第 33 卷 第 2 期 计 算 机 工 程 2007 年 1 月
Vol.33 No.2 Computer Engineering January 2007
·软件技术与数据库· 文章编号：1000—3428(2007)02—0085—03 文献标识码：A 中图分类号：TP311

大规模数据库系统中的面向服务架构研究

李 慧[1]，宋怀明[1,2]，焦丽梅[1,2]，刘 莹[1,2]，王 洋[1,2]，王启荣[1]

(1. 中国科学院计算技术研究所，北京 100080；2. 中国科学院研究生院，北京 100039)

摘 要：为了满足复杂的大规模数据库系统所要求的简单、高效、高可靠的需求，该文定义了面向服务的架构。该架构把繁杂多变的系统模块分为多个服务模块，每个模块独立实现其功能，介绍了面向服务的架构如何处理服务间的协调、部署、通信和高可用方案，使诸多松散耦合的服务保持统一的管理和彼此间的联系，面向服务的架构大大简化了复杂的大规模数据库系统，并使整个系统的结构清晰化。

关键词：大规模数据库系统；面向服务的架构；互联网通信引擎

Study of Service-oriented Architecture in Large-scale Database System

LI Hui[1], SONG Huaiming[1,2], JIAO Limei[1,2], LIU Ying[1,2], WANG Yang[1,2], WANG Qirong[1]

(1. Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080;
2. Graduate School of Chinese Academy of Sciences, Beijing 100039)

【Abstract】 To satisfy the need of simple, efficient and high-available large-scale database system. This paper defines service-oriented architecture, which divides the complicated system into many service modules. Each of the modules realizes its functions independently. It also introduces how the architecture achieves service harmonization, deployment, communication and high-availability to keep uniform management and contact among these loosely coupled services. Service-oriented architecture simplifies large-scale database system and makes the system architecture much clearer.

【Key words】 Large-scale database system; Service-oriented architecture(SOA); Internet communications engine (ICE)

随着计算机技术的发展，应用系统的规模增大，系统开发越来越成为是一个纷繁复杂的过程。面向服务的架构的思想是以服务层为基础，应用层通过网络对服务直接调用，来实现分布在不同节点系统中服务的共享与重用，从而简化系统的开发，方便系统的管理。面向服务架构的一个基本前提是服务使用者与提供者、服务与服务之间松散耦合。本文以大规模数据库系统为背景，介绍面向服务的架构思想在真实系统中的实现，主要解决如何对松散耦合的服务进行协调、部署以及通信的问题。

1 大规模数据库系统介绍

1.1 系统结构

本文所提及的大规模数据库系统的基本功能是为用户实现数据加载、查询、管理和备份等服务。此系统的特点是数据量庞大，操作频繁。系统在线数据库容量为 59.035TB，要求能够持续完成每分钟写入约 700 万个数据记录，一天 10GB 的数据记录，一年 3.6TB 的数据记录。同样，查询、管理、备份都有比较重的负载。

根据用户需求，系统设计了 18 台节点存放数据库数据，8 台加载节点以提供加载服务，4 台查询节点，2 台管理结点以及 1 台备份节点。数据库系统节点繁多，是一套初具规模的分布式系统，它的结构如图 1 所示：最上层为客户端程序接口，它为客户端提供访问并行数据库的通道；最下层由若干独立执行的数据库系统组成，负责具体的数据加载存储、查询和管理。并行数据库中间层(虚线框所示)是位于客户与数据库之间的一层软件，对上要提供客户端程序的接口；对下要提供多个并行数据库系统的数据加载、数据查询、数据备份、数据统计和管理功能。本中间层由若干服务中间件组成，在为客户端程序提供的接口中，它屏蔽数据分布存储和请求的分布执行，为客户提供一个单一的运行接口和环境；在对数据库进行工作时，它要协调多数据库服务器的数据分布和协同工作[1]。

1.2 服务管理中的关键问题

系统的中间层部分为客户提供了加载、查询、管理、控制、统计等服务，这些服务分布在 15 个节点中。有些服务涉及多个节点，例如，有 8 台加载节点都能执行加载服务，这给客户端带来一个困惑，如何自动地去寻找一个可用的加载服务？客户端可以通过轮询的方式查找可用的服务，但这需要同服务器大量的交互，这种情况下客户端与服务器部分的耦合性很大，而且在多个客户用相同策略请求服务的时候会出现负载倾斜。作为大规模的数据库系统，事务的稳定性问题非常重要。一个单独事务中包含的多段细粒度请求可能使事务处理时间过长、导致后台服务超时，从而中止[2]。

理想情况下，松散耦合的服务使用者和服务层之间不必进行多次的往复，而是一次往复就足够得到合适的服务。如何解除服务使用者与服务提供者之间的紧密耦合，这是中间件服务管理部分(图 1 中灰色框部分)要解决的首要问题。松散耦合的服务结构除了表现在服务使用者与服务提供者之间的接口外，还体现在不同服务之间。该设计中的服务指的是“单独的”、“独立的”、“封装完善的”服务，它体现了系统中的多个加载、查询、管理、控制、统计服务模块之间的关

作者简介：李 慧(1980－)，女，硕士，主研方向：应用系统及其评价；宋怀明，博士生；焦丽梅，博士生、副研究员；刘 莹、王 洋，博士生；王启荣，硕士

收稿日期：2006-01-23 **E-mail**：lihui@ncic.ac.cn

系。这样方便了各模块的开发人员，同时也为管理员提供了一种通过直接管理松散耦合服务，有针对性的优化业务流程的方式。

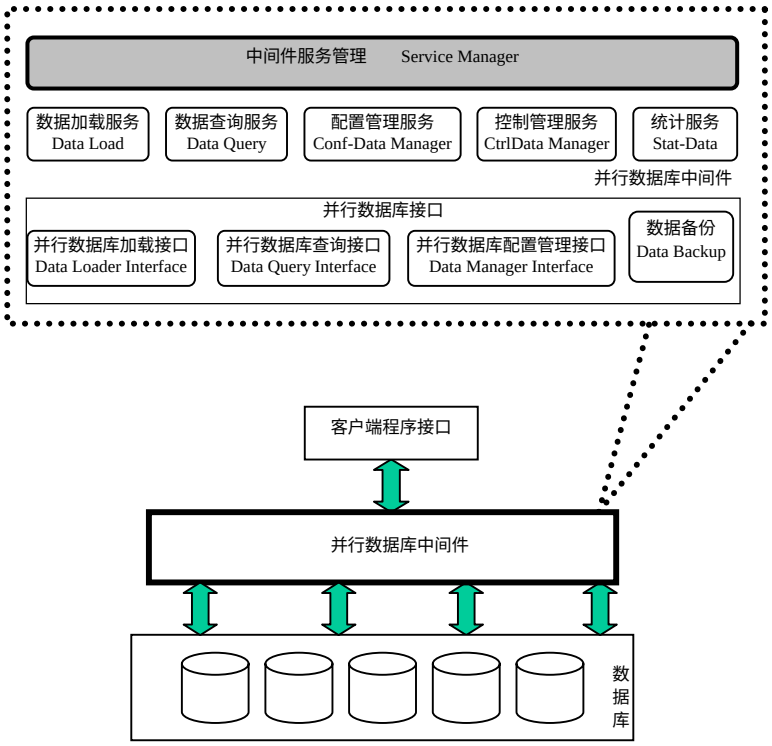

图 1 并行数据库系统软件结构

松散耦合的服务结构也带来了几个的技术上的新问题。如何协调松散的服务，维护负载平衡并使客户在短时间内找到服务？如何管理、部署服务，从而维护共享数据的一致性？如何屏蔽各服务所在平台、语言和协议的影响，实现两个独立的服务之间的通信？针对上述 3 个问题，本系统引用了面向服务的架构。

2 面向服务的架构简介

面向服务的架构(Service-oriented Architecture,SOA)的应用已有多年的历史，IBM CICS 和 BEA TUXEDO 就是过去被用于构建 SOA 应用的两种技术范例。在近期的企业级应用开发领域，SOA 的研究非常多。下面首先给出服务和面向服务的架构 SOA 的定义。在 W3C 中，服务的定义：

定义 1 服务就是服务提供者完成一组工作，为服务使用者交付所需的最终结果。最终结果通常会使使用者的状态发生变化，但也可能使提供者的状态改变，或者双方都产生变化。

对于 SOA，Service-architecture.com 如下定义：

定义 2 SOA 在本质上是服务的集合。服务间彼此通信，这种通信可能是简单的数据传送，也可能是两个或更多的服务协调进行某些活动。服务间需要某些方法进行连接。所谓服务就是精确定义、封装完善、独立于其它服务所处环境和状态的函数。

Looselycoupled.com 中 SOA 的定义为：

定义 3 SOA 是按需连接资源的系统。在 SOA 中，资源被作为可通过标准方式访问的独立服务，提供给网络中的其他成员。与传统的系统结构相比，SOA 规定了资源间更为灵活的松散耦合关系。

不同厂商对 SOA 有着不同的理解，但他们都在强调 SOA 的几个关键特性：一种粗粒度、松耦合服务架构，服务之间通过简单、精确定义接口进行通信，不涉及底层编程接口和通信模型。

本系统将要解决的 3 个问题与 SOA 的特性结合，设计了面向服务的系统架构。

3 架构在系统中的应用

本系统专门设计一个中心节点，用来实现面向服务的架构的服务管理功能。对应于 SOA 的关键特性以及本文 1.2 节最后提出的 3 个问题，面向服务的架构在本系统实现时有 3 个关键技术：服务的协调，服务部署和服务通信。

3.1 服务的使用与协调

为了使客户在短时间内准确地定位到松散耦合的服务，同时还维护负载平衡，系统在服务管理节点上设计了注册、定位服务的功能。

系统启动时，系统的每个服务都在管理节点注册，管理节点把服务器的相关信息都存在于它自身的数据库中。当客户发出请求时，各服务器无需再运行，管理节点会在第 1 个客户到达时，随需启动其中一个服务器。随着客户到达的数量越来越多，管理节点查找它自己数据库中可用服务器的个数，根据一定的策略自动控制均衡的任务分配。管理节点还负责协调服务器的端口，服务器的代码中没有定制的服务端口，每个客户也无需事先知道服务的地址和端口，而只需要知道管理节点的地址和端口。具体流程如图 2 所示。

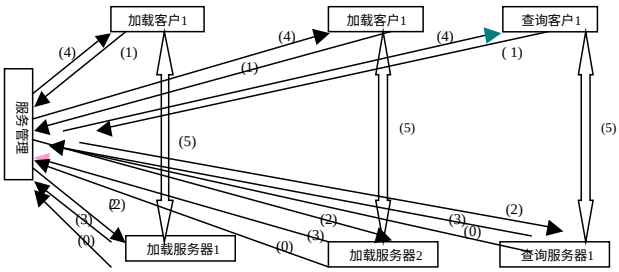

图 2 客户通过服务管理节点被服务的过程

图 2 中，0 表示每个服务在服务管理器上注册。当客户需要服务时，它进行下列动作：

(1)客户首先到管理节点申请服务。

(2)其中一个服务器通过管理节点发出信息感到客户的请求，然后启动。

(3)管理节点获知可用服务器的地址和端口。

(4)管理节点把可用服务器的地址和端口传给客户。

(5)客户端用新得到的地址和端口开始被服务的过程。

根据定位服务的算法设计，每个客户到来时都会得到一个可用的服务，而统一的定位策略又使得服务器之间的负载平衡得以保证。除此之外，这种设计的优点还表现在以下两个方面：(1)客户端不需要事先得知服务器的地址和端口，减少了服务使用者和服务提供者的耦合性。(2)服务器可移植性强，它可以随意变换地址和代码，只需在管理节点上重新注册一下即可。

3.2 统一的服务部署接口

服务管理节点是整个面向服务架构的核心，除了掌管服务注册、定位功能外，它还统一管理服务的配置、部署等事宜。

本系统中的各服务模块由于业务的特殊性，都要访问 18

个底层数据库节点，若各模块在本地存有底层数据库名和口令，这种做法不利于维护，而且有悖于数据库安全性。作为系统的核心，服务管理节点负责管理并推送这些配置参数。这项任务是在部署服务阶段完成。

部署意味着为某个应用配置注册域。部署涉及到服务添加到注册表中并创建带有服务配置参数的服务器配置文件。本系统采用一种数据驱动的部署服务，其中的描述符用 XML 编写。服务所需的属性都被远程部署在 XML 文件里，例如配置参数数据库名、用户名、口令可通过如下方式进行部署：

在服务管理节点的部署文件 app1.xml 里，有：

```
<property name="tnsname" value="dbnode01"/>
<property name="username" value="system"/>
<property name="passwd" value="manager"/>
```

如图 3 所示，虚线箭头表示各服务器在服务管理节点上注册产生各自的注册数据；虚线部分表示服务部署的整个流程。服务管理员通过服务部署接口把 app1.xml 传给服务管理节点，服务管理节点根据 app1.xml 中所描述的信息转化为每个服务器的配置文件，当服务启动时，加载、查询等各服务模块可通过下列远程的方式获取配置文件参数：

```
string tnsname = properties->getProperty("tnsname")
string username = properties->getProperty("username")
string passwd = properties->getProperty("passwd")
```

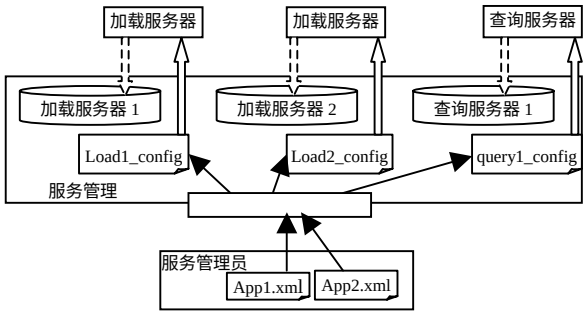

图 3 服务管理员部署服务

本系统部署方案设计具有以下优点：(1)实行服务配置的统一管理，有利于配置数据安全，有利于维护配置数据一致性。(2)采用 XML 文件格式进行部署，通过服务管理节点推送参数的方式，屏蔽了语言层和协议层的影响，简化了服务器端获取参数的编码，减小了服务与管理之间的耦合性。(3)服务管理员可通过调整配置参数的方式直接管理开发人员所构建的服务，这远胜于以往管理单个应用的方式。另外，此设计还能使管理员有针对性地优化业务流程。

3.3 发布/订阅消息模型

分布式系统中两个服务模块之间的消息传递不可避免。例如在本系统中，作为加载服务节点，系统需要知道 18 个数据库节点实时的可用情况，以便制定加载策略。对于 8 个加载服务节点来说，各自都对 18 个数据库节点进行监控是一种性能的浪费。于是系统设计为监控由其中一个节点独立承担，当此监控节点发现有数据库节点发生状态改变时，向 8 个加载节点发送消息。

本系统的消息传递采用发布/订阅消息模型，来屏蔽不同平台、语言和协议对松散耦合的两个服务的影响。该模型中，订阅者向服务中心注册自己希望获取消息的类型，发布者向服务中心注册自己能够提供的信息的类型。当任何发布者向系统提交消息时，服务中心自动把消息发送给所有订阅该消息的订阅者。发布/订阅模式在空间、时间和控制及数据流 3 个方面提供了松耦合能力[3]。

具体在本系统中，如图 4 所示，监控节点对 18 个数据库节点进行监控，然后注册一个表示数据库状态改变的消息主题 T1；8 个加载节点分别订阅主题为 T1 的消息，此时的

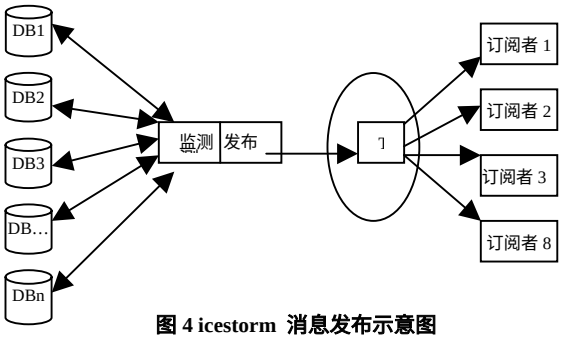

图 4 icestorm 消息发布示意图

加载节点的角色变成图 3 中的订阅者 1、订阅者 2…订阅者 8；当监控节点发现数据库状态有改变时，发送主题为 T1 的消息；此时，订阅者会像收到常规的向上调用(upcall) 一样收到消息。整个过程，发布者使用“ 推” 的消息递送模式， 避免传统消息的轮询模式方式，从而节省了作为订阅者的加载节点的性能。

同理，其它需要传递消息的节点间也可以通过这种订阅者/发布者的形式传递消息。

消息传递采用发布/订阅模型的优点在于：(1)屏蔽了服务系统的复杂性、异构性和需要不断扩展的特性。(2)屏蔽了消息本身内容的多样性和易变性。(3)降低消息两端的耦合度，带来可动态扩展的优势。

4 系统的实现与高可用性优化

本系统采用ICE 中间件辅助实现面向服务架构(ICE—Internet Communications Engine是由前CORBA 专家 Michi Henning 等人在CORBA 的设计思想上，克服CORBA 一些设计上的缺陷，设计的一个全新的面向对象的分布式系统中间件平台。它更加的简单易用，并且能够适应各种尺度的编程模型[4])。具体服务的注册与部署是用icepack 实现，发布/订阅消息模型用icestorm 实现。在系统开始，服务管理节点启动 icepackregistry 服务，每个服务节点都用 Icepacknode 命令在服务管理节点上注册服务，最后系统用 icepackadmin 统一部署服务[4]。本系统中还用服务管理节点作为消息管理节点，启动管理消息的icebox 服务， 能够提供消息的节点和需要提供消息的节点都在服务管理节点的服务上注册，从而达到可以自动传递消息的功能。

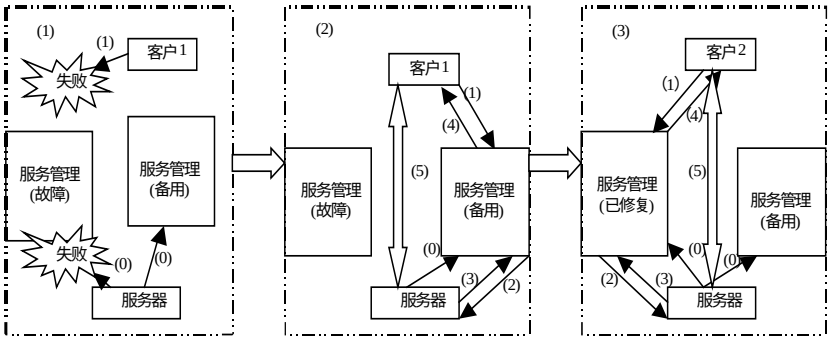

图 5 高可用的实现流程

(下转第 96 页)