG 则无法表示不同节点间的依赖关系.

Neeraj[12]等人发明了依赖结构矩阵来描述遗产软件间的依赖关系.从软件工程的角度来看,它在浏览软件架构方面很有效.遗憾的是,他们没有讨论在分布式的动态网格中非常常见的有状态的依赖.因此,依赖结构矩阵不方便适用于网格的动态部署.

服务胶囊(Service capsule)是由 Lingkun[6]等人提出的一种新机制,该机制支持依赖状态的自动识别和基于线程服务的依赖管理.然而该机制对容错的关注要明显多于维护.另外,它工作于多线程的集群服务器.

系统可用性是分布式系统的一个非常重要的性能指标,在文献[6,11,13]中对它做了详细的介绍.评价一个系统整体可靠性的两个重要指标是 MTBF(平均无故障时间)和 MTTR(平均恢复时间).通常我们需要花很长的时间来测试这些指标.最近提出了一种叫做错误注射(Fault Injection)的方法,该方法提供了有效并且花费较短时间来获得系统可用性的途径[10].

在前面的工作中[11],我们提出了两种用于提高服务架构可用性的动态部署方案(服务级和容器级).结果证明使用小粒度部署方案可以在一定程度上提高系统可用性.但是不管是服务级别还是容器级别,性能的提高受到了基础组织的制约.特别是当存在于不同容器中的服务之间存在复杂调用依赖的时候,它无法保证全局的可用性.

由 GGF 制订的配置、描述、部署以及生命周期管理(CDDLM)的规范[2]和由 W3C 制订的可安装单元部署描述规范(IUDD)[4]规范化了分布式软件或服务的维护工作.但是这些规范不能保证维护操作的质量和维护运行时可用性.Smart Frog[5]工程作为 CDDLM 的一个很大的引用,是一款经典的更改和配置管理工具.

Vanish[13]等人从刻度、复杂度、表示和障碍等方面对比分析了分布式服务下包括基于人工、脚本、语言、模型的部署方案.虽然文章考虑了依赖的问题,但是没有关注系统部署对系统可用性的影响.

Nagaraja 等人在文献[9]中讨论了互联网服务中的操作错误.这篇文章示范了如何通过建立确认环境来探测操作错误,该确认环境作为在线系统的一个扩展,其组件在组合成一个可运行的服务前使用实际任务进行确认.但是这种方案只能用来探测和防止维护错误,不能优化运行时的系统可用性.

6　结论与展望

本文提出了一种网格环境下的面向服务依赖关系

的维护体系—Cobweb Guardian.通过研究网格运行时的不同类型依赖关系(调用依赖、部署依赖和环境依赖)的影响,Cobweb Guardian 能够生成网格环境下的优化维护方案.测试证明 Cobweb Guardian 可以有效地提高系统维护期间可用性和吞吐量.

未来的主要工作是研究网格服务维护机制的系统开销问题,对 Cobweb Guardian 机制的维护开销进行分析和测试,并对分布式维护中容错机制开展研究.

参考文献:

[1] China Grid Support Platform [OL]. http://www.chinagrid.edu.cn/cgsp

[2] Configuration, Deployment Description Language and Management, GGF [OL]. http://www.gridforum.org/documents/GFD.50.pdf

[3] Globus Toolkit Project. Globus Alliance [OL]. http://www.globus.org

[4] Installable Unit Deployment Descriptor Specification [OL]. http://www.w3.org/Submission/InstallableUnit-DD/

[5] Smart Frog Project [OL]. http://www.smartfrog.org.

[6] L Chu, K Shen, et al. Dependency isolation for thread-based multi-tier internet services [A]. Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies [C]. Miami, FL, USA: IEEE Press, 2005. 796 - 806.

[7] Foster, C Kesselman, S Tuecke. The anatomy of the grid: enabling scalable virtual organizations [J]. International Journal of Supercomputer Applications, 2001, 15(3): 200 - 222.

[8] D Kimelman, D Zernik. On-the-Fly: Topological sort a basis for interactive debugging and live visulization of parallel programs [A]. Proceedings of the 1993 ACM/ONR Workshop on Parallel and Distributed Debugging [C]. San Diego, California, US, 1993. 12 - 20.

[9] K Nagaraja, F Oliveria, R Bianchini, et al. Understanding and dealing with operator mistakes in internet services [A]. Proceedings of 6th USENIX Symposium on Operation Systems Design and Implementation (OSDI'04) [C]. San Francisco, CA, US, 2004. 61 - 76.

[10] K Nagaraja, X Li, B Zhang, R Bianchini, et al. Using fault injection and modeling to evaluate the performability of cluster-based services [A]. Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems [C]. Seattle, WA, Vol 4, 2003. 2 - 3.

[11] L Qi, H Jin, I Foster, et al. HAND: highly available dynamic deployment infrastructure for globus toolkit 4 [A]. Proceedings of the 15th Euromicro Conference on Parallel, Distributed and Network-based Processing [C]. Naples, Italy, 2007. 155 - 162.

[12] N Sangal, E Jordan, V Sinha, et al. Using dependency models to manage complex software architecture [A]. Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming systems languages and applications [C]. New York, NY, USA: ACM, 2005, 40(10): 167 - 176.

[13] V Talwar, D Milojicic, et al. Approaches for service deployment [J]. IEEE Internet Computing, 2005, 9(2): 70 - 80.

[14] S Yau, J Tsai. Knowledge representation of software component interconnection information for large-scale software modifications [J]. IEEE Transactions on Software Engineering, 1987, SE-13(3): 355 - 342.

[15] Y Wu, S Wu, H Yu, et al. CGSP: An extensible and reconfigurable grid framework [A]. Proceedings of the 6th International Workshop on Advanced Parallel Processing Technologies [C]. Hong Kong, China, 2005. 292 - 30.

[16] 胡春明,怀进鹏,孙海龙. 基于 Web 服务的网格体系结构及其支撑环境研究 [J]. 软件学报, 2004, 15(7): 1064 - 1073.
HU Chun-min, HUAI Jin-peng, SUN Hai-long. Web service-based grid architecture and its supporting environment [J]. Journal of Software, 2004, 15(7): 1064 - 1073. (in Chinese)

[17] 王意洁,肖侬,任浩,卢锡城. 数据网格及其关键技术研究 [J]. 计算机研究与发展, 2002, 39(8): 943 - 947.
WANG Yi-jie, XIAO Nong, REN Hao, LU Xi-cheng. Research on key technology in data grid [J]. Journal of Computer Research and Development, 2002, 39(8): 943 - 947. (in Chinese)

[18] 徐志伟,冯百明,李伟. 网格计算技术 [M]. 北京:电子工业出版社, 2005. 1 - 10.

[19] 钱德沛. 基于 Internet 网格计算模型研究 [J]. 西安交通大学学报, 2001, 35(10): 1008 - 1011.

[20] 翁楚良,李明禄,陆鑫达. 面向服务的网格高性能计算策略 [J]. 小型微型计算机系统, 2006, 27(10): 1793 - 1797.
WENG Chu-liang, LI Ming-lu, LU Xin-da. Service-oriented strategy for high performance computing application based on grids [J]. Mini-micro Systems, 2006, 27(10): 1793 - 1797. (in Chinese)

作者简介:

吴 松 男,1975 年出生于湖北武汉.教授、博士生导师、中国计算机学会高性能计算专委会委员、中国电子学会云计算专委会委员.2003 年在华中科技大学获计算机系统结构专业工学博士学位,2007 年入选新世纪优秀人才支持计划.现为华中科技大学服务计算技术与系统教育部重点实验室副主任、计算机科学与工程系副主任、中国国家网格武汉结点负责人.主要从事网格计算、计算系统虚拟化等方面的研究工作.

E-mail: wusong@hust.edu.cn