学术论文

具有冗余结构的分布式数据库服务器及其负载平衡模型

田俊峰[1,2]，刘玉玲[2]，杜瑞忠[2]
（1. 中国科技大学计算机系，合肥 230037；
2. 河北大学计算机网络技术研究所，河北保定 071002）

[摘要] 服务器冗余技术在解决传统分布式环境的可用性和性能瓶颈问题的同时给系统的管理带来了新的挑战。介绍了一种分布式数据库服务器的构成原理及工作模型，重点讨论了它的冗余关系、基于移动代理的负载平衡模型及其性能分析等。

[关键词] 分布式数据库；冗余；负载平衡；移动代理；性能分析
[中图分类号] TP311 [文献标识码] A [文章编号] 1009－1742（2004）06－0035－08

1 引言

当前许多 MIS（management information system）的开发都是建立在客户机/服务器（client/server）模式基础上的，因为这样不仅开发价格低廉，无需采用专用并行机系统，可以充分利用目前高性能、低价格的微型机，而且还可以充分利用已有的管理系统的数据库资源。

C/S 系统结构具有以下 2 个优点[1]：a. 能集中使用高性能的系统平台；b. 平台与软件之间的互操作性好。C/S 结构由客户机和服务器两部分组成，通常将应用数据库集中存放在服务器，而大量的应用程序存放在客户机上。客户机与服务器之间的通信是通过一些称为“中间件”的程序接口来实现的。这种模式的缺点是[1]，对集中数据库的访问往往容易出现瓶颈拥塞现象，导致数据查询效率降低。为提高处理数据查询的效率，减轻服务器瓶颈对整个网络系统速率的影响，构建分布式数据库环境应该是一个可选择的方案。基于这基本思想在 Windows NT 和 Linux 异构平台上设计、实现了一个分布数据库系统（DDSS，distributed database server system），较详细地介绍了该系统的总体设计思路，并重点讨论了该系统的负载平衡模型。

2 DDSS 中数据库的存储和管理的分布实现

目前，流行的方案是：采用分布式数据库（如 ORACLE、SBASE 等）在通用的局域网上建立分布式管理系统[2,3]。其存在的问题是，其分布管理和控制机制使得可靠性设计和冗余设计的实现较为复杂[3]，且价格昂贵。

DDSS 是在 C/S 模式下实现的一个分布式的信息管理系统，它根据分布式数据库中数据存储的基本规格，将信息管理系统中的数据库划分为超类数据库和子类数据库。超类数据库中存放公共的基本信息，虽然超类数据库的规模一般相对不是很大，但用户对该库的访问频率最高，因此可将其置于主服务器上；子类数据库中存放的是大量的专用数据，例如人事管理信息系统中每个职工的人事档案、业务档案等；学籍管理信息系统中学生的成绩等。专用数据库不仅量大而且响应速度要求高，但其访问频率相对超类数据库低，因此，可让其基本信息继承其超类数据库。DDSS 的逻辑结构模型如图 1 所示。

[收稿日期] 2003－06－12；修改日期 2003－07－28
[基金项目] 河北省自然科学基金资助项目（60091）；河北省产业化基金资助项目（020501）
[作者简介] 田俊峰（1965－），男，河北藁县人，河北大学计算机网络研究所教授

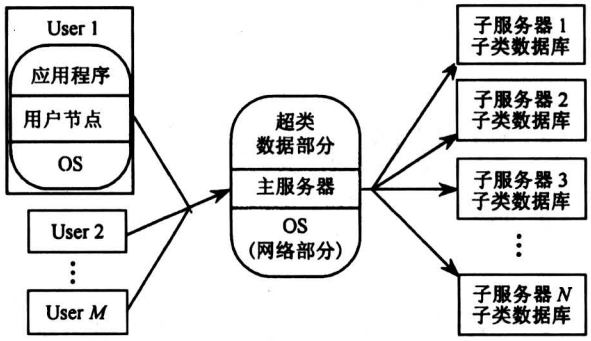


图1 DDSS的逻辑结构模型框图

Fig.1 Logical architecture mode of DDSS

图1中，User为应用程序，OS为各机器上运行的软件，包括操作系统、数据库管理系统DBMS以及各工作站自己使用的应用程序。在C/S模式中，客户机运行那些使用户能阐明其服务请求的程序，并将这些请求传送到主服务器。

3 系统的运行模型

3.1 系统初始化

系统初始化时，主服务器首先生成一张服务器系统的结构映射表，如下：

```
Structure Map_table_node
{
int server_number;              /服务器编号（标识）/
char IP_address;                /服务器的IP地址/
int information_type;           /信息类型，与数据库相关/
int backup_server_number        /备份数据库编号/
int status;                     /该数据库的当前状态/
}; Map_table [N];
```

该表内容包括：a. 服务器系统规模 N（服务器节点个数）；b. 数据库分配情况，即哪种类型数据分配在哪个服务器（主管）上；c. 数据备份关系，服务冗余关系；d. 各服务器的IP地址；e. 服务器当前状态，“00”表示该服务器处于正常响应状态，“01”表示其处于备份状态，“10”表示其处于保留状态，“11”表示其处于脱线状态。

然后创建通信消息队列（接收和发送），将通信中间件调入主存。最后与各子服务器进行握手，以确定其目前的状态并修改映射表，该握手操作在系统处于正常响应态时是周期性进行的，即主服务器要定期地了解各子服务器的状态，动态地更新映射表，同时以广播方式发送给客户端，以保证系统可靠运行。各客户端初始工作时，需要向主服务器索要结构映射表，并随时接收主服务器的更新信息。该系统的结构模型形式描述如下：

1）服务器 Server :: = ＜MS, SS＞，其中，MS（Main Server）为主服务器的集合，SS（Sub-Server）为子服务器的集合。

$MS = \{MS_i \mid 1 \leqslant i \leqslant 2\}$；

MS_i:: = ＜Name, Database－Object, State＞；

$SS = \{SS_i \mid 1 \leqslant i \leqslant N-1\}$，$N-1$ 为可同时工作的子服务器的个数；

SS_i:: = ＜Name, Database－Object, State＞；

其中，Name为服务器的名字，是服务器的唯一标识，Database－Object为数据库（连带服务）对象，State为服务器的状态。

2）数据库对象 Database－Object $= \{DO_i \mid 1 \leqslant i \leqslant m\}$，Database为在某一服务器上可提供的数据库对象及相应服务的集合，m 为对象个数，且 $1 \leqslant m \leqslant N$，$DO_1$ 为该服务器的固有数据对象，而 DO_2，DO_3，…，DO_m 为其他服务器之固有数据对象的备份。

DO_i:: = ⟨DO－Name, Attributes, Access⟩，i 为数据对象序号，DO-Name为其唯一标识，Attributes为数据对象的属性，Access是数据对象当前的被访问情况，Access = (0, 1)，0表示未被占用，可访问；1表示正在被占用，访问被阻塞。

3）状态 State :: = ⟨0, 1⟩，0表示服务器未在线（子服务器），或处于备份状态（主服务器）；1表示服务器处于正常响应态。

4）请求 Request :: = ⟨REQ_1, REQ_2⟩，REQ_1 表示具有写属性的操作的集合，如修改、创建、删除、追加等；REQ_2 表示具有读属性的操作，如检索、统计等。

3.2 运行模型

正常运行时，客户端发出操作请求，服务器系统响应请求，首先对其访问操作的属性进行解析，然后其根据请求的不同，选择以下2种模式之一进行处理。

1）对于读请求——不修改数据库内容或结构的请求

读请求分2种情况处理：a. 如果读操作只涉及到某个子服务器或主服务器，那么，请求直接发给该服务器，并由其返回结果；b. 如果读操作涉及到多个服务器，则将请求发给主服务器，主服务器根据操作需求查找映射表，以定位所访问的数据

库对象所在的子服务器，并交由这些子服务器分别进行相应的操作，如某子服务器忙，则选择其备份服务器集中空闲者完成操作，操作完成后将结果返回主服务器，主服务器按用户要求将结果汇总后再返回客户端。

2) 对于写请求，操作完成后要修改数据库的记录或结构。

主服务器根据操作数的属性查找映射表，以定位所访问的数据库对象所在的子服务器，并交由该子服务器进行相应的操作，如该服务器忙，则选择其备份服务器集中空闲者完成操作，操作完成后由子服务器将结果返回客户端的同时也交给主服务器，由主服务器负责对其备份数据进行修改，以维护数据的一致性。

4　DDSS 的负载平衡模型

分布式对象技术是在分布计算环境中对象技术和客户/服务器计算模式的集成，它改变了以往面向多机系统的进程和主机的计算模型，更适合基于客户/服务器模式的分布计算环境[4]。采用分布式对象技术可以有效的描述和解决分布式系统中如冗余服务定义、负载平衡等问题。譬如，冗余服务应具有 2 个特性：一是等效性，即无论冗余服务是如何实现的，它们所提供的功能都应该相同；二是从属性，等效的服务并不一定是冗余服务，只有属于同一目标环境的服务才是冗余服务。冗余服务的等效性可以用服务接口的一致性来描述，但从属性很难确切地定义。利用对象和实例的概念，对冗余服务的特性进行定义。

1) *主动对象*　采用主动对象描述分布计算环境，具有两层含义：一是不但能够向用户提供状态信息，还能提供计算服务；二是不仅能接收客户的请求，还能调用其他对象的服务，即实现对象的嵌套调用。

2) *利用请求来划分客户和服务*　由于分布式对象数据库、事件、方法封装在一起，对外提供统一的调用接口，引进分布式对象后，将分布计算环境中各种独立存在的服务资源作为对象处理，有利于对各种资源的统一管理，也有利于对分布式计算环境中的冗余服务进行合理的描述，所以可以用请求来对客户和服务进行划分：当一个对象向其他对象发出请求时，它是客户；当一个对象接收其他对象的请求时，它就成为服务器。这样，就可以把分布式计算环境中的客户/服务器模型抽象成请求和对象之间的关系。

由此，DDSS 的冗余服务便可如图 2 所示，被描述为实例池与请求池之间的分配与调度问题，即如何将一个申请某类服务的请求发送到相应服务对象的实例池上，以及如何在实例池中选择适当的实例响应该请求。

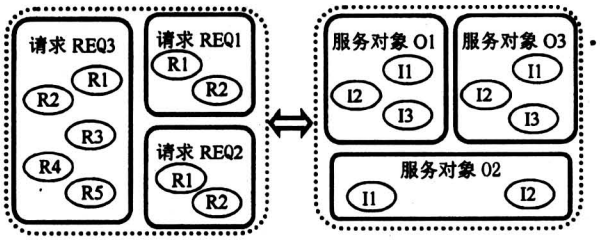


图 2　请求池与实例池

Fig.2　Request pool and instance pool

4.1　系统框架描述

基于上面的讨论，可从 DDSS 冗余服务系统中抽取两类被管理的资源对象：一类是硬件资源（主机 HOST)；另一类是软件资源（服务对象和对象实例)。进而其系统框架可抽象为一个 3 层管理模型，由 Administrator, Manager, Agent 3 部分组成，如图 3 所示。

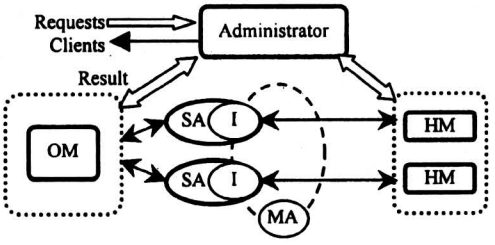


图 3　DDSS 冗余服务系统管理框架

Fig.3　Management framework of redundant servers based on DDSS

1) Administrator 是系统中所有 Manager 的管理者，负责收集和维护全局的管理信息以及全局的调度工作；

2) Manager 包括 OM（即 object manager）和 HM（即 host manager）2 类，它们既是 Administrator 的被管理对象，同时又是 Agent 的管理者。OM 为一类冗余服务对象提供管理接口，集中管理某一特定服务类的冗余服务；HM 则驻留在各个主机上，为主机提供管理接口；

3) Agent 包括服务代理 SA（server agent）和移动代理 MA（mobile agent）。SA 在服务对象实例上增加了管理接口，辅助 Manager 实现管理功能；MA 在冗余服务对象之间按一定的规程移动，实现负载信息收集、负载平衡等管理功能。

4.2 基于 MA 的负载平衡模型

4.2.1 MA 的体系结构

1) MA 的功能 移动代理（MA）的功能主要是收集所到达的服务对象实例的负载及状态信息，并根据服务对象实例的当前状况与 SA 一道协同 OM 完成负载平衡任务。

2) MA 的组成 每个生命周期中的 MA 由三部分组成：移动支持模块、方法集和状态集。其中，移动支持模块负责完成 MA 的身份认证、MA 的自主移动以及同其执行环境的通信；方法集，即 MA 的任务集，提供用于收集服务对象实例的当前负载及相关状态信息的方法；状态集则包含服务对象实例的负载信息状态表和 MA 自身的移动规程。MA 的体系结构如图 4 所示。

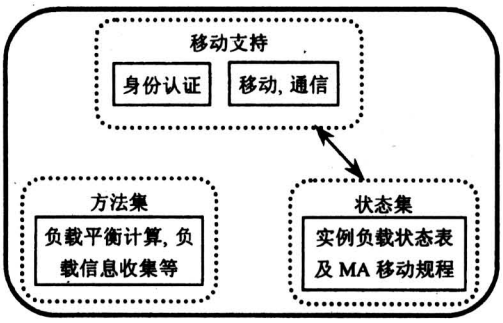


图 4 MA 的体系结构

Fig.4 Architecture of MA

3) MA 的生命周期 移动代理的生命周期可以分为 3 个阶段：创建阶段、运行阶段和消亡阶段。

a. 创建阶段在 OM 处完成，主要包括移动规程的确定；

b. 运行阶段是指移动代理在由若干服务代理 SA 组成的逻辑环上持续迁移并收集负载信息的过程；

c. 消亡阶段则包含从移动代理确定应该返回 OM 开始，直至返回 OM 后汇报完毕收集到的所有负载信息并被 OM 回收为止。

4) MA 的迁移规程 系统中每一个移动代理实体都是由 OM 创建的，并且都具有各自不同的迁移规程。迁移规程与以下 3 方面的因素相关：

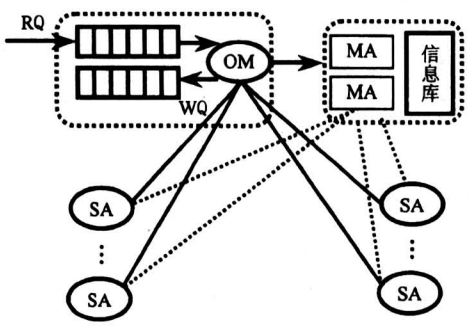
a. 系统当前所允许的请求冗余度 n，它决定着迁移规程的预期长度。

b. 所有服务对象实例的当前状态，它则决定着迁移规程的实际长度。当所有冗余服务的对象实例都处于重载状态时，就没有可以参与冗余服务的对象实例，OM 会延缓此次请求的分发，相应的移动代理也不会被创建；当有足够服务对象实例的当前状态是处于轻载时，OM 就会根据负载索引的大小将冗余服务的对象实例进行排队，并从其中选出 n 个状态最优者与 OM 共同组成相应移动代理实体的移动规程；当没有足够的服务对象实例时，OM 会根据具体情况决定是否继续分发请求。如果决定继续，那么相应移动代理的迁移规程的实际长度会小于等于 n。

c. 在移动代理实体迁移过程中，如果遇到故障或失效节点，会自动调整，并选择新的目标节点。同时，将故障或失效信息收集起来。

4.2.2 负载平衡模型 MMA

负载平衡模型 MMA（object manager-mobile agent-server agent）是 OM，Mobile Agent，SA 三者的逻辑交互模型，如图 5 所示。其中，OM 的职责是创建 MA、启动 MA 及回收 MA；MA 一旦进入它的生命周期，便开始按照预定的移动规程和方法参与系统负载平衡工作；SA 负责为到访的 MA 提供相应的执行环境，包括 MA 的接收、初始化及本地化、为 MA 的执行和迁移提供完备的方法等。三者相互配合，协同完成系统中的负载平衡任务。



RQ 为请求等待响应；WQ 为已被响应的请求等待返送结果

图 5 负载平衡模型 MMA 送逻辑结构模型

Fig.5 Logical architecture of load balance model on MMA

如图 5 所示，每个客户请求的达到都会促使 OM 创建一个 MA。MA 在按照自己的迁移规程进行迁移的同时，还承担着收集服务对象实例的负载信息以及进行平衡负载的双重工作。每个 MA 实体都记录着唯一的请求标识，只有当确定对应的请求已

被执行完毕且 MA 已经遍历过迁移规程包含的所有 SA 一次之后，MA 才能够携带最新负载信息最终返回 OM，这也意味着它的生命周期即将结束。

由于 MA 本身特有的移动性与智能性，使得原来由 OM 负责完成的负载平衡工作可以交给特别订制的 MA 及 SA 来协作完成，这样，从很大程度上缓解了 OM 的工作压力，同时也降低了 OM 成为系统瓶颈的可能性。

系统的负载平衡从 2 方面得以保证：

a. OM 按照实例信息列表中记录的实例对象的负载索引，主机的负载及实例的工作成绩将实例进行排序，根据系统负载计算得出当前所允许的请求冗余度 n，从排好顺序的实例列表中选择出 n 个实例作为请求分发的对象，同时构成 MA 的迁移规程；

b. MA 按照迁移规程依次访问 SA，当 MA 到达 SA 后，首先对当前实例信息进行收集，然后根据自己已经遍历过的实例对象的工作情况，从当前 SA 的请求队列中删除已经执行完毕的请求，及时调整当前实例的负载，在利用冗余实例保证系统性能的同时，充分利用系统资源。

1) *服务代理 SA 的结构*　由于服务管理者 OM 的具体工作通过 MA 与 SA 协作完成，所以 SA 应该为来访 MA 提供一定的执行环境，从这个角度来讲，SA 不再是简单的服务代理，它的功能应该包含 2 个方面：一方面，SA 要为服务管理者 OM 提供关于服务对象实例的管理接口；另一方面，SA 还应该能够支持来访 MA 的执行，其中包括 MA 的接收、MA 的初始化及本地化、MA 的发送。其结构应如图 6 所示。

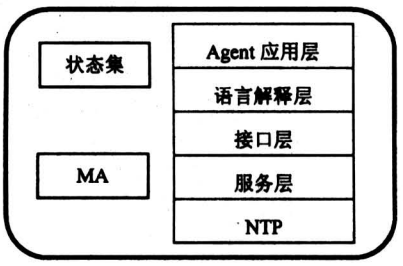


图 6　服务代理结构

Fig.6　Service-agent architecture

a. 网络传送层（NTP）　是与现有网络通信协议的接口，SA 通过 NTP 可以与其他 SA 进行通信，而且 MA 的传递和接收也是在 NTP 层完成的；

b. 服务层　为 MA 建立运行环境和安全保护机制，协调和监视各个 Agent 的运行；

c. 接口层　为 MA 与宿主机及其他 MA 之间的通信提供底层的界面；

d. 语言解释器　MA 是由跨平台语言（如 Java 等）实现的，SA 接收到每一个 MA，都要在所建立的相应的语言解释器的支持下工作；

e. Agent 应用层　MA 在以上各层的支持下最终完成自己的调度任务。在这一层可以包含多个 MA；

2) *负载索引的定义*　负载索引通常被定义为衡量主机或服务负载情况的标准[5]。衡量主机负载的参数包括：驻留在主机上的进程个数，当前正在运行的进程个数，正在等待的进程个数，以及 CPU 队列长度、某段时间内的平均 CPU 队列长度、CPU 利用率等。本系统采用驻留在主机上的所有 SA 请求队列的长度之和来描述主机的负载情况。

另外，为衡量服务对象实例最近时期的服务情况，定义一个新参数——实例的工作成绩，保存于各个 SA 上。每个周期开始之前，SA 都要将本周期内服务对象实例完成的任务量保存下来。在一个周期之内，服务对象实例每执行完毕一次请求，都要对其工作成绩进行适当的增加。这个增量，是在同时考虑到该服务对象实例所属计算机的处理能力和该服务对象实例本身的处理能力的前提下确定出来的。因此，将一个服务对象实例在上一个周期的工作成绩作为负载索引提供给 OM，既可以反映出服务对象实例本身的处理能力，也可以反映出异构情况下节点的计算能力对服务对象实例的影响。

3) *负载信息的收集*　负载信息的收集方式为周期性的推取方式，由 MA 的定时自我复制功能来实现。OM 根据不同类型请求的执行时间（Execute_Time）的差异，为每个 MA 设计不同的复制周期，即 OM 的负载信息收集周期。当 MA 进入生命周期后，按照预定规程迁移遍历参与冗余服务的若干 SA 时，将对相关信息进行收集，同时在每个收集周期末进行自我复制，携带收集到的所有信息返回 OM。

4.3　MMA 模型的实现

4.3.1　MA 的实现　整个冗余服务系统选用 JAVA JDK 编程环境实现[5~7]。

MA 统一由其自身的 ExecuteOnArrival（Object_Name）方法来实现。

1）请求标识（Req_Id） 由于每一个请求的到达都会促使OM创建一个新的MA，所以将相关的这个请求标识Req_Id记录在Mobile Agent中。当该请求被执行完毕时，就标志着相应MA的生命周期即将结束，即相关请求是否执行完毕这一信息是MA是否继续运行的决定因素之一。

2）实例信息列表（Instance_List） 存储服务对象实例的相关信息，MA需要实现一张Instance_List表，每个表项中包含着下列信息：

a. OM_Id，服务对象实例所属的对象管理者OM的标识；

b. Hm_Id，服务对象实例所驻留的主机管理者HM的标识；

c. Instance_Id，服务对象实例的标识；

d. Host_Ip_Addr，服务对象实例所驻留的主机的IP地址；

e. Port，服务对象实例所驻留的主机分配用于监听MA的端口号；

f. Max_Load，服务对象实例所驻留主机能承受的最大负载值；

g. Status，服务对象实例的状态，即对象实例驻留的主机的状态。分为Normal和Over_Load两种；

h. Load_Index，服务对象实例的负载索引值；

i. Current_Load，服务对象实例的当前负载值；

j. Execute_Time，服务对象实例执行一个单位任务的平均时间。

3）已执行完毕的请求队列（EndedReq_Queue） EndedReq_Queue队列中存储的是当前服务对象实例已经执行完毕的请求，数据完全来源于服务对象实例的EndedReq_Queue。在MA到达下一个服务对象实例时，根据自身EndedReq_Queue队列中存储的信息扫描当前服务对象实例的请求队列Request_Queue，如果发现其中含有由其他实例已经执行完毕的请求，则立刻将其从当前服务对象实例的请求队列中删除，从而避免由任务的重复执行而造成的资源浪费。

4）移动规程（Move_Line） Move_Line是MA的移动规程，其中存储着在它的生命周期中预定的整个移动路线。每个节点的信息包括5部分：

a. Node_Id节点的标识有两种可能，一是当MA返回OM时，Node_Id应是相应的OM_Id；二是当MA在SA的逻辑环中移动时，Node_Id则应是相应的Instance_Id；

b. Node_Ip节点的IP地址和Node_Id一样也有相应的两种情况；

c. Port节点用于监听Mobile Agent的端口号；

d. Hm_Id节点所属主机管理者的标识；

e. OM_Id节点所属对象管理者的标识。

4.3.2 执行方法ExecuteOnArrival（Object_Name） ExecuteOnArrival（Object_Name）是一个MA实体移动到某个服务对象实例所对应的节点后的主要计算工作，包括如下几个步骤：

a. 将MA的EndedReq_Queue中记录的已经执行完毕的请求从当前服务对象实例的Request_Queue中删除；

b. 将当前服务对象实例的EndedReq_Queue中的元素复制到MA的EndedReq_Queue中，并将当前服务对象实例的EndedReq_Queue中与MA的Req_Id一致的请求记录删除；

c. 收集当前服务对象实例的相关信息；

d. 如果发现与自己相关的请求（由Req_Id标识）已经执行完毕，再进一步判断MA当前是否迁移到最后一个服务对象实例。如果是，Mobile Agent则返回OM。否则，MA将继续开始新一轮的迁移；如果与自己相关的请求尚未有结果返回，MA将调用本地方法DispatchAgent（Movenode，Mobile Agent_Id，Reportflag）（见SA的实现）继续向下一个节点迁移。

4.3.3 服务代理SA的实现

1）为OM，HM提供管理接口

a. 相关的状态集合 为配合OM，HM实现对服务对象实例的管理工作，服务代理SA需要为每个服务对象实例配置下列状态，即这部分的状态集如下定义：

StateSet_ForOMHm：=〈Instance_Capbility，Current_Load，History_Mark，Instance_Item，Excute_Time，Service_Type，Load_Index，Init_Num，Request_Queue，EndedReq_Queue，OMImpl，HmImpl〉。

其中，Instance_Capbility是服务对象实例处理请求的能力；Current_Load是服务对象实例所驻留主机的当前负载值；History_Mark是服务对象实例上1周期内完成的任务数量；Instance_Item是服务对象实例的信息集合；Excute_Time是服务

对象实例完成单位任务量所需要的执行时间；Service_Type 是服务对象实例能够完成的服务类型；Load_Index 是服务对象实例的负载索引值；Init_Num 是服务对象实例的序号，用来构成对象标识；Request_Queue 是服务对象实例的请求队列；EndedReq_Queue 是服务对象实例已经执行完毕的请求的队列；OMImpl 是关于所属 OM 的对象引用；HmImpl 是关于所属 HM 的对象引用。

b. 相关方法集合　为配合 OM、HM 实现对服务对象实例的管理工作，服务代理 SA 需要提供下列方法，即这部分的方法集如下定义：

MethodSet_ForOMHm：＝〈SetStatus，GetItem，ChangeOM，DeleteSA，ReceiveReq，RegisterToHM，RegisterToOM〉。

其中，SetStatu 是改变服务对象实例的状态；GetItem 是为 OM，HM 提供本地服务对象实例的信息集合；CurrentLoad 是服务对象实例的当前负载值；ChangeOM 是 OM 发生切换时向 SA 进行通知；DeleteSA 是删除本地服务对象实例；ReceiveReq 是服务对象实例接收由 OM 分配的请求；RegisterToHM，RegisterToOM 方法分别负责完成服务对象实例向 HM，OM 的注册。

2）为 Mobile Agent 提供执行环境

a. MA 的接收、初始化及本地化　SA 的内部设计了一个线程类 ReceiveAgent，内设一个 Agent_Queue。该线程在 SA 启动的初始同时被启动，负责监听和接收到访的移动代理，在对其进行完毕初始化和本地化的工作之后开始执行。主要在 ReceiveAgent（Sa_Name，Port），AddAgent（Agent）内部方法和 ReceiveAgent 线程自身的 run（）方法中实现。

b. MA 的执行和迁移　MA 经过本地化后，便可调用本地服务代理为其提供的计算资源开始计算工作。SA 为其提供的方法主要有：

GetItem 为 MA 提供本地服务对象实例的信息集合；RemoveReq(Req_Id)为删除本地服务对象实例的请求队列中的请求；RemoveEndedreq(Req_Id)为删除本地服务对象实例的已执行完毕请求队列的请求；IsEndedReq_queueEmpty 为判断本地服务对象实例的 EndedReq_Queue 是否为空；IsWorkingAgent_QueueEmpty 为判断本地服务对象实例的 Agent_Queue 是否为空；DispatchAgent（Movenode，Mobileagent_Id，Reportflag）为将标识为 Mobileagent_Id 的 Mobile Agent 发送到下一个节点。

3）对本地服务对象的管理　SA 对本地服务对象实例的管理主要由以下 2 个方法和 1 个线程来完成。

WorkOver 负责在实例对象完成一个请求之后对部分信息进行调整并主动通知 HM；BindInstance 负责完成 SA 自身与本地服务对象实例的绑定。线程 ModifyMArk 负责定期将实例的工作成绩（History_MArk）清零，重新记录。

需要特别指出的是，在具体的实现过程中，SA 被设计成为一个线程类。这样实现，不仅在很大程度上简化了 SA 被启动和被删除的过程，而且符合服务实例对象随时等待请求到达的特点。当服务对象实例接收到一个请求时，SA 就会根据自己的服务类型进行一次实例的绑定，并启动实例对象执行请求。

4.4　性能分析

如前所述，由于在负载索引的定义中引入了工作成绩的概念使得负载轻重的描述更加精确，也更接近实际，因此负载平衡工作也就更有效。

由于收集负载信息的代价主要是网络通信的开销，因此下面就以此为主要指标与传统的推拉工作模式进行比较分析。

设某服务对象实例的冗余度为 k，则 MMA 的工作模式如图 7 所示，而传统的推拉工作模式[4]如图 8 所示。假设系统中任意两结点间的通信开销相同，设为 Cost。MMA 模式下收集所有 SA 负载信息的通信开销为：$C_M = \text{Cost} \times (k+1)$；推拉模式下通信开销为：$C_P = \text{Cost} \times 2k$。显然有 $C_M \leqslant C_P$。

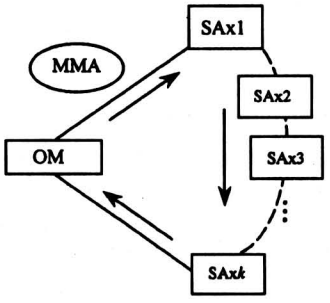


图 8　MMA 工作模式

Fig.8　MMA operating model

5　结语

DDSS 的设计技术已经被成功地应用到了一些

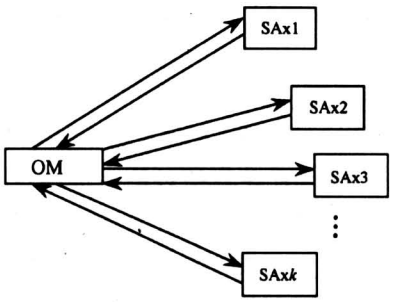


图9 推拉工作模式

Fig.9 Push-pull operating model

大数据量、多数据类型的信息管理系统以及BBS服务器等应用项目的开发中。经试用及分析均表明，与目前多数用户使用的的高端服务器和分布式服务器[2,3]相比具有性能价格比高、可扩展性好、并行性好、可靠性高、资源配置合理等优点。

参考文献

[1] 田俊峰，王凤先. 一种基于C/S模式的分布式数据库服务系统及其排队模型 [J]. 计算机工程与科学，2002，24(2)：88～91

[2] Chang W, sheikholeslami G, Zhang A. Efficient resource selection in distributed visual information systems [A]. ACM Multimedia '97 Electronic Proceedings, No 8～14 [C]. 1997

[3] Kahle B, Medlar A. An information system for corporate users: wide area information servers [J]. Conations-The Interoperability Report,1991, 5(11): 2～9

[4] 钱 方，邹 鹏，陈 渝，等. 基于分布对象的冗余服务模型 [J]. 计算机研究与发展，1999，36(11)：1392～1393

[5] 傅 强，郑伟民. 一种适用于机群系统的任务动态调度方法 [J]. 软件学报，1999，10(1)：19～23

[6] 陈志刚，李 登，曾志文. 基于Java RMI技术的中间应用服务器动态负载均衡模型的实现 [J]. 计算机工程，2001，27(7)：48～50

[7] 蔡洪波，张大方，谢高岗. Mobile Agent选择移动迁移机制的设计与实现 [J]. 计算机工程与科学，2001，23(5)：54～56

Model of a Distributed Database Server With Redundant Structure and Its Load Balance

Tian Junfeng[1,2], Liu Yuling[2], Du Ruizhong[2]

(1. *Department of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027, *China*; 2. *Institute of Computer Network Technology, Hebei University, Baoding, Hebei* 071002, *China*)

[**Abstract**] The technology of redundant services has improved the system availability, but also poses new challenges to the distributed system configuration management. This paper introduces the model of structure and work for a distributed database server system, and discussions are focused on its redundancy relations, model of load balancing based on mobile agent and performance analysis.

[**Key words**] distributed database; redundancy; load balance; mobile agent; performance analysis