计算机科学2004Vol.31№.2

基于CORBA架构的组件负载平衡的设计与实现

张 硕 顾毓清
(中国科学院软件研究所 北京100080)

摘 要 由于目前OMG的CORBA规范里没有实现负载平衡,本文在CORBA标准协议的基础上扩展设计了组件负载平衡服务。客户由负载平衡服务器重定向到一台轻负载的簇集服务器上。
关键词 CORBA,对象请求代理,IDL,组件负载平衡

CLB's Design and Implementation Based on CORBA Framework

ZHANG Shuo GU Yu-Qing
(Institute of software,Chinese Academy of Sciences,Beijing 100080)

Abstract Because there has been no load-balanceing service in OMG's CORBA now,this paper intruduces the extend design and implementation of component load-balanceing service based on CORBA standard protocal. The CLB server receives a client's initial request,designates a lightly-loaded machine in the cluster servers as a proper host for the requested instance,and re-direct the client's invocations to it.
Keywords CORBA,ORB,IDL,CLB

随着计算机技术的不断发展,向用户提供可伸缩、更高效、易于管理的服务逐渐成为一种趋势。传统的多对一的Client/Server模型已经不能满足用户日益增长的需要,多对多的C/S模型越来越受拥有大量用户的企业的欢迎。

1 CORBA的主要概念

通用对象代理体系结构CORBA(Common Object Request Broker Architecture)是对象管理组织(OMG)所定义的用来实现现今大量硬件、软件之间互操作的解决方案。CORBA标准主要分为三个部分:接口定义语言(IDL)、对象请求代理(ORB)以及ORB之间的互操作协议IIOP。

接口定义语言[1]提供了将对象的接口与其实现分离的能力。IDL提供了抽象,它提供了将事务与其具体实现分离的概念。它还为我们提供了一套通用的数据类型使得我们可以使用它们来定义更为复杂的类型。我们将采用所有这些数据类型来定义分布式服务的功能。IDL的另一个好处是它剥离了编程语言和硬件的依赖性。ORB是对象之间建立C/S关系的中间件。使用ORB[2],客户可以透明地调用一个服务对象上的方法,这个服务对象可以在本地,也可以在通过网络连接的其他机器上。ORB截获这一调用同时负责查找实现服务的对象并向其传递参数、调用方法返回最终结果。基于CORBA架构的C/S模型,客户并不知道服务对象位于什么地方,它的编程语言和操作系统是什么,也不知道不属于对象接口的其他系统部分。这样,ORB在异构分布环境下为不同机器上的应用提供了互操作性,并无缝地集成了多种对象系统。由于CORBA的以上特性,采用CORBA作为多对多C/S模型的中间件实现Client和Server之间的通信成为许多软件开发商的选择。

2 多对多的群集服务

在多对多的C/S模型下,众多提供服务的Server组件组成一个群集,对客户来说这个群集是透明的,就相当于一个Server组件在提供服务。客户不知道也不关心有多少服务提供者、这些服务是在什么地方实现的、服务的编程语言和操作系统是什么。客户只关心自己提交请求的正确、迅速的响应。

要为客户提供正确、迅速的响应,必须知道Server群集中每一台Server组件的工作情况,有多少个Client的请求正在上面处理,网络的拥塞情况,Server组件的响应时间如何。但是OMG提出的CORBA架构本身并没有提供这种服务,本文就CORBA在组件负载平衡(CLB)方面的不足提出一些扩展设计及其实现。

3 系统设计

3.1 工具选择

我们采用当前应用比较广泛的Visibroker作为CORBA的实现。Visibroker[3]提供的Smart Agent是一种动态的、分布的、用于定位对象服务的目录服务,它必须在局域网的至少一台计算机上运行。ORB利用UDP协议找到Smart Agent,如果局域网内运行有多个Smart Agent,ORB将使用第一个发出回应的Agent。一个服务对象首先在Smart agent上注册,这样Smart Agent维护一个可以使用的服务的列表。当一个Client发出一个请求,ORB利用Smart Agent上的信息定位这个请求相应的实现对象以及在需要的时候激活这个对象。

3.2 CLB Server(组件负载平衡服务)

负载平衡将最终用户的通信量分配给所有可用的服务器——在同一数据中心内或在全球范围——以避免一个服务器阻塞而另一个服务器空闲。连接和会话速度得到最大化,因而所有用户均得到优化的性能。另外通过灾难防护,还可以受益于更高的可靠性。如果一个站点关闭,通信量会被立刻重新定向到另一区域内的服务器。

使用CLB Server时,不是在本地服务器上创建组件,而是用路由列表来帮助把客户的请求传递到已实现负载平衡的Server群集。然后,通过Client和Server的一次握手,实现Client和Server的直接会话。这个会话一经创建,CLB Server就不再对其进行更多的操作。

3.3 路由列表

路由列表位于CLB Server上,在路由列表里存储每一个Server组件的位置,为路由列表提供支持的包括每一个处于激活状态的Server组件的访问计数表和响应时间表,如图1

张 硕 硕士研究生,主要研究领域为软件工程;顾毓清 博士生导师,主要研究领域为软件工程。

所示。

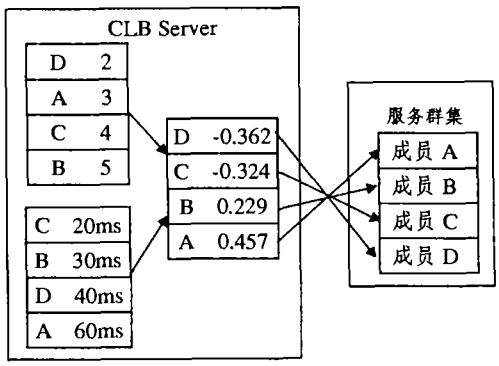

图1 路由列表

路由列表最初以每个 Server 组件提供等质的服务初始化，每个 Server 组件在生存周期的开始自动在 CLB Server 上注册，在生命周期的结束解除注册。响应时间表在初始化的时候为无穷大，访问计数表为零。在以后的时间里 CLB Server 上的每一个群集成员的信息和 Server 组件本身自动保持同步。这样一来，就可以实现路由信息的动态维护，并且支持群集成员动态地增加和减少。

(1)*响应时间表* 每隔一定时间(如：500 毫秒)，CLB Server 就对已经注册的 Server 组件进行轮询，更新响应时间表。在响应时间表里保存每一个 Server 组件对 CLB 问询的返回时间——响应时间越快的 Server 组件优先级越高。响应时间的长短反映 Server 所在的网络的状况和目前 CPU 的负荷轻重。

(2)*访问计数表* 一旦 Client 要求连接 Server 组件，首先向 CLB 提出申请，CLB 在路由列表里选择一个响应最快、最不繁忙的 Server 组件，并把这个 Server 的组件标识返回给提出申请的 Client。Client 用这个标识和 Server 组件建立连接，如果成功则由 Server 把增加了服务数的信息返回给 CLB，CLB 则把相应 Server 组件的访问计数加一，刷新访问计数表。当 Client 想断开与 Server 的连接时，也必须显式地由 Server 组件通知 CLB，供 CLB 对访问计数进行减一操作。

(3)*路由表* 是由响应时间表和访问计数表的数据生成的。在 CLB 轮询 Server 组件以后，开始路由表的更新工作。路由权值的算法可以根据系统的实际需要来决定，这里介绍比较普遍和简单的一种。

R_i 为第 i 个 Server 组件的访问数，T_i 为第 i 个 Server 组件的响应时间，$\overline{R}$ 为访问数的平均值，$\overline{T}$ 为响应时间的平均值，W_R 为访问数的权值，W_T 为响应时间的权值，P_i 为第 i 个 Server 组件的优先级的值。

$$P_i=(R_i-\overline{R})/\overline{R}*W_R+(T_i-\overline{T})/\overline{T}*W_T$$

其中 P_i 越小优先级越高，一般情况下 W_R 和 W_T 同取值1，表示访问数和响应时间权重相同。在这种情况下，P_i 的取值范围为$[-2,2n-2]$，n 为 Server 组件的数目。

这样每次 Client 提出服务申请，CLB 就在路由表里选择一个优先级值最小的 Server 组件返回给 Client，如表1所示。

表1 优先级表

	响应时间	响应时间权	访问计数	访问计数权	优先级
A	0.600	1	−0.143	1	0.457
B	−0.200	1	0.429	1	0.229
C	−0.467	1	0.143	1	−0.324
D	0.067	1	−0.429	1	−0.362

4 系统的健壮性

4.1 单点故障

(1) Server 端 当一个 Server 组件发生故障，CLB 在进行轮询的时候就会发生超时的错误。一旦这样的错误发生，CLB 立即将这个组件的响应时间更改为无穷大，这样就可以立即阻止其他 Client 和 Server 建立连接，防止错误的进一步扩散。如果这样的超时错误连续发生三次，系统就认为这个 Server 组件已经死亡，在路由表里永久的删除这个组件，直至这个 Server 作为一个新的组件再次注册。相应地，和这个发生故障的 Server 组件已经建立连接的 Client 也会收到超时的错误，一旦超时，Client 可以向 CLB 提出新的连接请求，这样 CLB 就会把 Client 重新定位到一个正常工作的 Server 组件上。

(2) Client 端 如果已经和 Server 组件建立连接的 Client 发生故障，由于 CLB 的轮询机制并不包含 Client，因此 CLB 不能直接查询 Client 的状态。这样就需要在 Client 和 Server 组件之间建立 key-to-key 的机制，具体实现如下：当 Client 首次向 Server 提交会话请求时，Server 生成一个全局唯一的会话句柄并把这个会话句柄传递给 Client，作为这个 Client 的唯一标识，此后 Client 的每一次服务请求都带有这个句柄。Server 在 Client 的最后一个服务请求结束以后开始计时，如果超过一定时间(如：1个小时)Client 还没有新的请求，就认为这个会话已经丢失，则关闭这个会话请求，并且在 CLB 上把自己的访问计数减一。如果 Client 并没有真的发生故障，当它再次向 Server 组件提出服务请求时，由于它使用的句柄已经过期，则会收到 Server 抛出的一个异常，这时就需要 Client 重新向 CLB 提出连接请求，建立新连接。

4.2 群集的动态扩张与缩减

(1)*动态扩张* 当系统中 Server 的数量不能满足需要时，就需要增加 Server 的数量。在本系统的架构下，Server 的增加不需要对服务进行停机处理。当在一台新的 Server 组件启动时，它会自动在 CLB 上实现注册，并把自己添加在路由表里。在 CLB 的第一次轮询以后，这个 Server 组件就像其他的组件一样有了自己的响应时间表和访问计数表，并向 Client 提供服务。

(2)*动态缩减* 当需要减少一个 Server 组件时，Server 首先向 CLB 发出一个关闭的请求，CLB 把相应的访问计数表更改为无穷大。这样 CLB 就会拒绝所有新的 Client 的请求映射到这个 Server 组件上。接下来可以分为两种关闭方式"立即"和"等待 Client 重新定位"。在"立即"这种方式下，Server 立即关闭，这样 Client 的服务申请会得到 CORBA 抛出的超时异常。Client 会认为 Server 已经死亡，并向 CLB 提出新的连接请求。在"等待 Client 重新定位"这种方式下，Server 对已经连接的 Client 提出的新的服务请求抛出句柄已经过期的异常，这样就强制 Client 重新向 CLB 提出连接请求。当这个 Server 上所有的 Client 都重定向了其他的 Server 以后(即 Server 的所有会话都已经过期)，这个组件就可以安全的关闭了。当组件关闭以后，CLB 在轮询的时候就会收到超时的错误，三次超时过后这个 Server 组件的注册就会被永久删除。

5 系统实现

5.1 Basic Object Adapter(BOA)与 Portable Object Adapter(POA)

轻便对象适配器(POA)代替了 visiBroker 原有的基础对象适配器(BOA)，它可以在 Server 端提供更加轻量级的服务。POA 作为 ORB 和组件服务之间的桥梁，服务管理者在 POA 上定位并且指派一个对象服务。在每一个 POA 上都存在一个活动对象映射表，上面把对象 ID 和活动对象以及它提供的服务绑定起来，每一个服务都和一个 ID 一一对应，这个 ID 就是服务的唯一标识。在 Visibroker 里，这个对象 ID 是由一个全局唯一的字符串表示。

5.2 服务标识

(1)标识生成 由于对象 ID 在整个系统里是唯一的,因此对象 ID 由两部分组成,第一段是 Server 组件所在机器的 IP 地址,第二部分是服务的名字。这样由于各个计算机 IP 地址不同,就可以保证不同机器有不同 ID,服务的名字则区分了同一台机器上各个不同的服务。

(2)标识解析 Server 组件需要实现两个注册,一个是 CORBA 的注册,把对象的唯一标识和对象实现在 POA 上绑定,这样 Client 就可以通过这个对象 ID 对 Server 提供的服务进行访问。由于 Server 组件对 Client 提供匿名服务,Client 仅仅通过服务名无法定位组件服务,因此 Server 组件还需要把自己的对象 ID 在 CLB 上注册,只有 CLB 知道服务的真实对象 ID。当 Client 向 CLB 提出服务请求时,CLB 根据 Client 提供的服务名,以及已经注册的组件服务,得到多个提供这个服务的对象 ID,并根据路由列表得到服务的最优提供者,接下来把这个最优的对象 ID 回传给提出申请的 Client。

5.3 IDL 设计

CLB 端的 IDL

```
//CLBSvr.idl
module CLBSvr
{
    interface CLBForClient
    {
        string getServer(in string serviceName);
    };
    exception LessThanZeroExp
    {
    };
    interface CLBForServer
    {
        long regService(in string serviceFullName);
        void pleaseCloseMe(in string serviceFullName);
        void addRef(in string serviceFullName);
        void releaseRef ( in string serviceFullName ) raises
        (LessThanZeroExp);
    };
};
```

Cluster Server 端的 IDL

```
//ClusterSvr.idl
module ClusterSvr
{
    interface ClusterForCLB
    {
        string ping();
    };
    exception LostSessionExp
    {
    };
    interface ClusterForClient
    {
        string getSession();
        void closeSession ( in string sessionHandle ) raises
        (LostSessionExp);
        string workWithSvr(in string sessionHandle,in long params)
        raises(LostSessionExp);
    };
};
```

5.4 Session Table

在 Cluster Server 上需要管理 Client 的会话句柄,以唯一地标识每一台 Client。在这里可以使用 Session Table 机制。

(1)Session 的生成和传递 具体实现如下:Client 首先调用 CLB Server 的 getServer 接口函数,获得 Cluster Server 的组件标识,通过这个标识和 Cluster Server 建立连接;然后 Client 调用 Cluster Server 的 getSession 方法,Cluster Server 生成一个随机的字符串序列,保存在自己的 Session Table 里作为 Session 的 ID,并把这个 Session ID 返回给 Client;在随后的 Client 每一次访问 Cluster Server 时都要带有这个 Session ID,Cluster Server 在自己的 Session Table 里检索这个 Session,如果存在则提供访问,反之则抛出一个 LostSessionExp 的异常。

(2)Session 的超时判断 Cluster Server 首次生成 Session 的时候,同时保存在 Session Table 里的还有此时的系统时间。在此后每一次 Client 的服务调用时都要更新这个系统时间。Cluster Server 上有一个 Session Manager 的类,除了生成、注册 Session 这些基本的管理以外,还要启动一个线程,这个线程每隔一定时间刷新一次 Session Table,把 Session Table 里每一个 Session 的最后一个访问时间和当前的系统时间比较,如果超时则在表中删除这个 Session ID,当持有这个 Session 的 Client 再次访问时因为 Cluster Server 在 Session Table 里无法查找到相应的 Session 则会收到 Cluster Server 抛出 LostSessionExp 异常。

5.5 系统负载平衡能力的实验

本次试验的环境是在同一个局域网内,访问数的权值和响应时间的权值同取1。共有5个 Cluster Server。试验结果见表2。

表2 试验数据

时间片		#1			#2			#3			#4			#5		
S	C	R1	T1	P1	R2	T2	P2	R3	T3	P3	R4	T4	P4	R5	T5	P5
+3		0	50	0.667	0	20	−0.333	0	20	−0.333						
	+1	0	50	−1.118	0	30	−1.471	1	90	2.588						
	+8	2	50	0.167	4	30	0.233	3	20	−0.400						
	+1	2	51	−0.022	4	20	−0.259	4	40	0.282						
	+2	3	50	0.250	5	20	−0.150	4	30	−0.100						
+1		3	50	−0.500	5	20	−0.133	4	20	−0.467	0	310	1.100			
	+1	3	50	0.352	5	20	0.110	4	30	0.008	1	40	−0.549			
+1	+1	3	50	0.113	5	20	0.202	4	20	−0.155	2	40	−0.452	0	110	0.292
	+27	6	50	−0.078	11	20	−0.182	12	30	0.178	10	30	−0.066	2	80	0.149

S 代表 Server,C 代表 Client,R 代表访问数,T 代表响应时间,P 代表优先级。

从以上结果可以看出,系统满足设计的要求,Client 提出的服务请求根据访问量和响应时间均匀分布到不同的 Cluster Server 上。

结束语 虽然在 OMG 的 CORBA 标准里并没有负载平衡的实现架构,但在 CORBA 的标准接口上用户可以自行扩展实现负载平衡服务。当系统的用户众多或有突发的大量访问时,把压力平衡地分布到多个服务器组成的群集上对于系统的健壮性、有效性都是一个大大的提高。

参考文献

1 Valley S. Programmer's Guide Visibroker for Java, Inprise Corporation, Version 4.5
2 http://www.omg.org/gettingstarted/orb_basics.htm
3 Bartlett D. OMG Interface Definition Language — Defining the capabilities of a distributed service, IBM developerWorks, Sep. 2000
4 Malik S. Dynamic Load Balancing in a Network of Workstations, Nov. 2000