ODS 中增量 ETL 过程的 CSP 模型研究*

熊安萍, 苏 磊, 蒋 溢
(重庆邮电大学 计算机科学与技术学院, 重庆 400065)

摘 要: 增量 ETL 过程的并行化是提高 ODS 数据实时性的有效途径。结合通信顺序进程理论研究了增量 ETL 过程模型,形式化分析了增量 ETL 过程事件在并行环境下执行状态的变换过程,提出了增量 ETL 过程并行调度算法,解决了增量 ETL 过程在并行环境下调度策略的问题。应用及实践表明,模型及算法具有源系统负载小、数据的实时性高等特点。

关键词: 通信顺序进程; 操作数据存储; 增量 ETL; 并行调度
中图分类号: TP311.13 **文献标志码**: A **文章编号**: 1001-3695(2012)09-3330-04
doi:10.3969/j.issn.1001-3695.2012.09.034

ODS in incremental ETL process of CSP model

XIONG An-ping, SU Lei, JIANG Yi
(*College of Computer Science & Technology, Chongqing University of Posts & Communications, Chongqing* 400065, *China*)

Abstract: Parallel incremental ETL process is to improve the ODS data real-time effective way. Combine with communicating sequential processes theory of the incremental ETL process models, formal analysis of the incremental ETL process events in parallel execution environment, and the state of transformation process, this paper proposed the incremental ETL process proposed parallel scheduling algorithm to solve the incremental ETL process a parallel scheduling policy environment in question. Application and practice show that the model and algorithm with the source system load is small, and high real-time data.

Key words: CSP; ODS; incremental ETL; parallel scheduling

0 引言

ODS(operational data store,操作型数据储存)是数据仓库体系结构中的重要组成部分,是面向主题的、集成的、当前或接近当前的、不断变化的数据[1]。ODS 数据获取方式包括全量 ETL 过程和增量 ETL 过程[2]。全量 ETL 过程一般用于 ODS 的初始化,而增量 ETL 过程则用于 ODS 数据维护。高效的增量 ETL 过程是 ODS 提供跨系统准实时数据支撑的基础和关键,而增量 ETL 过程的并行化是提高 ETL 性能的有效途径。因此研究增量 ETL 并行过程中的调度等问题具有重要价值。

并行增量 ETL 过程指将顺序执行的增量 ETL 工作流程转换为多个并行处理工作流程的过程。文献[3,4]采用并行结构解决了 ETL 过程中多数据流并行处理的问题,降低了响应时间,提高了吞吐量,适合于处理全量 ETL 过程。文献[5]利用已有的物化视图增量维护的方法,提出了一种增量 ETL 过程的自动化产生方法,简化了增量 ETL 过程的实施。文献[6]提出了一种基于传播的不准确的安全更新和魔集优化状态关系的增量 ETL 过程控制方法,显著地提高了增量过程的性能。分析前人对增量 ETL 过程并行化的研究工作可以看出,目前对增量 ETL 过程并行化的研究主要集中于增量数据流的产生方法,对并行环境下增量 ETL 过程中系统内交互模型的研究较少。而增量 ETL 过程并行化下无序的交互调度往往会产生非确定性的脏数据,或因异常调度产生大量冗余操作而对系统性能产生较大的损耗。为此,本文首先结合通信顺序进程(communicating sequential processes,CSP)理论,建立了一种基于 CSP 的并行增量 ETL 过程模型;其次,利用 CSP 描述模型分析了增量 ETL 过程执行的时序和逻辑关系,研究了增量 ETL 过程执行的相关逻辑状态,提出了一种增量 ETL 过程调度算法,解决了增量 ETL 工作流的并发调度策略及调度时产生的死锁等问题;最后,举例说明了本文所述增量 ETL 方法,并通过实验测试了本文方法对源系统的性能损耗和增量数据的实时性。

1 增量 ETL 过程

增量 ETL 过程的主要功能是按照 ODS 数据映射关系,将数据源中相关数据的变化情况经过若干整合处理,并产生相应 ODS 概念模型的增量实例数据,以实现 ODS 与源系统间数据同步的过程。增量 ETL 过程主要包括抽取、加载和转换等三个主要步骤。

1)抽取 数据源监控子进程以异步方式来捕获变化的数据。当源表 $table_i$ 发生变化时,增量数据以 STREAMS 方式提交到 DBMS,源表 $table_i$ 的监控子进程 $EProcess_i$ 通过数据库 REDO-LOG 捕获数据源的变化情况,并发送增量数据至转换子进程 $TProcess_i$,以完成增量数据抽取过程。

2)转换 转换子进程 $TThread_i$ 接收到增量数据后,按照 ODS 映射协议对增量数据进行转换。映射协议一般包括如下转换操作:$P=(\pi,\sigma,\bowtie)$。其中 π 指投影操作,属于一元操作,表示选择出子进程 $EProcess_i$ 捕获增量数据的若干属性列进入

收稿日期: 2012-01-11; **修回日期**: 2012-03-15 **基金项目**: 国家自然科学基金资助项目(61003247)

作者简介: 熊安萍(1970-),女,四川泸县人,副教授,硕士,主要研究方向为面向服务计算、分布式计算与并行处理(xiongap@cqupt.edu.cn);苏磊(1987-),男,硕士研究生,主要研究方向为数据仓库;蒋溢(1969-),男,副教授,硕士,主要研究方向为计算机网络、软件体系结构.

$table_j$;σ 指选择操作,属于一元操作,表示选择满足条件的记录;⋈指自然连接,属于二元操作,表示将增量数据通过外键与相关属性进行关联,以保证增量数据的语义完整性。转换子进程将转换操作 P 处理后的增量数据发送至加载子进程,以完成增量数据的转换过程。

3)加载　加载子进程 $LProcess_i$ 接收到转换后的增量数据,按维表与事实表的依赖及约束关系以时序逻辑加载增量数据至 ODS 的维表和事实表中,以完成 ODS 实时数据同步。STAR = (D,F) 分别是维表和事实表的集合,$D = \{D_1, D_2, \cdots, D_v\}$,$F = \{F_1, F_2, \cdots, F_w\}$,维表与事实表依赖、约束关系可表示为偏序集$\langle D_1, F_1, F_2, D_2, \cdots \rangle$。在增量 ETL 过程中,加载进程需按照偏序关系解决异步进程并发时通信和同步等问题,以实现增量数据的按序更新,以防发生更新丢失或脏数据等问题。

2　增量 ETL 过程模型的 CSP 描述

CSP 是 Hoare[7] 提出的一种基于进程代数的形式化建模方法,它可用来分析并发、异步、非确定等系统行为,并描述并行系统间进行交互的模式,具有良好的语义与可扩展性。CSP 通常以进程示客观事物的行为模型,以进程的字母表表示进程参与事件的集合,以进程的迹表示进程所允许的事件序列。

根据 CSP 理论,并行增量 ETL 过程是一种通过并行转换管道进行的数据交互过程,其模型如图 1 所示。它监控源系统的数据变化,并将捕获的增量数据发往内部管道(发送端);在管道传输前,增量数据按照 ODS 映射协议完成数据转换过程,然后将整合后的增量数据传输给 ODS 端(转换协议);ODS 端在收到增量数据后根据维表与事实表依赖、约束关系完成加载过程(接收端)。

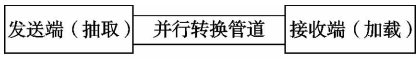

图1　增量ETL过程的CSP管道模型

为描述并行增量 ETL 过程,本文定义如下:

定义 1　增量 ETL 过程发送端、转换管道和接收端的每个动作称为一个 CSP 事件。

定义 2　增量 ETL 过程中可见事件的有限序列称为进程的迹。

并行增量 ETL 过程由多个事件以特定的时序和逻辑协同完成,所有的事件工作需在进程的迹内。这些事件分别由数据捕获进程 EProcess、数据转换进程 TProcess 和数据加载进程 LProcess 三类 CSP 进程完成。

不妨设有 n 个数据发送子进程 $SCP_i(i = 1,2,\cdots,n)$ 向数据转换子进程 $CCP_k(k = 1,2,\cdots,m)$ 发送增量数据 $data(i,t) = \{type, newRecord, oldRecord\}$。其中:type 表示数据源发生的更新操作类型,包括 insert、update 和 delete 等三种操作;newRecord表示数据源更新后的记录;oldRecord 表示数据源发生变化前的旧记录,若为 insert 操作,则该字段为空。数据通道进程 CCP_k 收到 $data(i,t)$ 后检查映射关系 $map(i,k)$,若表 $table_i$ 需和表 $table_j$ 进行关联转换,则向 SCP_j 查询 t 时刻源表 $table_j$ 的数据,SCP_j 向 CCP_k 返回 $result(j,t)$;CCP_k 根据映射协议 $P = (\pi, \sigma, \bowtie)$ 组合增量数据并进行传输。数据接收进程 RCP_p 按照时态逻辑加载增量数据,RCP_p 接收 CCP_k 转换的增量数据 $data(k,t)$,若 RCP_p 增量数据 $data(k,t-1)$ 尚未处理,则 RCP_p 需要向 CCP_k 请求 $request(k,p,t-1)$,并将 $data(k,t)$ 置于等待队列,等候 $data(k,t-1)$ 处理完成后,按照维表与事实表依赖、约束关系可表示为偏序集$\langle D_1, F_1, F_2, D_2, \cdots \rangle$加载增量数据。$CCP_k$ 收到 RCP_p 发送请求 $request(k,p,t-1)$,需提高 $data(k, t-1)$ 的优先级或重发。综上所述,三类 CSP 进程的外部事件集可描述为

$$\alpha(S) = \cup_{i=1,2,\cdots,n}(s_i)$$
$$s_i = \{data(i,t), result(j,t)\};$$
$$\alpha(C) = \cup_{i=1,2,\cdots,n}(c_i)$$
$$c_i = \{query(i,t), join(i,j,t), send(i,t), addPrior(k,p,t-1), wait(j,t)\};$$
$$\alpha(R) = \cup_{i=1,2,\cdots,n}(r_i)$$
$$r_i = \{request(k,p,t-1), receive(i,j,t), load(i,j,t), wait(i,t-1)\};$$

根据以上事件集,并行增量 ETL 过程用 CSP 可描述为

$$P = S \parallel C \parallel R$$
$$S = s_1 \parallel s_2 \parallel \cdots \parallel s_m$$
$$C = c_1 \parallel c_2 \parallel \cdots \parallel c_m$$
$$R = r_1 \parallel r_2 \parallel \cdots \parallel r_m$$
$$s_i = (change(i,t)?\ y{:}data(i,t) \rightarrow c_j) \square query(i,t)?\ y{:}result(i,t) \rightarrow c_j$$
$$c_j = (send(j,t)!\ map(i,k)?\ j \rightarrow result(j,t) \rightarrow jain(i,j,t)) \rightarrow data(i,t) \square (receive(j,p,t)?\ y{:}addPrior(j,p,t-1)) \square wait(j,t)$$
$$r_i = (receive(i,j,t)?\ y{:}data(i,t) \rightarrow c_j) \square (query(i)?\ y{:}result(i,t) \rightarrow c_j)$$

3　增量 ETL 过程执行状态描述

受增量 ETL 过程各事件间数据交互关系的影响,增量 ETL 过程事件在执行过程中存在复杂的通信行为。根据这些事件的行为,本文定义如下:

定义 3　在增量 ETL 过程事件集中,事件 A 和 B 存在执行先后顺序,如只有事件 A 执行完毕,事件 B 才可能被执行,称事件 A 为事件 B 的先驱,事件 B 为事件 A 的后继,事件 A 和事件 B 间的顺序关系称为约束。

定义 4　增量 ETL 过程事件 A 定义成一个与时序和逻辑相关的函数 $state(t) = \{begin, block, run, complete\}$,即事件 A 在每个执行周期内的状态分为开始、阻塞、运行和完成四种。

在增量 ETL 执行过程中,当事件 A 接收事件请求消息后,事件 A 根据约束触发进程 $Process_i$ 执行。进程 $Process_i$ 的执行用 CSP 可表述为

$$Process_i \rightarrow Block() \square Run()$$

若进程 $Process_i$ 不需要等待其他事件执行,则立即响应请求,完成请求动作;否则,进程 $Process_i$ 发送事件请求消息,等待先驱事件执行完毕后触发完成。设 $P(in, out)$ 是一个增量 ETL 过程 CSP 进程,它接收阻塞的事件请求消息,并输出前驱事件请求消息。该进程用 CSP 可表述为

$$P(in,out) = in?\ x \rightarrow wait \parallel out \rightarrow req$$

综上所述,根据增量 ETL 过程事件状态的描述,对于 $\forall Process_i \in P(in,out)$,$Process_i$ 执行进程的状态包括:

a)若事件 $event(i,t)$ 发生,则 $Process_i$ 的状态函数 $state(t) = begin$,$Process_i$ 进入可调度任务列表。

b)若 $Process_i$ 的状态函数 $state(t) = begin$,且事件 $event(i,$

t)的前驱事件已执行完毕,则 $Process_i$ 的状态函数 state(t) = run,$Process_i$ 进入执行队列。

c)若 $Process_i$ 的状态函数 state(t) = begin,但事件 event(i,t)的前驱事件尚未全部执行完毕,则 $Process_i$ 的状态函数 state(t) = block,$Process_i$ 进入等待队列。

d)若 $Process_i$ 执行完毕,则 $Process_i$ 的状态函数 state(t) = complete;对于任意后继事件进程 $Process_j$,重新检查其执行条件,若 $Process_j$ 前驱事件已执行完毕,$Process_j$ 的状态函数 state(t) = run,否则,$Process_j$ 的状态函数 state(t) = block。

4 增量 ETL 过程调度算法

为分析增量 ETL 过程调度策略,可构造有向图 $G(V,E)$。其中,$V=\{V_i \mid V_i$ 为增量 ETL 过程执行中的节点,$0\leqslant i\leqslant n\}$;$E=\{e(i,j) \mid e(i,j)$为 V_i 到 V_j 的有向边,表示两个节点间的约束关系,$0\leqslant i\leqslant n, 0\leqslant j\leqslant n$,且 $i\neq j\}$。为保证增量 ETL 过程的可调度性,需对调度过程进行死锁检测,检测算法可描述为算法 1。

算法 1 增量 ETL 过程节点死锁检测算法

输入:增量 ETL 过程调度约束关系图 $G(V,E)$;i 表示待检测的节点编号。

输出:待检测节点是否死锁。

算法步骤:

```
L1   visited = ∅;
L2   trace = ∅;
L3   Function findCycle(i){
L4       if(i ∈ visited) {
L5           if(i ∈ trace){
L6               return true;}
L7               return false;}
L8       visited = visited∪i;
L9       push(trace,i);
L10      for each j in V {
L11          if(E[i][j] = 1)
L12              findCycle(j);}
L13      pop(trace);
L14      return false; }
end
```

在算法 1 的描述中,trace 为一个栈,记录了出发节点到当前节点的轨迹,visited 为标志已访问的节点。L4 判断节点是否进行过访问操作,若已访问则,L5 ~ L7 判断节点是否包含于 trace 中,若包含,则表明该节点构成了死锁;若未访问,L8 标记该节点为已访问,L9 将该节点加入 trace,L10 ~ L12 递归访问 i 节点的相邻节点是否构成死锁,L13 用于递归过程中删除 trace 最末端节点的轨迹,以控制递归条件。

为保证增量 ETL 过程调度过程中的时序和逻辑关系,本文的增量 ETL 过程调度过程如算法 2 所示。

算法 2 并行增量 ETL 事件调度算法

输入:增量 ETL 过程事件 event(i,t),其中 i 表示事件节点编号,t 为时序;增量 ETL 过程调度约束关系图 $G(V,E)$。

输出:事件 event(i,t)相关事件的状态。

算法步骤:

a)调用 findCycle(i)函数检测事件 event(i,t)是否会引起死锁,若存在死锁,则停止调度;否则转向步骤 b)。

b)若事件 event(i,t)阻塞事件队列 $blockList_i$ 为空,则执行事件 event(i,t),直至事件结束更新其状态 event(i,t). state = complete,转向步骤 c);否则更新其状态 event(i,t). state = block,转向步骤 c)。

c)遍历与事件 event(i,t)所属节点 i 相关子节点 j,通知 j 节点相关事件 event(j,t)当前节点 i 的状态 event(i,t). state,若事件 event(i,t). state = complete 时,在 j 节点的阻塞队列 $blockList_j$ 中删除事件 event(i,t),并触发事件执行;否则该事件继续阻塞。

5 应用及性能分析

5.1 应用案例

为验证本文提出的增量 ETL 过程调度策略逻辑正确性,本文考虑一个简单的增量 ETL 过程的例子。案例抽取 S1 用户下的 customer、city 表和 S2 用户下的 order 表,并对三张表的数据进行转换,整合为 orderData 表,最后将 orderData 表的数据加载至 ODS 的事实表和维表。案例所述 ETL 过程数据映射关系如图 2 所示。

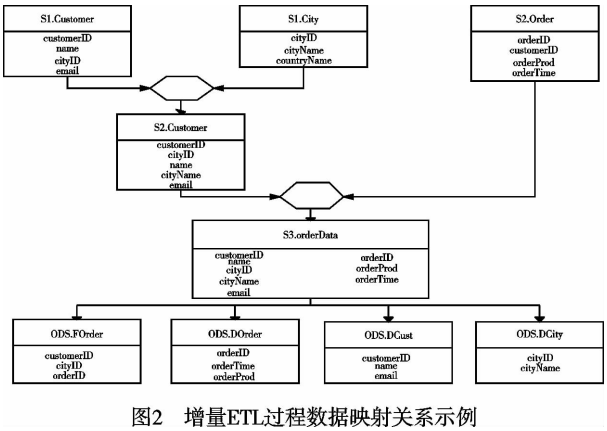

图2 增量ETL过程数据映射关系示例

假设源系统在 t_1 时刻和 t_2 时刻分别修改了某个订单的客户和订购产品信息,即分别更新了 S1. Customer 的客户数据 data(1,t_1)、data(1,t_2)以及 S2. Order 表的订购产品数据 data(2,t_1)、data(2,t_2),其中 $t_1 < t_2$。根据时序关系,data(1,t_1)和 data(2,t_1)必须在 data(1,t_2)和 data(2,t_2)之前;根据逻辑关系,data(1,t_1)加载过程应同步于 data(1,t_2)加载过程。因此增量 ETL 过程行为可描述为:

a)增量数据监控子进程 $EProcess_1$ 和 $EProcess_2$ 分别捕获源表的变化数据并发往转换进程 $EProcess_1$,增量 ETL 过程转入下一阶段由转换线程 $TProcess_1$ 执行,该阶段增量 ETL 进程的迹可表示为

$$\langle data(1,t_1), data(2,t_1), data(1,t_2), data(2,t_2), trace(TProcess_1)\rangle$$

b)若增量数据转换子进程接收到 t_2 时刻的增量数据 data(1,t_2)或 data(2,t_2),且 data(1,t_1)和 data(2,t_1)已加载完毕,即执行节点对应的阻塞队列为空,则线程加载数据,并将加载后的数据发往加载线程 $LProcess_1$,该阶段增量 ETL 进程的迹可表示为

$$\langle data(1,t_2), data(2,t_2), trace(LProcess_1)\rangle$$

c)若增量数据转换子进程接收到 data(1,t_2)或 data(2,t_2),但 data(1,t_1)或 data(2,t_1)尚未全部加载完毕,即执行节点对应的阻塞队列不为空,则 data(1,t_2)或 data(2,t_2)加载阻塞,该阶段增量 ETL 进程的迹可表示为

〈data(1,t_2),data(2,t_2),block〉

d)若增量数据转换子进程接收到增量数据为 data(1,t_1)或 data(2,t_1),则转换子进程根据数据映射规则进行数据转换,并将加载后的数据发往加载子进程 $LProcess_1$,该阶段增量 ETL 进程的迹可表示为

〈data(1,t_1),data(2,t_1),trace($LThread_1$)〉

e)加载线程 $LThread_1$ 根据事实表与维表的约束关系加载增量数据,该阶段增量 ETL 进程的迹可表示为

〈$LProcess_1$〉

综上可得增量 ETL 过程的迹为:

〈〈data(1,t_1), data(2,t_1), data(1,t_2), data(2,t_2), trace($TThread_1$)〉,〈data(1,t_2),data(2,t_2),trace($LProcess_1$)〉,

〈data(1,t_2),data(2,t_2),block〉,

〈data(1,t_1),data(2,t_1),trace($LProcess_1$)〉,〈$LProcess_1$〉〉

根据算法 2,案例所述增量 ETL 过程进程调度状态序列如表 1 所示。根据表 1 的调度状态序列可以看出,本文的方法可完成增量 ETL 过程在并行环境下的调度问题,本文方法有效。

表 1　增量 ETL 过程调度状态序列

序列序号	调度状态序列
1	($EProcess_1$.state(t) = begin→$EProcess_1$.state(t) = run→ $EProcess_1$.state(t) = complete) ‖ ($EProcess_2$.state(t) = begin→ $EProcess_2$.state(t) = run→$EProcess_2$.state(t) = complete)
2	$TProcess_1$.state(t) = begin→$TProcess_1$.state(t) = block→ wait($TProcess_1$.state(t) = complete and $Tprocess_2$.state(t) = complete)→$TProcess_1$.state(t) = run→$TProcess_1$.state(t) = complete
3	$LProcess_1$.state(t) = begin→$LProcess_1$.state(t) = block→ wait($TProcess_1$.state(t) = complete)→$LProcess_1$.state(t) = run→$LProcess_1$.state(t) = complete

5.2　系统性能

为测试本文提出的增量 ETL 过程方法的性能,本文分别从增量 ETL 过程对服务器的负载和增量 ETL 过程的实时性两个方面进行了测试。

实验 1　增量 ETL 过程服务器负载实验

在增量 ETL 过程中,对源系统的性能损耗是衡量增量 ETL 方法可行性的重要标准。为测试本文所述方法对源系统的性能损耗,本文在 HP-RX8640 服务器上测试。测试服务器配置 16 * 1.6 GB CPU 以及 64 GB 内存,测试过程分别对 200 张以 100 条/min · 表、500 条/min · 表、1 000 条/min · 表、2 000 条/min · 表、5 000 条/min · 表及 10 000 条/min · 表的频率输出增量数据,统计增量 ETL 过程运行情况下和未运行情况下系统的负载。参照文献[8]提出的事务处理服务器的性能评价方法,以 5 000 个用户并发执行 24 h 内处理的事务数作为系统负载的评价指标,实验结果如表 2 所示。测试结果表明,在业务峰值下系统对源系统的性能损耗在 5% 以内。因此,本文的增量 ETL 过程对源系统的负载较小,基本不会影响源系统的业务操作,方法具有较强的可行性。

实验 2　增量 ETL 过程实时性实验

为保证 ODS 为下游系统提供准实时性的数据支撑,数据的实时性是衡量增量 ETL 过程的另一个重要指标。为测试本文所述并行框架下增量 ETL 过程方法的实时性,本文在两台 HP-RX8640 服务器上进行测试,服务器配置同实验 1。在测试系统中,分别更新 200 张源表的 100、1 000、2 000、5 000、10 000 和 20 000 条记录,并对其进行增量 ETL 过程,使其同步至 ODS。实验重复 100 次,每表每次所有增量数据同步至 ODS 的耗时数据如图 3 所示。测试结果显示,在系统每次产生 20 000 条增量记录的情况下,增量 ETL 过程约在 90 s 内可完成数据同步的过程。由此可以看出,与传统的用后台 Job 和存储过程方式每天或每月定时执行数据同步过程的处理方法相比,本文的增量 ETL 过程在实时性要求上都有很大的提高和优化,基本可满足 ODS 为下游业务系统准实时性的业务支撑。

表 2　不同加载速率下系统负载情况统计

速率	未运行增量 ETL 过程下每用户平均每小时源系统处理事务数	运行增量 ETL 过程后每用户平均每小时源系统处理事务数	损耗/%
100 条/min · 表	9 270.93	9 259.81	0.12
500 条/min · 表	9 270.93	9 240.34	0.33
1000 条/min · 表	9 270.93	9 191.22	0.86
2000 条/min · 表	9 270.93	9 125.38	1.57
5000 条/min · 表	9 270.93	9 022.47	2.68
10000 条/min · 表	9 270.93	8 967.84	3.27
20000 条/min · 表	9 270.93	8 855.65	4.48

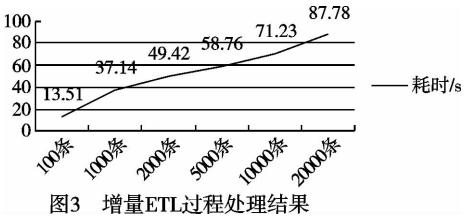

图3　增量ETL过程处理结果

6　结束语

增量 ETL 过程的可行性和运行效率是一个 ODS 建设的决定性因素,它直接影响着 ODS 项目的成功与否[9]。本文基于 CSP 理论,分析了并行增量 ETL 过程,并研究了并行增量 ETL 调度策略问题。实验证明,本文的增量 ETL 方法具有如下特点:a)对源系统的负载较小;b)数据的实时性高。下一步工作将重点关注 ETL 过程的可伸缩性,在保障可靠性和稳定性的前提下,不仅使 ETL 过程可运行于高性能主机上,而且可运行于普通 PC 机上,减少企业的硬件投入。

参考文献:

[1] 张磊,钟勇. 一种新型数据仓库体系的实现[J]. 计算机应用,2003,23(10):111-113.

[2] 徐俊刚,裴莹. 数据 ETL 研究综述[J]. 计算机科学,2011,38(4):15-20.

[3] 许力,牟晓光,马云存. 并行 ETL 过程的研究与实现[J]. 计算机工程与应用,2009,45(13):170-172.

[4] SIMITSIS A,GUPTA C,WANG Song,*et al*. Partitioning real-time ETL workflows[C]//Proc of the 26th International Conference on Data Engineering Workshops. Washington DC: Computer Society, 2010: 159-162.

[5] 张旭峰,孙未未,汪卫,等. 增量 ETL 过程自动化产生方法的研究[J]. 计算机研究与发展,2006,43(6):1097-2003.

[6] BEHREND A, BEHREND A. Optimized incremental ETL jobs for maintaining data warehouses[C]//Proc of the 14th International Database Engineering & Applications Symposium. New York: ACM, 2010: 216-224.

[7] HOARE C A R. Communicating sequential processes[M]. London: Prentice-Hall International,1985.

[8] 谷长勇. 事务处理服务器的性能评价研究[D]. 北京:中国科学院研究生院(计算技术研究所),2006.

[9] STRANGE K H. ETL was the key to this data warehouse's success, CS-15-3143[R]. Stamford:Gartner,2002.