文章编号:1001 - 9081(2008)04 - 0860 - 03

基于 ESB 的实时 ETL 系统的设计与实现

高 彬,谷建华,符 宁,张海辉
(西北工业大学 计算机学院,西安 710072)
(gaobinnew@163.com)

摘 要:随着数据仓库技术的应用发展,企业对数据的处理要求更短的延迟,具有一定的实时需求,而目前的大部分系统都不具备该特性。提出了一种基于 ESB 平台的实时 ETL 解决方案,通过在 ESB 平台上运行的组件实现 ETL 功能,利用实时分区加载和更新实时数据。实验表明,该设计方案能够实现实时 ETL 功能,并具有良好的通用性和可扩展性。

关键词:实时 ETL;企业服务总线;数据仓库
中图分类号: TP311;TP391 **文献标志码**:A

Design and implementation of real-time ETL based on ESB

GAO Bin, GU Jian-hua, FU Ning, ZHANG Hai-hui
(*College of Computer, Northwestern Polytechnical University, Xi'an Shaanxi* 710072, *China*)

Abstract: Data warehouse technology and its applications develop promptly, accompanying which are the demands by enterprises for shorter data processing delay and real-time requirements. It is a regrettable fact that most existing systems actually fail to deliver such needed features. An Enterprise Service Bus (ESB) based real-time Extract, Transform, and Load (ETL) solution was proposed. The functionalities of ETL were realized as components running on the ESB platform. Real-time partition was created to load real-time records. Experimental results reveal that this design scheme successfully delivers the real time property and meanwhile maintains good versatility and extensibility.

Key words: real-time Extract, Transform, and Load (ETL); Enterprise Service Bus (ESB); data warehouse

0 引言

传统的 ETL 通常采用批处理的方式,一般来说是每天的夜间进行,当天的数据要到第二天才可以获得。随着数据仓库技术的逐步成熟,企业对数据仓库的时间延迟有了更高的要求,希望达到零延迟,也就出现了目前的实时 ETL[1,2]。实时 ETL 指的是将业务数据源中产生的数据实时地抽取加载到数据仓库,为数据挖掘系统、OLAP 软件、商务智能软件等决策支持系统产生更加有效的结果提供数据支持。实时 ETL 过程的目标是在业务系统的数据发生变化后,以最少的延误保持数据仓库更新。

由于目前的大部分 ETL 系统都不具备实时性[2],人们提出了多种实时 ETL 的解决方案,主要有微批处理、企业应用集成(Enterprise Application Integration, EAI)、CTF、EII 等。但是,各种解决方案都有一定的局限性,只满足于特定的场景[1]。SOA 和 ESB[2,3]的出现,为实时 ETL 的解决提出了新的思路。目前,SUN 公司在 OpenESB[4] 中就提出了一种在 ESB 中实现 ETL 的解决方案,但是该方案没有提供实时性和增量更新的支持。本文针对实时 ETL 的需求和实现方案进行了初步的研究,并提出了基于 ESB 平台的实时 ETL 的设计方案,设计并实现了通用的组件和模块,能够对各种技术和应用系统提供支持,具有很强的灵活性和扩展性。

1 实时 ETL 常用的解决方案

在构建实时 ETL 时,可选择的技术方案通常有以下几种[1]:

1)微批处理(MicroBatch ETL,MB-ETL)

微批处理的方式和我们通常的 ETL 处理方式很相似,但是处理的时间间隔要短,例如间隔一个小时处理一次。这种方式是最简单的实时 ETL 的实现方式,通过提高 ETL 的执行频率来增强系统的实时性,在对延迟时间要求不高的情况下可以采用。

2)企业应用集成(Enterprise Application Integration,EAI)

EAI 也称为功能整合,通常由中间件来完成数据的交互。对实时性要求非常高的系统,可以考虑使用 EAI 作为 ETL 的一个工具,可以提供快捷的数据交互,数据延迟可保证在 1 分钟左右。不过在数据量大时采用 EAI 工具效率比较差,而且实现起来相对复杂。

3)CTF(Capture, Transform and Flow)

CTF 是一类比较新的数据整合工具,它采用的是直接的数据库对数据库的连接方式,数据延迟可保证在 15 分钟左右。通常的处理方式是建立数据准备区,采用 CTF 工具在源数据库和数据准备区的数据库之间相连接,数据进入数据准备区后再经过其他处理后迁移入数据仓库。CTF 的缺点是只能进行轻量级的数据整合。

4)EII(Enterprise Information Integration)

EII 是另一类比较新的数据整合软件,可以给企业提供实时报表,数据延迟可以保证在 1 分钟左右。与 ETL 不同的是,EII 中的数据是"随需应变"地抽取的,联邦查询经过优化、分段又被返回所有的数据源,而结果则汇入数据源的虚拟

收稿日期:2007 - 10 - 16;**修回日期**:2007 - 12 - 20。
基金项目:国家发改委高技术产业化基金资助项目(20052139);国家 863 计划项目(2006AA0121626);十一五国防预研项目(06004089)。
作者简介:高彬(1981 -),男,山东威海人,硕士研究生,主要研究方向:分布式计算; 谷建华(1965 -),男,陕西西安人,教授,主要研究方向:分布式计算、嵌入式系统技术; 符宁(1976 -),男,陕西西安人,博士研究生,主要研究方向:分布式计算、可信计算; 张海辉(1977 -),男,湖南娄底人,博士研究生,主要研究方向:分布式计算、网格计算。

视图。EII 工具是“访问”而不是“移动” 数据。

本文提出的实时 ETL 的解决方案充分借鉴了 EAI 的实现方式的优点,采用 ESB 作为实时 ETL 的实现平台,以 ESB 平台的组件的形式分别实现抽取、转换、清洗和装载功能,以 ESB 中的流程形式表现 ETL 过程。这样的设计具有很好的灵活性和扩展性。

2　系统体系结构

2.1　ESB 平台简介

本文所使用的 SYNPUESB 平台主要由适配器组件、管理控制组件、JMX、JMS 通信总线,以及正规消息路由(Normal Message Route, NMR)模块组成。

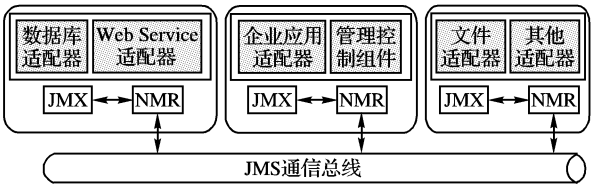

图 1　SYNPUESB 架构

适配器的作用[5]是将已有系统中的业务逻辑和业务数据包装成企业服务总线支持的协议和数据格式。通过企业服务总线,这些被包装起来的业务逻辑和数据就可以方便地参与上层的业务流程,从而已有应用系统的能力可以得以继续发挥。这里的已有应用包括遗留应用、各种企业数据存储和各种 Web 服务。

NMR 是 ESB 系统的核心,完成 ESB 系统的消息路由功能。NMR 从适配器接收消息并将其路由到适当的组件进行处理。ESB 系统使用 JMX 机制进行管理。ESB 实现提供特定的 JMX 管理 Beans(MBeans)管理 ESB 系统提供的基础设施以及组件。

2.2　实时 ETL 设计

本文实现的解决方案以 SYNPUESB 作为 EAI 中间件,将传统的 ETL 只有一道抽取转换装载过程分解,以 SYNPUESB 组件的形式实现 ETL,每个组件完成 ETL 中的一个功能,每个组件有其自己的接口定义,实现单独的功能。同时,在静态的常规的数据仓库外建立一个实时分区,用来加载存储处理实时的数据更新。实时 ETL 流程从应用数据库将实时数据抽取出来,通过清洗转换后存放于实时分区。用户可以通过访问实时分区获取实时数据。实时分区只用来存取当天的数据,每晚定时以批处理的形式将数据加载到静态数据仓库,然后清空实时分区。

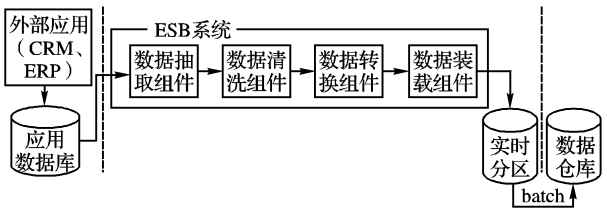

图 2　实时 ETL 系统结构

本文提出的实时分区解决方案分为以下几部分:

1)实时分区

在数据仓库中建立一个实时分区,用来加载存储处理实时的数据更新。Ralph Kimball 指出,实时分区是数据仓库执行处理实时任务的一个务实可行的方案[1]。

2)基于 XML 的数据库信息表示

在数据的抽取和装载时,需要解决 XML 格式与关系数据库中的记录的表示映射的问题,需要设计对表的记录的 XML 表示方法。基于模板的映射和基于模型的映射是两种常见的 XML 到数据库的映射方法[5]。这里采用基于模型的映射,这也是目前主流转换软件使用的方法。

3)基于组件的实时 ETL 模块设计

将传统的 ETL 的数据抽取转换装载流程分解为数据抽取、多次的数据清洗、数据转换和数据装载过程,系统中的组件包括实时数据抽取组件、数据转换组件、数据过滤组件和数据装载组件。

数据抽取组件　用户通过设置它来直接连接源数据库,当将组件与数据库连通时,就可以读出连接数据源中所包含的所有表以及表中的元数据结构,用户设置自己需要抽取的内容,并且提供接口以供数据转换组件获取抽取出的数据的格式。数据条件设置方面,在与源数据相连的每一个转换链中有一个提取数据设置页,用户输入对可以输出数据的要求。对于在不同数据库系统的通用访问技术,使用 JDBC 数据连接方案来操纵不同的数据库,实现异种数据库的数据转换的通用性。在对数据增量的实时抽取方面,在组件启动时在源数据库建立需要的触发器和辅助表,及时采集数据库中的变化。

数据清洗组件　由于从数据源抽取出来的数据可能存在着“脏数据”[6],因而需要在数据转换前先进行数据清洗。考虑到由于数据清洗规则经常发生变化,这要求基于规则的数据清洗技术的实现必须能灵活地定义规则和调整规则,因而采用规则引擎 drools 是个很有效的解决手段[7]。由于组件间传递的数据是通过 XML 封装的数据,数据过滤的方式是基于消息内容的,因而规则的设置是用户使用 XPATH 来动态地添加规则和变量。例如,XML 信息通过设置填写 XPATH 规则:

```
/employee/Record[ ID! = '' and NAME ! = '' and AGE ! = '']
```

生成规则文件如下:

```
<rule name = "RuleNum0" >
  <java: condition > XPathContain
    ("/employee/Record[ ID! = '' and NAME ! = '' and
    AGE ! = '']", exchange)
    </java: condition >
  <java: consequence >
    jbi. forwardToService
      ("http: //servicemix. org/cheese", "myScreenOutput 1");
  </java: consequence >
</rule >
```

该规则文件定义了一条规则(<rule>标记),其中规定了匹配条件(<java:condition>标记)。当消息中的数据满足匹配条件,则触发执行<java:consequence>标记中的 Java 代码,将消息发送到指定的下一个组件,这样可以将数据项中含空值的记录清洗掉。

数据格式转换组件　数据抽取组件抽取出的数据是以 XML 格式封装了的数据,展现的是源数据端的数据格式,数据装载组件操作的数据是与目标数据库格式相一致的数据,由于目标数据库的数据格式与源数据库的数据格式不一定相同,因而需要对格式进行转换,将一种数据库系统中定义的 XML 模型转化为另一种数据库中的 XML 模型,笔者采用 XSLT 技术实现。数据格式转换组件通过调用前端和后端的组件接口,获取它们的数据格式,进而通过图形化展示,用户可以通过连线来方便灵活地设置数据格式间的转换。

数据装载组件　用于将前端组件传输过来的数据存储进数据库。数据装载组件在用户配置时连接目标数据库,获取连接数据源中包含的所有表以及表中的元数据结构,并且提

供接口以供数据转换组件获取数据装载组件需要的数据的格式。数据装载组件接收到消息后,通过解析 XML 消息,将消息中的数据装载进数据库。

3 性能测试

3.1 测试方案

基于以上设计思想,我们对实时 ETL 做了实例验证。

使用 SYNPUESB 流程编辑器(SYNPUESB Process Orchestration, SPO)设计一个数据集成流程。数据抽取组件从源数据库捕获到实时数据,首先通过数据清洗组件将空数据清除,然后经过数据转换组件转换,数据装载适配器将数据写入在目标数据库中的实时分区中。测试的源数据库表的列数为6,测试目标表的列数为4。

3.2 测试环境

源数据库 SQL Server 2000 服务器:192.9.200.83;

目标数据库 Oracle 服务器: 192.9.200.88;

JMS 服务器:192.9.200.78;

网络:100 Mbps 局域网;

ESB 平台地址:192.9.200.80。

3.3 测试数据结果

数据的延时时间主要受两方面因素影响:其一,是数据抽取组件端发现数据的时间。数据抽取组件通过触发器写进临时表,然后通过轮询将数据提取出来。因而,轮询时间会对数据的延时时间产生影响。其二,是 ESB 平台中实时 ETL 流程对数据的处理时间。

1)系统吞吐量测试

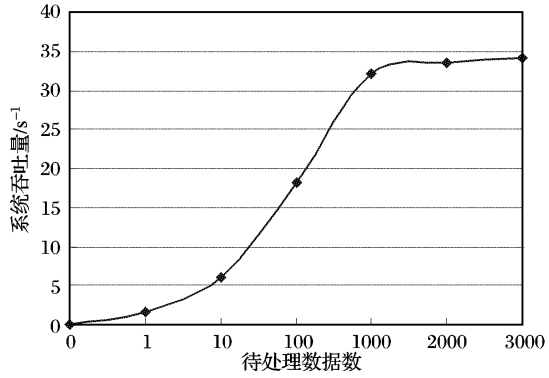

图3 系统吞吐量与数据条数关系

在一次处理少量的数据(1~10 条)时,系统的吞吐量很小;随着数据量的增大(10~1 000 条),系统的吞吐量迅速增大;而数据量达到 1 000 条之后,系统吞吐量达到最大。

系统中存在瓶颈,导致系统的吞吐量只能达到大约 35 条/s。分析瓶颈存在的主要原因:其一,组件对消息的处理速度,由实验可知,对单条消息,所有组件的处理时间之和大约是 0.5 s;其二,消息中只有一条记录。对系统的性能改进主要可以通过增加消息中记录的条数来实现,若组件可以一次处理多条记录,则吞吐量会有一定提升。

2)待处理记录为 3 000 条,测试消息中的记录条数对系统吞吐量的影响。

消息中的记录条数对系统吞吐量有很大影响,随着每条消息中记录数的增加,吞吐量随之以曲线上升。

3)插入记录 1 000 条,平均数据插入时间间隔0.03 s,测试轮询时间对数据处理的总时间(从第一条记录进入源数据库到最后一条记录插进目标数据库的时间间隔)的影响。

系统处理的总时间与轮询间隔相关,随着轮询间隔的增加,系统处理时间也会线性增加。

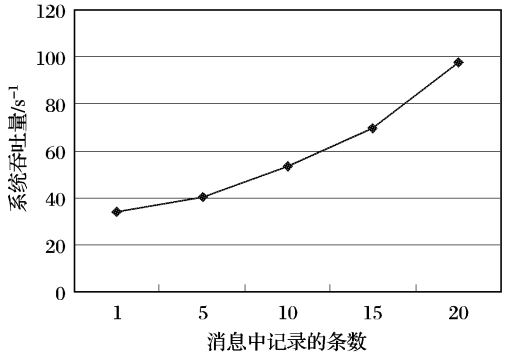

图4 消息中的记录条数对系统吞吐量的影响

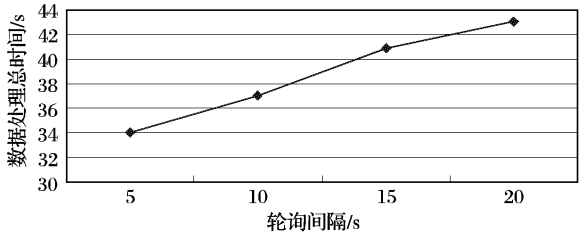

图5 处理时间与轮询间隔关系

4)轮询时间设为 10 s,平均插入数据间隔时间 0.03 s,测试系统的数据延迟时间(记录进入源数据库到插进目标数据库的平均时间间隔)。

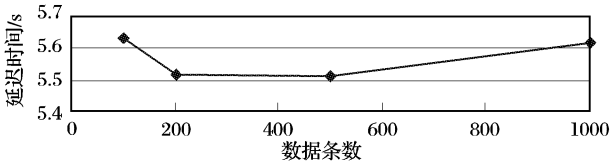

图6 数据延迟时间与数据条数关系

数据延迟时间与数据条数无关,当轮询时间为 10 s 时,延迟时间不超过 6 s。

试验表明,基于 ESB 平台设计建立的实时 ETL 系统可以在数据量要求不是特别大的情况下完成实时 ETL 功能,数据延迟可保证在 6 s 左右。

4 结语

本文提出了一种在 ESB 平台下的实时 ETL 实现方案。通过使用 XML 统一数据格式,并采用 ESB 平台的组件形式运行,使其具有较好的通用性、灵活性和可扩展性。实践证明它能够完成实时 ETL 的任务要求,具有良好的应用前景。

参考文献:

[1] KIMBALL R, CASERTA J. Data Warehouse ETL Toolkit[M]. Indianapolis: Wiley Publishing, 2004.

[2] 许力,马瑞新. 基于 SOA 的实时 ETL 的研究与实现[J]. 计算机系统应用,2007(8):24-27.

[3] SATAPATHY A, SRINIVASAN R, APTE N. ETL Integrator[EB/OL].[2007-08-08]. http://wiki.open-esb.java.net/jbiwiki/Wiki.jsp?page=ETLSE.

[4] 李晓东,杨扬,郭文彩. 基于企业服务总线的数据共享与交换平台[J]. 计算机工程,2006,32(21):218-223.

[5] 顾天竺,沈洁,陈晓红, 等. 基于 XML 的异构数据集成模式的研究[J]. 计算机应用研究,2007,24(4):94-97.

[6] 郭志懋, 周傲英. 数据质量和数据清洗研究综述[J]. 软件学报,2002,13(11):2076-2081.

[7] 叶舟,王东. 基于规则引擎的数据清洗[J]. 计算机工程,2006,32(23):52-54.