分布式企业服务总线平台数据集成研究及应用

范 菁 熊丽荣 徐 聪
(浙江工业大学计算机学院 杭州 310014)

摘 要 为实现大规模的异构数据集成,解决数据源异地分布的问题,满足不同系统和应用之间的信息交互和共享,设计了一种企业服务总线(ESB)平台下的数据集成模型。该模型采用WSDL和XML描述,能够结合ESB系统的集成场景进行数据集成。提出了一种基于消息流程的负载均衡算法,该算法根据服务执行组件的负载情况和分布式节点的资源状况进行流程节点分配,并将其应用于分布式ESB系统的应用集成模型中,能够高效地处理ESB系统数据传输过程中的大量消息,有效解决应用流程执行时存在的消息处理能力低下的问题。最后,以医疗信息系统集成的仿真应用为例,在采用上述模型和算法的分布式ESB平台上,验证了其在解决大规模异构数据服务集成以及消息处理的负载均衡问题时的可行性和有效性。

关键词 企业服务总线,数据集成模型,负载均衡,流程调度
中图法分类号 TP391.9 **文献标识码** A

Research and Application of Data Integration in Distributed Enterprise Service Bus Platform

FAN Jing XIONG Li-rong XU Cong
(College of Computer, Zhejiang University of Technology, Hangzhou 310014, China)

Abstract To implement the integration of large scale heterogeneous data, solve the problem of data source distribution and satisfy the requirementd of information communication and sharing across various systems and applications, this paper presented a solution for data integration in distributed enterprise service bus platform. The data model for integration based on WSDL and XML was proposed. Besides, to improve performance on processing large amount of messages, a load balancing algorithm based on the ESB process was presented. This algorithm can allocate the process node according to the load of component, and is applied to the system integration model of the distributed ESB platform. An example for hospital information system integration was presented, which verifies the feasibility and effectiveness of the proposed framework and algorithm to solve the integration of larger scale heterogeneous data sources and load balancing problem.

Keywords Enterprise service bus, Data model for integration, Load balance, Process scheduling

面对各种自治、异构、分布的数据,如何实现异构数据源数据的灵活转换以及透明集成和访问,从而得到准确及时的高质量信息服务,是当前信息系统集成面临的重大课题。

数据集成是信息系统集成的基础和关键。被集成的数据源通常是独立开发的,存在着语法和语义的异构性[1]。语法异构一般是指源数据和目的数据之间在命名规则、数据格式、数据类型等存在冲突;语义异构一般涉及领域知识,需要直接处理数据的内容,复杂度高,解决的难度较大。

用户一般希望在不影响原有系统运行和不需要进行物理数据集成的情况下,完成异构数据的透明访问。

在数据集成过程中,当集成的数据源数量不断增多时,系统中传输和交换的数据量也会不断变大,因此,安全可靠、高效的数据传输日益受到关注。

分布式企业服务总线(D-ESB)是一种便利的中间件数据集成解决方案,主要是为了解决大规模的跨组织应用和服务集成问题。D-ESB可以将相互关联的分布式异构数据源集成到一起,使用户能够以透明的方式访问这些数据源。在基于ESB的数据集成方法中,数据传输过程中请求消息会在事先指定的服务执行组件中进行处理和传递。但这种数据集成中的同步方式没有考虑数据传输过程中分布式ESB节点的资源状况以及服务执行组件的消息处理能力,容易造成各个节点的负载分布不均、系统的运行效率降低。

本文研究分布式ESB系统的数据集成方法。针对数据源异构性相关问题,本文开展了分布式ESB系统的数据集成模型设计研究,采用适配器技术对异构数据源进行抽取,在ESB系统中采用统一的XML数据共享格式,将异构数据源

到稿日期:2013-04-13 返修日期:2013-07-11 本文受浙江省重点科技创新团队(2009R50009),重大科技专项重大工业项目(2012C11026-2)资助。

范 菁(1969—),女,博士,教授,博士生导师,CCF理事,主要研究方向为虚拟现实及可视化、软件中间件技术,E-mail:fanjing@zjut.edu.cn;熊丽荣(1973—),女,硕士,副教授,CCF会员,主要研究方向为服务计算和软件中间件,E-mail:lilybear@zjut.edu.cn(通信作者);徐 聪(1987—),男,硕士,主要研究方向为软件中间件技术。

信息转换成基于 WSDL 的消息格式，把数据的集成问题转换成 ESB 系统的面向流程的消息集成模型。针对数据传输过程的节点负载分布不均的问题，从数据传输效率的角度，本文研究了分布式 ESB 数据集成的性能问题，采用基于流程负载均衡算法来提高效率。最后在钱塘 ESB 系统上将上述方法应用于面向医疗信息的数据集成，证明了本文方法的有效性。

1 国内外研究现状

在数据集成中异构数据源之间往往存在着语法和语义方面的数据冲突问题。XML 技术能够较好地解决语法层面的数据冲突[2,3]，而对于语义层的异构问题，一般采用知识库和本体技术[4,5]，通过描述不同数据源的概念信息，并构建语义映射关系加以解决。

已有的数据集成工作主要采用联邦数据库系统、数据仓库技术和基于中间件的信息集成技术来实现。

联邦数据库系统(FDS)是多数据库系统数据集成方法，在已存在的局部数据库(Local Database System，LDS)之上为用户提供统一的存取数据环境。FDS 由一组独立的 LDS 组成，实现数据库系统间部分数据的共享[6]。

以数据仓库(Data Warehouse，DW)为主的数据集成方法，通过对相关数据库的链接，抽取数据记录，复制需要的字段，将异构或同构数据源相关数据复制到特定数据源上，从而解决数据的分布和异构的问题，达到集成的目的。

上述两种方法对异构数据大都采取先把数据转换为统一、静态的格式，再将数据转换和整合规则融合在定制代码中。由于应用系统代码与数据库模式紧密耦合，不能适应数据动态变化，这两种数据集成方案较脆弱。

目前主流的数据集成方式是基于中间件的数据集成方案。传统的中间件解决方案是用 DCOM、CORBA 及 RMI 等分布式对象模型来构建信息集成系统。这种方法可以有效地避免联邦数据库系统开发代价大、代码重用难的问题，但分布式对象模型要求服务客户端与服务之间必须进行紧密耦合。

随着 SOA 框架和 Web Service 技术的发展，传统应用中间件向企业服务总线(Enterprise Service Bus，ESB)过渡。ESB 成为企业数据集成采用的主流技术，它将基于事件驱动架构(Event-Driven Architecture，EDA)的思想与面向服务体系架构(Service-Oriented Architecture，SOA)的思想相结合，简化业务单元的集成，在异构平台和环境之间建立了联系[7]。ESB 系统的消息转换和消息路由功能为数据异构性问题提供了解决方案。

集成大量的异构数据源需要建立高效、可靠的数据传输机制，集中式的 ESB 实现架构难以满足集成的需求。IBM 提出了联邦式的 ESB 模式以及中介式的 ESB 模式，将多个服务总线通过 JMS 相联接，以支持大规模的跨组织应用集成[8]。国内外对 ESB 的实现架构进行了大量研究，并建立了多个基于 JBI(Java Business Integration)规范的分布式 ESB 平台，如 Mule ESB[9]、Apache ServiceMix[10]、Open ESB[11]等。目前的一些分布式 ESB 系统已经可以支持应用整合、复杂业务集成以及消息的可靠传输。

在负载均衡方面，动态负载均衡策略考虑的关键问题是及时、准确地把握节点的负载状况，并根据各节点当前的负载状态来动态调整和分配任务[12]。动态负载均衡策略一般可分为两类[13]，一类是求最优解问题，即集群系统的节点提供系统负载、流量及其他一些资源信息，并通过高效通信机制传给负载均衡器，负载均衡器监视所有节点的状态，以便将下一个任务分配给负载最轻的节点。在进行负载最轻的决策时，由于系统处于不断变化中，从更新负载信息到做出决策之间存在时间差，有可能出现负载信息过时的状况。为了解决这一问题，有些研究人员提出先从全集中随机选取包含 K 个元素的子集，再从子集中选择负载最低的一个[14,15]。这种基于反馈的动态选取方式虽然存在一些局限性，但是对于节点数相对较少的分布式系统来说仍然是一种行之有效的方式[16,17]。另一类动态负载均衡策略主要是通过负载迁移的方式来保持节点之间的负载均衡[18-20]，这种策略的关键是确定过载和轻载的对象，以及需要迁移的负载数。负载迁移虽然可以较好地处理系统的过载现象，但是频繁的负载迁移会增加系统的额外开销，而且还会造成“负载抖动”问题[21]。

在基于企业服务总线系统的负载均衡机制方面，目前主流的开源 ESB 项目如 Mule 和 ServiceMix 都在一定程度上加入了一些负载均衡的实现，但是 Mule ESB 的集群比较弱，只能配置一个主实例和一个从实例，因此它的负载均衡机制主要是为了解决服务路由的问题，即根据集成到总线上的服务负载状况，选择负载最轻的一个作为目标服务，以此确定路由路径[22]。

目前，国内外针对基于分布式企业服务总线实现架构下的负载均衡机制的研究还比较少，尤其是面向数据集成场景时，多个集群节点中大量数据消息的负载平衡仍是需要解决的问题。

本文主要研究分布式 ESB 系统中的数据集成模型及方法，以解决异构数据的集成问题，并通过设计负载均衡策略来保证系统在大规模数据集成时的效率。

2 分布式企业服务总线 D-ESB

服务总线系统通过实现多种中介模式来生成特定的中介组件。中介组件的主要工作就是消息表现层的转换和消息内容的转换，消息表现层转换是指改变消息格式，而消息内容的转换主要是为了解决数据集成中的语法及语义异构性问题。

在企业服务总线系统中，可以通过实现特定的组件来完成传输协议的转化，这类组件也通常被称为适配组件，例如数据库适配器、HTTP 适配器等。此外，在 ESB 中，可以通过服务端点来实现服务提供者的地址透明性，因为每一个接入的服务都会暴露成一个服务端点，而服务端点又与特定的适配器相绑定。

本文研究的分布式企业服务总线结构如图 1 所示。整个框架由 ESB 主节点、ESB 从节点、监控管理平台以及流程建模工具 4 部分组成。ESB 主节点主要负责分布式环境管理，监控和统计各个客户端的资源信息，并且通过加载中介组件来提供消息转换和消息路由的功能。ESB 从节点主要负责异构系统及外部应用的接入。监控管理平台是 ESB 服务端与平台管理员的可视化接口，管理员可以通过 Console 完成对各个客户端的资源管理以及消息流监控。流程建模工具给用户提供了图形化工具来完成集成场景的建模和部署。

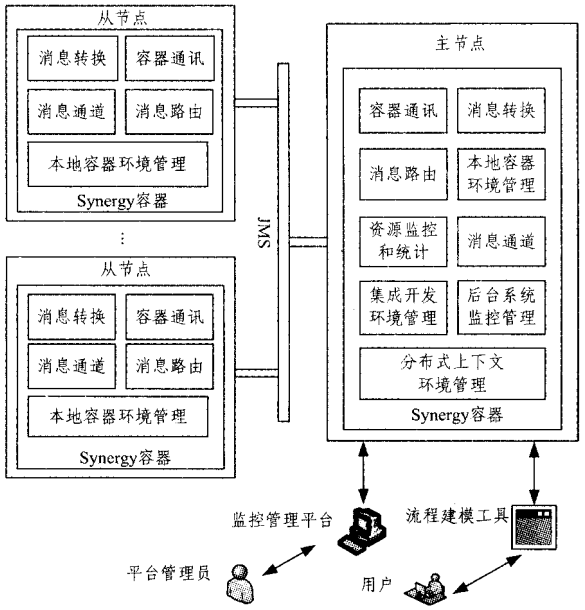

图1　分布式企业服务总线结构图

3　分布式 ESB 系统的数据集成模型

3.1　基于消息的数据集成方法

分布式 ESB 系统采用基于消息的数据集成方法。在这种基于消息的数据集成方法中，适配器作为企业服务总线的一种消息处理组件，主要充当外部异构数据源与总线之间交互的代理，异构数据源通过适配器接入 ESB 总线。适配器和 ESB 总线，以及 ESB 总线内部通过消息进行通信。

分布式 ESB 系统的数据集成基本思路如下：

1)企业服务总线平台在系统/异构数据源的数据传递与互操作上采用 XML 作为数据描述与交换的语言。

2)将异构数据源抽象成统一的 XML 格式数据。将 XML 数据封装成 JMS 消息，将异构数据源数据的转换和传输问题转换成 ESB 环境下的消息转换和路由问题。

3)通过 XSLT 转换引擎实现对不同的 XML 消息格式的转换。

面向数据集成的 ESB 消息转换工作流程如图 2 所示。

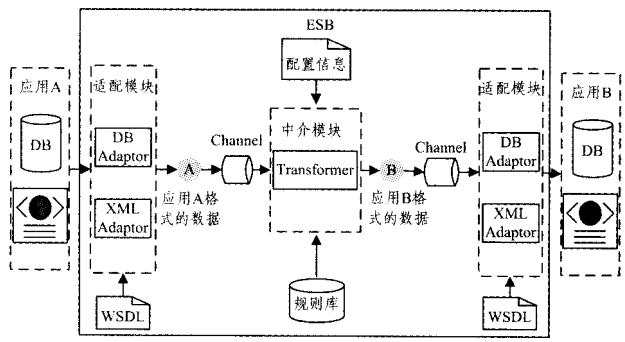

图2　基于 XML 的 ESB 消息转换工作流程图

消息转换的整个流程可分为应用接入和数据转换两个过程。

作为应用接入部分的适配器逻辑完成对具体数据源的包装、连接及各种读写操作。一方面它将从数据源获取的结构化、半结构化数据转换为公共的基于 XML 的数据格式，并以消息流的方式发送到下一个消息组件(转换中介)的消息通道中；另一方面，适配器从消息转换中介的消息通道中取出消息，并将消息体中经过转换的数据格式转换成需要的结构化或半结构化数据，输送到目的数据源。

而 ESB 的中介模块的作用是接收消息发送者所定义格式的消息，经过转换或者路由以后以消息接收者所预期的格式传递到接收方。这个过程中涉及到多方面的消息处理，可以是对消息的格式、内容进行转化，也可以是加密、解密等自定义操作。

在数据集成中，适配器既可以作为服务提供者，也可以作为服务的消费者。以数据库适配器为例，作为服务提供者的数据库适配器提供了 4 个接口，即 CRUD 这 4 种数据库操作，这些操作可以通过标准的服务描述规范配置成具体的某一个服务发布到注册中心。外部系统可以通过服务注册中心查找特定种类的服务，在获得服务的描述信息以后去调用服务进行具体操作。而在 ESB 环境内部中，适配器可以接收到消息通道中的 ESB 消息，并对消息内容进行解析和提取，之后调用数据库适配器内部接口进行具体的数据库操作。处理完成后通过适配器把回复内容转化为 ESB 消息，消息目标为服务消费者的回复端点。

作为服务消费者的数据库适配器可以定时地检测本地数据库表是否有记录更新，发现有新的记录则将更新的内容封装成 ESB 消息以后进行输出。适配器只需要知道要负责监听的数据库表的地址就可以完成如上的操作。

对于如何提供这个地址给适配器，具体的适配器解决的方式也可以不同。比如，可以将监听地址配置在流程中，当适配器框架部署某个适配器的流程时将地址信息部署到该适配器上，之后适配器就能够启动监听；另一种方式也可以将监听地址的信息配置在服务配置文件中，在部署服务的时候将地址信息部署到某个具体适配器上，之后在部署服务端点的时候适配器启动监听。本文的数据库适配器在开发的时候选择了后者的处理方式。

3.2　基于 WSDL 的消息模型

企业服务总线平台内部的适配模块要在异构系统与 ESB 之间提供代理的功能，特定的适配器可以抽象出接入到总线上的数据源的具体实现，通过协调不同系统间的传输协议(例如 FILE、JMS、SOAP、JDBC 等)来完成消息转换过程中的传输层转换。为了实现上述功能以及组件间的交互通信，在面向数据集成的企业服务总线平台中定义了基于 WSDL 的消息模型。

JBI 规范中使用 WSDL1.1 和 2.0 规范描述组件所提供的和消费的服务模型。WSDL 在以下两个层面上定义了基于消息的服务模型：

1)抽象服务模型：使用抽象消息模型定义的、未限制到特定消息交换协议的服务。WSDL 服务描述中抽象服务模型需要定义以下几个元素：抽象消息类型(Message Type)、抽象操作(Operation)以及抽象服务类型(Service Type)。

2)具体服务模型：建立在抽象服务模型之上，为抽象服务同特定通信协议及通信端点的映射提供描述信息。WSDL 具体服务模型需要定义以下几个元素：绑定类型(Binding Type)、端点(Endpoint)以及服务(Service)。

以外部应用为数据库系统为例，在面向数据集成的企业

服务总线平台中定义了如下的基于 WSDL 的消息模型：

1）type：Message Type 表示抽象消息类型，类型中定义了合法的消息结构和约束，一般通过 XML Schema 来表示，如图 3 所示。

```
〈? xml version="1.0" encoding="UTF-80"?〉
〈wsdl:definitions targetNamespace="http://www.zju.org/synergy/"
    xmlns="http://schema.xmlsoap.org/wsdl/"
    xmlns:ns="http://j2ee.netbeans.org/xsd/tableSchema"
    xmlns:wsdl="http://schemas.xmlsoap.org.wsdl/"
    xmlns:tns="http://www.zju.org/synergy/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:jdbc="http://www.zju.org/esb/wsdl-extensions/jbdc/"〉
〈wsdl:documentation/〉
〈wsdl:types〉
  〈xsd:schema〉
    〈xsd:import namespace="http://j2ee.netbeans.org/xsd/tableSchema"schemaLocation="PERSON.xsd" /〉
  〈/xsd:schema〉
〈/wsdl:types〉
```

图 3　WSDL 消息模型中的抽象消息类型定义

type 元素中通过 import 引入的 XSD 文件（XML Schema Definition）即 XML 结构定义，描述了 XML 文档的存放结构，因此可以通过该文件来生成 JMS 消息体中的 XML 文档格式。例如，当外部应用为数据库系统时，数据接入服务需要在 XML 文档结构和数据库结构之间建立映射，把数据从数据库转换成 XML 文档，或者把一个 XML 文档转换到数据库中。本文将关系模式映射到 XML 模式，图 4 表示了将数据库表 person 结构映射成 XML Schema 的 person.xsd 文件。

```
/*个人信息结构表*/
create table person (
      id varchar(10),
      name varchar(255) not null,
      processed boolean,
      constraint pk_id primary key (id)
);
〈? xml version="1.0" encoding="UTF-80"?〉
〈xsd:schema elementFormDefault="qualified"
  targetNamespace="http://j2ee.netbeans.org/xsd/tableSchema"
    xmlns="http://j2ee.netbeans.org/xsd/tableSchema"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
〈xsd:element name="PERSON" type="PERSON"/〉
 〈xsd:complexType name="PERSON"〉
  〈xsd:sequence name="PERSON"〉
    〈xsd:element name="ID" type="xsd:string" /〉
    〈xsd:element name="NAME" type="xsd:string" /〉
    〈xsd:element name="PROCESSED" type="xsd:boolean" /〉
  〈/xsd:sequence〉
 〈/xsd:complexType〉
〈/xsd:schema〉
```

图 4　数据库表映射为 XML Schema 文件 person.xsd

2）operation：Abstract Operation 表示抽象操作，它定义了与某种服务进行交互的一次操作，抽象操作中一般定义了操作名称、消息交换模式以及消息类型。

3）portType：portType 表示抽象服务类型，也可以称为“服务接口（interface）”，它是一组相关联的抽象操作（operation）的集合，图 5 中定义了一个名称为 jdbcPollerPortType 的接口，该接口定义了一个名称为 pollrecords 的抽象操作。

```
〈wsdl:portType name="jdbcPollerPortType"〉
  〈wsdl:operation name="pollrecords"〉
    〈wsdl:input name="inputPoll" message="tns:inputMsg"〉
    〈/wsdl:input〉
  〈/wsdl:operaton〉
〈/wsdl:portType〉
〈wsdl:binding name="pollerBinding" type="tns:jdbcPollerPortType"〉
  〈jdbc:binding/〉
  〈wsdl:operation name="pollrecords"〉〈jdbc:operation/〉
    〈wsdl:input name="inputPoll"〉
      〈jdbc:input TableNmae="PERSON" operationType="poll"
        PKName="ID" paramOrder="" resultOrder="id,name"
        PollMilliSecond="500" MarkColumnName="PROCESSED"
        MarkColumnValue="1"
        sql="select id,name from person where processed=false"/〉
      〈/wsdl:input〉
    〈/wsdl:operation〉
〈/wsdl:binding〉
```

图 5　WSDL 消息模型中的操作定义

4）binding：binding 表示绑定类型，它是对 portType 中定义的服务接口的具体实现，在图 5 的 wsdl:binding 元素内部定义了 pollrecords 操作的实现细节。

5）service：service 提供了访问该服务的一组端点的集合，每一个服务实现了特定的服务类型（接口），其中包括了服务名称、服务类型名称（portType）以及端点，图 6 中定义了访问服务类型名称为 pollerPort 的一个端点。

```
〈wsdl:service name="jdbc-service"〉
  〈wsdl:port name="pollerPort" binding="tns:pollerBinging"〉
    〈jdbc:address dialect="mysql"
      driverClassName="com.mysql.jdbc.Driver"
      dbURL="jdbc:mysql://localhost:3306/defaultdb"
      userName="root" password="root"/〉
    〈/wsdl:port〉
〈/wsdl:service〉
```

图 6　WSDL 消息模型中的端点定义

3.3　基于 XSLT 的消息转换

企业服务总线平台内部所涉及的消息转换按层次角度区分，可以分为 3 类：传输层转换、消息表现层转换和消息内容转换层。其中传输层转换主要是为了协调不同系统间传输协议的区别，主要由适配模块完成；消息中介模块则负责消息表现层转换与消息内容转换层，图 2 中所示即为将应用 A 格式的数据转换为应用 B 格式数据的过程。

为了解决异构数据间的映射问题，在 ESB 平台内部采用 XML 来描述和存储数据，再通过映射规则进行数据转换。常用转换规则表示有 XSLT（eXtensible Stylesheet Language Transformations）和用户自定义规则等。XSLT 的主要功能

就是转换，它将一个没有形式表现的 XML 内容文档作为源树，将其转换为一个可显示的有样式信息的结果树。同时，在 XSLT 文档中定义与 XML 文档中各个逻辑成分相匹配的转换模板，以及匹配转换方式。

图 2 中的 Transformer 是将源消息格式转换为目的消息格式的主要模块，它可以根据配置信息来选择合适的消息转换器，并按照规则库中存储的 XSLT 规则把消息转化成合适的格式，图 7 所示为根据 XSLT 语法规则定义的样式文件。

```
〈? xml version="1.0" encoding="UTF-8"?〉
〈xsl: stylesheet xmlns: xsl = " http://www. w3. org/1999/XSL/Transform"
    xmlns: fn="http://www. w3. org/2005/02/xpath-functions"
    xmlns: var = " http://www. synchroesb. org/2007/var" verson = "2.0"〉
  〈xsl: template match="/"〉
  〈/xsl: template〉
〈/xsl: stylesheet〉
```

图 7　基于 XSLT 规则的样式文件

该样式中包含了 xsl:stylesheet 元素，该元素是 XSLT 文件的根元素，它包含了 XSLT 名字空间、函数名字空间、变量名字空间以及版本信息。在 XSLT 中使用 xsl:template 表示模板元素，它也是 XSLT 中最重要的一个属性元素，每个 xsl:template 有一个节点匹配属性，由“match=”指定。在对模板进行匹配时使用“xsl:apply-templates”选择要匹配的模板。

4　基于消息流程的 ESB 数据集成

4.1　基于分布式 ESB 平台的数据集成框架

基于分布式 ESB 平台的数据集成框架如图 8 所示。

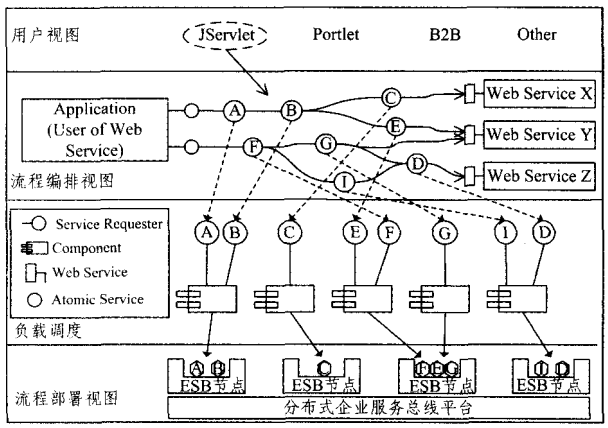

图 8　基于分布式 ESB 平台的数据集成框架

在数据集成框架中，用户视图代表的是外部应用层，对于特定的数据集成场景，一般都由用户来进行指定。流程编排的主要过程是用户根据具体的集成场景，使用特定的描述语言来完成对信息集成的建模。流程部署主要在系统的后台执行，主要过程是将流程中的单个服务部署到位于不同 ESB 节点上的执行组件中。而系统的负载均衡机制主要负责将流程中的单个服务与合适的执行组件进行绑定。

结合分布式 ESB 平台的集成方法，本文可以通过 ESB 流程建模工具对数据集成场景建模[23]，然后将生成的流程文件部署到后台系统执行以实现数据集成。

4.2　数据集成流程

数据集成流程是对数据集成场景的逻辑表示，由许多个相关的流程节点组合而成。为了更好地描述流程这个逻辑概念，以及将相关的流程节点部署到对应的执行组件上，本文引入 XML 描述语言以便很好地表述消息代理中介的逻辑符号与实际的消息代理中介之间的对应关系，并且能够表示流程节点中的参数信息以及相关节点之间的链接关系。通过构造流程配置文件来描述相关的中介规则信息以及服务描述信息。

下面给出相关概念和数据集成流程的相关定义。

1)Component：表示了分布式企业服务总线平台中的服务执行组件，也称为消息代理组件。它主要负责接收请求消息，处理消息的内容，并且将处理后的消息进行转发。系统中主要存在两种类型的组件：适配组件(adaptor)和中介组件(mediator)。

2)Web Service：表示所请求的外部服务，它可以是数据源、Web 服务(Web Service)或者应用程序等。

3)Application：表示服务的请求者，它也可以是数据源或者 Web 服务。

4)Atomic Service：表示了应用流程中的单个节点，可以将它看成是原子服务。流程中的所有节点可以分为两类：中介规则(mediation)和端点(endpoint)。中介规则描述了消息中介的规则(例如消息验证、消息路由以及消息转换)。对应数据集成场景，中介规则为消息转换。它被部署到对应的中介组件以后，中介组件才能执行具体的中介操作；端点又可分为两类，一类描述了所请求的服务的相关信息，比如服务名称以及服务地址等信息，另一类描述了客户端通过怎样的方式将请求发送给 ESB，比如暴露 HTTP 端口等等。与中介规则类似，端点被部署到对应的适配组件以后，适配组件才能执行具体的适配操作。

表 1 中给出了流程节点的常见服务名称及类型信息。

表 1　ESB 流程节点的服务类型示例

服务名称	组件类型	流程节点	执行组件
消息聚合	Aggregate	mediation	中介组件
消息分裂	Split	mediation	中介组件
基于内容路由	CBR	mediation	中介组件
消息广播	Recipient	mediation	中介组件
消息格式转换	Transformation	mediation	中介组件
消息内容转换	Transformation	mediation	中介组件
数据库接入	Database	endpoint	适配组件
文件接入	File	endpoint	适配组件
Web Service 接入	WebService	endpoint	适配组件

对于任意一条数据集成流程，给出如下形式化定义：

定义 1　esb process 表示基于 ESB 平台的数据集成流程，它主要由中介规则序列和服务端点集合组成，因此，它的主要属性包括 *processid*、*inendpoints*、*mediations*、*outendpoints*。其中：①*processid* 表示 ESB 流程的名称，全局唯一；②*inendpoints* 表示请求者端点集合，流程中可能存在多个服务请求者的相关信息；③*mediations* 表示中介规则的集合；④*outendpoints* 表示服务端点的集合，流程中有可能存在多个服务端点。

定义 2　requester endpoint 表示请求者端点，它描述了接入到总线上的服务请求者的相关信息。它的主要属性有：

consumer,*type*,*adaptor*,*description*,*nexthop*。其中:①*consumer* 表示服务请求者的名称,它在一条流程中是唯一的;②*type* 表示用于执行该流程节点的组件类型信息;③*adaptor* 表示用于执行该流程节点的组件名称以及位置信息;④*description* 表示服务请求者的描述信息;⑤*nexthop* 表示与该流程节点有逻辑关系的下一个节点的名称信息。

定义 3 mediation 表示中介规则,描述了中介服务信息,主要包括 *mediation*,*type*,*mediator*,*rule*,*inbounds*,*outbounds* 等属性。其中:①*mediation* 表示该中介服务的名称,它在一条流程中也是唯一的;②*type* 表示用于执行该流程节点的组件的类型;③*mediator* 表示用于执行该流程节点的组件名称以及位置信息;④*rule* 表示具体规则的内容,包括一些参数信息;⑤*inbounds* 表示与该流程节点逻辑相关的前一(或前几个)流程节点的名称;⑥*outbounds* 表示与该流程节点有逻辑关系的下一个(或几个)节点的名称信息。

定义 4 service endpoint 表示服务端点,它描述了服务提供者的相关信息,包含的属性有:*service*,*type*,*adaptor*,*description*,*replyto*。其中:①*service* 表示服务提供者的名称,它在一条流程中是唯一的;②*type* 表示用于执行该流程节点的组件类型信息;③*adaptor* 表示用于执行该流程节点的组件名称以及位置信息;④*description* 表示服务提供者的描述信息;⑤*replyto* 表示请求是否需要返回结果。

图 9 是一个基于 XML 的流程片段描述。

```
〈EsbProcess〉
  〈endpoints〉
    〈endpoint name="FeeRecordOutput" type="consumer"
     service="PollFeeRecord" adaptor="DataBaseAdaptor"
     container="clientcontainerl"〉
        〈description〉Output update records from Table FeeRecord〈/
         description〉
        〈mediation name="IntegrateInfo" sendSync="false" relia-
         ble="false" /〉
    〈/endpoint〉
    〈endpoint name="AggRecordInput" type="provider"
     service="InsertAggRecord" adaptor="DataBaseAdaptor"
     container="clientcontainer2"〉
      〈portType name="jdbcInsertPortType" /〉
        〈operationName〉insertAggRecord〈/operationName〉
      〈description〉Insert aggregated records into Table AggRecord
        〈/description〉
    〈/endpoint〉
  〈/endpoints〉
〈/EsbProcess〉
```

图 9 基于 XML 的流程片段描述

4.3 数据集成的流程部署

在流程部署视图中,当一条数据集成流程被部署到分布式 ESB 平台的主容器时,容器中的流程管理器会将流程分割成 n 个流程节点。每个流程节点中的 *type* 属性是在流程编排时由用户事先配置的,主容器的负载均衡模块会首先获取该节点的组件类型信息,之后找出合适的执行组件,并将信息添加到该流程节点中,这一过程被称为流程的重配置过程。最后,流程管理器会根据重配置后的结果,将流程节点分配到相应 ESB 节点上的相应执行组件中。

本文在分布式 ESB 系统 JTangSynergy[24,25] 上展开研究,在目前的 Synergy 系统的实现机制中,处理消息流的消息处理器只有一个,它需要负责各种消息流的处理;其次,所有的消息都存放在唯一的一个消息接收通道中,消息代理组件在取出消息以后还需要匹配相关的配置信息,才能决定采用哪种处理方式来处理特定种类的消息。这种处理方式相对效率较低。在本文的方案中,考虑在 D-ESB 主节点中增加负载均衡器,其负责将中介规则以及服务端点等信息分配到合适的消息代理中介上。同时,通过设计消息代理中介的消息队列模型,来提高中介的消息处理和传递效率。本文第 5 节将介绍负载均衡方案。

5 分布式 ESB 数据集成中基于流程的负载均衡策略设计

5.1 消息多队列模型

为了提高执行组件的消息处理性能,在实现流程部署的过程中,我们设计了如图 10 所示的执行组件的多消息队列模型。

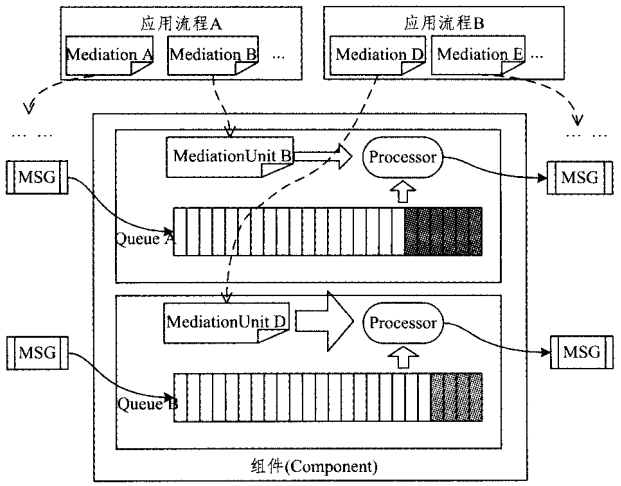

图 10 服务执行组件的多队列模型图

其中,每个组件内部有可能存在多个实例,而每个实例主要由 3 部分组成:处理器(Processor)、消息队列(Queue)和中介单元(MediationUnit)。消息队列是消息系统中的基本数据结构,主要用于存储消息,队列的创建和销毁都是基于 JMS 技术,每当一个流程节点部署到某个执行组件时,组件就会初始化一个具体的处理实例,并由该实例负责对应的消息接收队列的创建,该队列只负责接收与部署的流程节点相关的请求消息;MediationUnit 是流程中的中介(mediation)被部署到执行组件的时候创建的,它实际上是一个内存中的片段,描述了路由规则或者转换规则等信息;Processor 负责从消息通道中获取消息,处理消息体中的内容,最后根据中介单元的信息将处理后的消息转发给下一个目的地。该模型具有以下的一些优势:

1)与特定流程相关的消息都会被发送到指定的消息队列中,使得各个消息流可以并行传输而不会相互影响,从而提高了消息传递的可靠性;

2)组件中的每个实例都会从自己的消息队列中获取和处理同一类型的请求消息,从而提高了消息中介处理和传递消息的能力。

5.2 服务执行组件的负载评价

传统的负载均衡策略多数是基于网络的负载平衡和基于操作系统的负载平衡,这些策略的算法和实现都比较复杂,动

态参数很多而且难以控制。例如:基于操作系统的负载均衡策略常常通过计算系统资源来评价服务器的负载,比如 CPU 利用率、内存使用率等等。

中间件技术提供的异构环境下的通信和互操作功能,为解决分布式企业服务总线平台的负载均衡问题提供了有效的工具。分布式企业服务总线平台中负载调度的对象是执行组件,因此,本文提出将组件的消息队列数以及各个消息队列中的剩余消息总数作为组件负载评价的标准。结合之前提出的组件的多队列模型,对于分布式 ESB 系统中的任意一个执行组件,通过一个五元组来定义它的负载度量模型:

$$Component_i = (ComponentId, Type, TaskQueueSet, Qnum, Load)$$

其中,

1)*ComponentId* 表示执行组件的全局名称;

2)*Type* 表示该执行组件的类型;

3)*TaskQueueSet* 表示组件中消息队列的集合,对于其中任意一个消息队列 *Queue*,它又可以表示为 $Queue=(QID, ProcessID, QLength)$,其中,*QID* 表示消息队列的标识符,全局唯一,*ProcessID* 表示与该消息队列相关的流程名称,*QLength* 表示该队列中的剩余消息数;

4)*Qnum* 表示组件在某个时刻的消息队列数,它的值可以通过 *TaskQueueSet* 的长度获得;

5)*Load* 表示组件中所有消息队列的剩余消息数总和。

根据以上定义,分布式 ESB 系统中任意一个执行组件中的 Load 值可表示如下:

$$Component_i.Load=\sum_{j=1}^{n} Queue_j.QLength, 1\leqslant j\leqslant Qnum$$

当获取了执行组件的 $Component_i.Qnum$ 值和 $Component_i.Load$ 值后,采用如下的策略来比较具有相同服务类型的组件的负载高低:

1)优先比较组件列表中所有组件的 $Component_i.Qnum$ 的值,选择其中值最小的那个组件;

2)如果同时存在最小任务数相同的执行组件,则比较组件的 *Component.Load* 值,选择其中值更小的那个;

3)如果 $Component_i.Qnum = Component_j.Qnum$,并且 $Component_i.Load = Component_j.Load$,则优先考虑位于本地 ESB 节点的执行组件,如果找不到,则在组件列表中随机选取一个远程节点的执行组件。

5.3 基于流程的负载均衡策略

首先定义以下几个数据结构:

1)组件负载信息表(*ComponentId*,*Load* 等);

2)定时器 *H*;

3)组件列表 *list*{},初始为空,记录属于同一服务类型的组件 ComponentId。

基于流程的动态负载均衡策略主要由以下 3 个阶段组成:主节点负载信息的维护、基于负载统计的信息更新以及基于反馈的流程节点分配。具体步骤如下:

1)负载信息维护

①ESB 主节点成功启动后,主容器初始化组件负载信息表(*ComponentId*,*Load* 等);

②当任意一个 ESB 节点中有新的执行组件需要动态加载时,该节点的容器会将该组件的 *ContainerId*、*ComponentId*、*ComponentType* 以及 *State* 等信息以事件消息的方式通过消息通道发送给主节点容器,此时,组件的 *Qnum* 以及 *Load* 的初始值都为 0;

③主容器接收该组件初始化的事件消息,并将消息中的内容写入负载信息表中;

④当某个 ESB 节点需要动态卸载某个执行组件时,节点的本地容器会将 *ContainerId* 信息以及组件的 *ComponentId* 信息通过消息通道发送给主节点,主节点根据收到的信息将负载信息表中对应的组件删除。

2)负载信息更新

①主容器的资源监控模块使用定时器 *H*,当更新时刻到时,向各 ESB 节点中的组件发起当前负载查询请求;

②组件接收到查询请求,将当前的实时负载信息通过本地容器以事件消息的方式发送给主容器;

③主容器接收各节点反馈的负载信息,其中包括组件的 *Qnum* 和 *Load* 这两个参数的值,并将它们更新到负载信息表;

④主容器会根据组件类型(*Type*)创建多个组件列表 *list*,并将负载信息表中属于同一类型的组件加入到相同的 *list* 中。

3)流程节点分配

①解析流程:流程第一次部署到主容器时,主容器会首先解析流程配置文件,将中介规则和服务端点分割成 *n* 份,并且获取中介规则和服务端点中的组件类型信息 *type*;

②定位组件:根据上一步获取的 *type* 值,负载均衡器会根据 5.2 节中的评价策略在对应组件列表中查找其中负载最轻的组件;

③重组流程:根据上一步得到的组件,负载均衡器会获取组件的 *ContainerId* 和 *ComponentId* 值,并以"*ContainerId* ⋃ *ComponentId*"表示该流程节点的执行组件名称,该信息会被添加到流程节点中的"*adaptor*"属性或者"*mediator*"属性中;

④分配流程节点:流程管理器获取重组后的流程文件并再次解析,对于其中每个节点,获取"*adaptor*"属性或者"*mediator*"属性的值,之后,主容器根据 *ContainerId* 信息将该流程节点的完整内容发送给对应的 ESB 节点,并且由 ESB 节点根据 *ComponentId* 信息将流程节点最终交给本地的执行组件。

6 仿真实验

基于以上的研究,本文在分布式 ESB 系统 JTangSynergy 上采用本文的数据集成模型和负载均衡方法,对医疗系统进行数据集成。JTangSynergy 主要采用了 Master/Slave 的容器集群架构,主节点实现了分布式环境管理以及资源监控和负载均衡,从节点完成外部应用及异构数据源的接入。此外,用户可以通过系统提供的图形化流程开发工具完成应用场景的流程建模,并且通过监控管理平台 Console 实现对系统的资源管理和消息流监控。下面以模拟医院内部各个信息系统之间的数据集成为例对基于分布式 ESB 平台的应用步骤进行说明。

(1)服务封装。在基于 ESB 应用的角色分配中,服务一般是由第三方负责提供,而 ESB 平台主要完成服务集成的工作。但是,用于应用集成的服务也可由 ESB 系统的提供商负责统一封装。在这个仿真实验中,所涉及到的服务都是针对数据库访问操作,目的是实现对数据库数据集成操作的复用。针对这个应用案例,共封装了 12 个服务,如表 2 所列。

表 2　医疗信息集成案例中的所有服务

服务名称	服务描述	提供方
PollPatient	从挂号部门的 Patient 表中取出新产生的病人记录	HIS 挂号系统
SyncPatient_dc	新产生的病人基本信息同步到数据中心的 Patient_dc 表中	数据中心
SyncPatient	新产生的病人部分信息同步到医嘱系统的 Patient 表中	NIS 医嘱系统
PollAdvice	从门诊部门的 Advice 表中取出新产生的病人医嘱记录	NIS 医嘱系统
SyncAdvice_dc	新产生的病人医嘱信息同步到数据中心的 Advice_dc 表中	数据中心
SyncAdvise	新产生的病人医嘱部分信息同步到数据中心的 FeeRecord_dc 表中	EMR 计费系统
PollFeeRecord	从收费部分的 FeeRecord 表中取出病人的费用记录	EMR 计费系统
SyncFeeRecord_dc	新产生的病人费用信息同步到数据中心的 FeeRecord_dc 表	数据中心
SelectPatientinfo_dc	根据病人 ID 从数据中心的 Patient_dc 表中查询病人的基本信息	数据中心
SelectAdviceinfo_dc	根据病人 ID 从数据中心的 Advice_dc 表中查询病人医嘱信息	数据中心
SelectFeeRecord_dc	根据病人 ID 从数据中心的 FeeRecord_dc 表中查询病人费用信息	数据中心
InsertPatientInfo	将病人的所有数据信息汇总后插入文档数据库中	LIS 综合系统

(2)启动系统各个节点,安装组件。分别在需要进行集成的 HIS 系统、NIS 系统、EMR 系统以及 LIS 系统所在的服务器上部署一个 ESB 的从节点,并在数据中心的服务器上也部署一个 ESB 的从节点,另外,还需要一个高性能的服务器来部署 ESB 的主节点。这样一来,就确定了一个 ESB 主节点和 5 个标准节点的集成环境。此外,还需要在各个 ESB 节点上安装由系统提供的执行组件。图 11 和图 12 是通过监控管理平台的界面显示的 ESB 各个节点(客户端)以及执行组件的信息。

图 11　监控管理平台中的 ESB 节点管理列表

图 12　集成 NIS 系统的 ESB 节点下的中介组件列表

(3)数据集成流程的建模和部署。针对当前的应用案例,共设计了 4 条数据集成的应用流程,分别是病人信息同步流程、医嘱信息同步流程、收费信息同步流程以及病人信息汇总流程,在这些流程中应用了广播路由、聚合路由以及消息转换等服务。图 13 所示的是对病人信息同步流程的建模界面。

在图 13 所示的病人信息 HIS 系统原始数据源采用 MySql 数据库,NIS 系统的数据源采用 SqlServer 数据库,以符合数据源异构性的特征。在 Mysql 数据库中创建一张存储病人信息的表 Patient,表结构如表 3 所列;在 SqlServer 数据库中创建一张以 HL7 标准格式存储病人信息的表 Patient_HL7,表结构如表 4 所列。

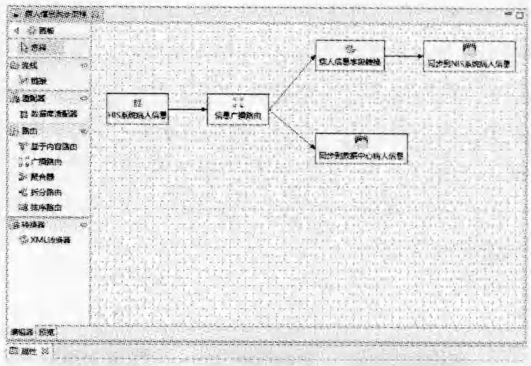

图 13　流程建模工具中病人信息同步流程图

表 3　Patient 表的表结构

Column Name	Datatype	NOT NULL	Key
ID	BIGINT(20)	No	Yes
Name	VARCHAR(50)	No	No
Birthday	DATETIME	No	No
Sex	VARCHAR(10)	Yes	No
Address	VARCHAR(150)	Yes	No

表 4　Patient_HL7 表的表结构

Column Name	Datatype	NOT NULL	Key
PID	BIGINT(20)	No	Yes
Name	VARCHAR(50)	No	No
Age	INTEGER(8)	No	No
Gender	VARCHAR(10)	Yes	No
Address	VARCHAR(150)	Yes	No

该场景下需要完成的转换操作主要有 ID-PID、Sex-Gender、Birthday-Age 这 3 组对象名称的转换以及 Birthday-Age 的值转换操作,通过流程定义器的配置界面定义生成转换规则集(命名规则和计算日期的运算函数规则),将上述规则作为参数进行输入,并根据定义的转换算法生成针对该具体实例的转换节点,从而实现了数据集成中的数据兼容问题。流程设计器中创建抽取原始数据源以及写入目的数据源的节点,这两个节点是通过适配器组件经过配置后生成。将生成的流程导出成为配置文件包,并将它部署到已运行的 ESB 环境中,至此,实现了对病人信息转换和信息同步的流程。

(4)系统运行及消息流监控。对于每条应用流程,可以通过监控平台来查看实时的消息流状况。图 14 显示了病人信息汇总流程的实时消息运行状况,对于流程中的每个节点,监控管理平台中都会显示该节点的消息发送和接收数据量。

图 14　病人信息汇总流程的消息流监控图

在该医疗信息集成测试用例下,需要同时运行的消息流程有 4 条。为了获取系统目前的消息吞吐量的变化情况,选择进行宽泛的测试方案,对于每条消息流程,保持每秒钟发送 330 条消息的数据量。当所有流程同时运行时,系统的总体消息传输率会达到 900 条/秒左右。

对于每条消息流程的运行情况,可以通过"监控"来查看实时的消息流状况,图 14 显示了病人信息汇总流程在传输了

130 万条消息时的运行状况，对于流程中的每一个节点，监控管理平台都会显示出该节点的消息发送和接收数据量，可以看到图中的病人消息聚合路由的消息接收数是前面 3 个节点的消息发送数之和，而发送出去的消息数是已经处理完的消息。图 15 显示了 ESB 主节点的实时负载监控情况，从中反映出了节点所在服务器的 CPU 使用率、内存使用率以及 JVM 的使用率等信息。

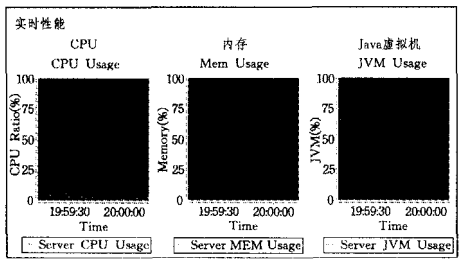

图 15　ESB 主节点的实时负载监控

从图上结果可以看到，基于本文的数据集成方法以及负载均衡策略实现的分布式企业服务总线平台可以有效解决异构分布式环境下的数据集成问题，并且可以充分利用各个 ESB 节点的系统资源，完成百万级数据量下的消息流调度，能够较好地满足实际应用集成的需求。

结束语　本文在基于分布式企业服务总线的系统框架之上，为了解决系统在面向大规模数据集成时碰到的数据源异构性问题，定义了一种基于 WSDL 和 XML 的数据集成模型；此外，为了解决系统的数据传输效率问题，提出了一种基于流程的负载均衡算法，以实现对分布式 ESB 各个节点中消息资源的有效分配，从而提高应用集成时的运行效率。

下一步工作将对大规模应用集成下的数据安全性以及消息传输的可靠性进行深入研究，通过在分布式 ESB 平台内部实现有效的安全机制来提高系统的安全性。

参 考 文 献

[1] 陈跃国，王京春. 数据集成综述[J]. 计算机科学，2004，31(5)：48-51

[2] 齐建军，刘爱军，雷毅. 基于 XML 模式的制造信息集成规范的研究 [J]. 计算机集成制造系统，2005，11(4)：565-571

[3] 韦银星，张申生，周晓俊，等. 企业应用集成技术研究[J]. 计算机集成制造系统，2002，8(8)：593-596

[4] Wache H, Voegele T, Visser U, et al. Ontology-based integration of information-a survey of existing approaches[C]// IJCAI-01 workshop: ontologies and information sharing. 2001: 108-117

[5] Milanovic N, Kutsche R, Baum T, et al. Model & metamodel, metadata and document repository for software and data integration[C] // MoDELS'08 Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems. Springer Berlin Heidelberg, 2008: 416-430

[6] Sheth A P, Larson J A. Federated database systems for managing distributed, heterogeneous, and autonomous databases[J]. ACM Computing Surveys (CSUR), 1990, 22(3): 183-236

[7] Maréchaux J L. Combining service-oriented architecture and event-driven architecture using an enterprise service bus[Z]. IBM Developer Works, 2006: 1269-1275

[8] Grund V, Rexroas C. Enterprise Service Bus implementation patterns[Z]. 2007

[9] Mule Open Source ESB [EB/OL]. http://www.mulesoft.org/display/MULE2USER/Outbound + Routers # OutboundRouters-RoundRobin

[10] ServiceMix A. The agile open source ESB[Z]. The Apache software foundation

[11] Open ESB. Sun open source ESB project [EB/OL]. http://java.net/projects/openesb

[12] 徐卫东，王康. 适用于内容分发网络的动态负载均衡策略[J]. 计算机科学，2005，32(1)：41-44

[13] 杨际祥，谭国真，王荣生. 并行与分布式计算动态负载均衡策略综述[J]. 电子学报，2010，38(005)：1122-1130

[14] Mitzenmacher M. How useful is old information? [J]. IEEE Transactions on Parallel and Distributed Systems, 2000, 11(1): 6-20

[15] Dahlin M. Interpreting stale load information [J]. Parallel and Distributed Systems, IEEE Transactions on, 2000, 11(10): 1033-1047

[16] Wang J, Ren Y, Zheng D, et al. Agent based load balancing middleware for service-oriented applications [C] // Computational Science-ICCS 2007. Springer Berlin Heidelberg, 2007: 974-977

[17] Guyton J D, Schwartz M F. Locating nearby copies of replicated Internet servers[M]. ACM, 1995

[18] Zeng W, Li Y, Wu J, et al. Load Rebalancing in Large-Scale Distributed File System[C]// 2009 1st International Conference on Information Science and Engineering (ICISE). IEEE, 2009: 265-269

[19] Roca J, Ortega J C, Álvarez J A, et al. Data neighboring in local load balancing operations[C]// Proceedings of the 9th WSEAS International Conference on Computers. World Scientific and Engineering Academy and Society (WSEAS), 2005: 1-6

[20] Cheung A K Y, Jacobsen H A. Dynamic load balancing in distributed content-based publish/subscribe [M]. Springer Berlin Heidelberg, 2006

[21] 鲍春健，吴俊敏，许胤龙，等. 支持动态负载平衡的分层消息队列模型[J]. 计算机工程与应用，2007，43(1)：155-158

[22] Jongtaveesataporn A, Takada S. Enhancing enterprise service bus capability for load balancing[J]. WSEAS Transactions on Computers, 2010, 9(3): 299-308

[23] Deng S, Wu Z, Kuang L, et al. Management of serviceflow in a flexible way [C] // Web Information Systems-WISE 2004. Springer Berlin Heidelberg, 2004: 428-438

[24] Chen H, Yin J, Liu H, et al. JTangSynergy 3.0: A framework and software tool for integrating cross-organizational applications[C] // CollaborateCom 2009. 5th International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2009. IEEE, 2009: 1-7

[25] Yin J, Chen H, Deng S, et al. A dependable esb framework for service integration[J]. Internet Computing, IEEE, 2009, 13(2): 26-34