供应链环境下一种分布式 RFID 发现服务

赵 文[1,3],李信鹏[2,3],刘殿兴[2,3],张世琨[1,3],王立福[1,3]
(1.北京大学软件工程国家工程研究中心,北京 100871; 2.北京大学信息科学技术学院,北京 100871;
3.北京大学信息科学技术学院软件研究所高可信软件技术教育部重点实验室,北京 100871)

摘 要: 本文针对大规模 RFID 应用和企业对其 EPCIS 进行完全的访问控制的需求,对 EPCIS 事件重新建模使之能够描述供应链活动所产生的绝大部分事件,并提出一种新的分布式 RFID 发现服务.这种发现服务基于"跟踪供应链"模式,利用编码解析服务(ONS),在发起查询时采用多个查询流以提高查询效率,在返回结果时并行地直接返回给客户端以缩减路由跳数.实验表明这种分布式 RFID 发现服务具有较高的效率和可用性.

关键词: 无线射频识别 (RFID); 供应链; 分布式; 发现服务; EPC 信息服务
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2010) 2A-099-08

A Distributed RFID Discovery Service for Supply Chain

ZHAO Wen[2,3], LI Xin-peng[1,3], LIU Dian-xing[1,3], ZHANG Shi-kun[2,3], WANG Li-fu[2,3]
(1. *School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871, *China*;
2. *National Engineering Research Center for Software Engineering, Peking University, Beijing* 100871, *China*;
3. *Key laboratory of High Confidence Software Technologies (Ministry of Education), School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871, *China*)

Abstract: Given requirements on large-scale applications and offering enterprises complete sovereignty on their EPCIS, an innovative distributed architecture for Radio Frequency Identification(RFID) Discovery Service is provided. We remodel the EPCIS events that represent most events arising from supply chain activities. Based on "Follow the Chain" mode and using Object Naming Service(ONS), our approach is focused on improving the query routing and reducing hops. Experiments on our approach for RFID Discovery show its high efficiency and availability.

Key words: radio frequency identification (RFID); discovery; electronic product code information service (EPCIS)

1 引言

RFID(Radio Frequency Identification)技术是一种非接触、多目标、移动目标识别的自动识别技术,近年来成为自动识别领域的研究热点.该技术可以广泛应用于物流供应链、食品安全、防伪、身份识别、军事等方面.针对大型开环的 RFID 应用,特别是供应链环境下的应用,需要建立跨地区、跨行业的 RFID 公共服务基础设施和信息共享机制,作为核心公共服务之一的 RFID 发现服务,负责收集物品在生命周期内的过程信息,即将分布的物品信息按时间序列整合成完整的物品信息链.

EPCglobal 网络[1]是一种网络基础设施,负责收集、共享和访问每个以 EPC(Electronic Product Code)[2]标记的物品在供应链中移动的相关动态信息.EPCglobal 网络所提供的核心服务主要包括:(1)EPCIS (EPC Information Service):根据预定义的 ECSpec(Event Cycle specification)[3]识别或检验与 EPC 相关的业务事件的发生,然后存储这些业务事件作为 EPCIS 事件,并为本地和远程的上层应用提供查询接口.对 EPCIS 事件的访问权限是由每个 EPCIS 本地控制的,因而每个企业单独决定其 EPCIS 的访问控制策略.(2)ONS(Object Naming Service):负责将一个 EPC 解析为对应的 EPCIS 或者其他服务的地址,ONS 由 EPC 的拥有者(通常是物品的生产商)管理和维护.(3)EPCIS Discovery:ONS 只是用来获取 EPC 的拥有者(通常为生产商)所维护的 EPCIS 服务地址,但是在供应链中,其它企业的 EPCIS 也可能捕获了与该 EPC 相关的物品动态信息,而通过 ONS 不能获取这些 EPCIS 服务的地址.这种服务由 EPCIS Discovery 提供.在多个参与方组成的供应链中,某参与方通过 EPCIS Discovery 可以查询到某一物品的组成部分/成分,这些组成部分/成

收稿日期:2009-08-31;修回日期:2010-01-10
基金项目:国家高技术研究发展计划(863)(No.2006AA04A119,No.2006AA04A121);国家重点基础研究发展计划(973)(No.2009CB320706);国家自然科学基金(No.60803014)

分来自于何处,以及该物品如何交付给最终用户.

一般来说,供应链连接着很多企业,从原材料的生产商开始,到使用物品的最终客户结束[4].此外,一个企业也可能存在于多个供应链中.供应链中的企业通常会有如下三种基本查询请求[5]:(1)Pedigree 查询,追溯物品的完全历史信息,包括在何处加工或发生变化,以及原材料来自于何处等.(2)Recall 查询,查明一批物品的当前位置.(3)Bill-of-Material(BOM)查询,识别物品所有组成部分或原材料的生命周期信息,这种追溯式查询需要对物品进行递归的组装或分解过程.

目前对 RFID 发现服务的研究存在三个问题:(1)在大型开环的 RFID 应用(特别是供应链环境)中,物品数量及物品信息链的数据量十分庞大,因而服务器过载问题严重;(2)面对高强度的查询压力,发现服务器容易失效,所以在单点失效情况下应能继续提供有效的服务;(3)企业的 EPCIS 信息属于私有信息,需要有效地对允许共享的信息和绝对保密的信息进行控制.为了解决这些问题,同时保持较高的查询响应效率,本文从大规模实际应用的需求出发,在企业对其 EPCIS 进行完全的访问控制的基础上,选择了一种切实可行的模式,即"跟踪供应链"模式,以该模式为基础,提出一种新的具有分布式结构的 RFID 发现服务;重新为 EPCIS 事件进行建模,使之能够表达供应链活动中的绝大多数事件;结合分布式和并行处理技术来改进查询路由并缩减跳步数.

2 相关研究工作

目前,国内外对 RFID 发现服务解决方案的研究可以分为以下三种:

(1)集中式仓库型

这种模式中有一个中央的全局数据仓库,物品在供应链中移动时所产生的 EPCIS 事件的详细信息不仅存储在企业本地,还要上传到全局数据仓库.尽管这种模式容易实现,但对于海量数据的存储以及提供有效的查询响应是相当困难的;此外,这种模式缺乏对私密性的保护.

(2)集中式索引型

它是 EPCglobal 提出的一种改进的模式,这种模式有一个全局的中央 DS(Discovery Server)[6].当物品在供应链中移动时,供应链各环节产生的 EPCIS 事件的详细信息存储在企业本地,而本地企业仅将轻量级的事件索引推送给中央 DS.这种模式在一定程度上保护了企业的隐私,并显著降低了中央 DS 存储的信息量,也降低了数据库的查询代价.但这种模式提供的用户接口较为复杂.

(3)跟踪供应链型

这种模式采用分布式结构来代替上述两种模式中的中央服务器.IBM 和 Microsoft 公司目前正在从事该模式的研究,研究一种叫做"Theseos"的查询引擎[7].这种查询引擎绑定在每一个 EPCIS 上,接收来自本地用户的查询请求后,首先根据本地的数据和私密性策略对查询请求进行处理,然后根据本地查到的物品移动信息修改原查询请求和识别出相关参与方,并将修改后的查询请求转发给供应链上的相关参与方.上述查询过程叫做"Process and Forward",该过程沿着供应链中每个相关的 EPCIS 递归的重复执行.每个相关的参与方将本地的查询结果和从其它参与方返回的查询结果进行整合,并将整合的查询结果转发给向它发出查询的参与方,最终所有关于该物品的查询结果就会返回给发出初始查询的客户端.这种模式取消中央 DS,但是查询响应时间较第二种模式长,这是因为一个初始查询会衍生出多个新查询,而且这些查询的结果也需要经过多次转发才能返回到初始客户端.

通过对上述三种模式的分析,第二种模式是建立在企业愿意共享 EPCIS 数据(如:事件的索引)的假设之上,一旦这个假设在以后的实际应用中不成立,我们就会发现第三种模式更具可行性.因此,本文以第三种模式为研究重点,研究如何在这种模式的基础上改进查询路由策略,以及减少查询结果返回的转发次数.

3 一种分布式 RFID 发现服务系统

本节给出了 RFID 发现服务的一种分布式结构,该结构改进了"跟踪供应链"模式(或者说"Theseos"方法).第一,客户端的查询请求不仅提交给本地的 EPCIS,还可以提交给被查询物品的所有生产商的 EPCIS,而这些 EPCIS 可以根据被查询物品的 EPC 编码来查询 ONS 得到.这样处理以后,一个查询就可以分解为多个并行的查询流,从而可以提高查询处理的效率.第二,通过本地缓存机制来处理并行查询流之间的"查询碰撞"."查询碰撞"是指由于多条查询流并行,同一节点可能先后接收到其直接上下游节点发来的相同查询.为了避免重复查询,后来的查询应该被忽略.第三,供应链中的每个节点将其本地查询结果直接返回给发出初始查询的节点,从而避免查询结果沿着查询转发过来的路径的反方向经多次转发回到客户端.

3.1 基本假设

本文提出的方法基于三个假设:(1)在同一个供应链中,各参与方彼此共享它们的 EPCIS 数据.调查表明,为了降低成本、加强重点业务和关键环节、寻找增加收入的新机会,业务伙伴之间是乐于共享业务信息的[8].(2)"one step back-one step forward"原则.ISO/DIS 22005[9]提出了该原则,并详细说明了可追溯性的要求,即每一

个交易至少能够确定它的直接上游(供应方)和直接下游(接收方).(3)利用 ONS.由于 ONS 系统已经建立起来并投入了实际应用,我们可以利用 ONS 来获得物品的生产商的 EPCIS 地址.

3.2　分布式结构概述

RFID 发现服务系统由安装在各个 EPCIS 节点上的 DS 引擎构成,每个 DS 引擎仅为其所在的供应链伙伴节点提供访问内部 EPCIS 信息的接口,每个 DS 引擎只知道其直接上游和直接下游的节点的访问接口地址.系统的总体结构和查询处理流程如图 1 所示.

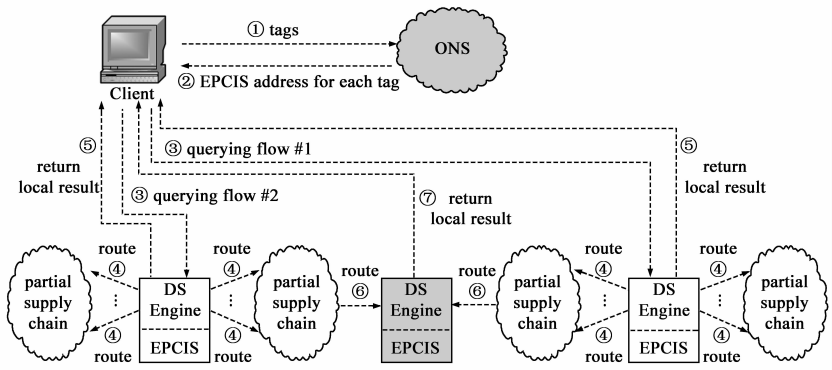


图1　RFID发现服务系统的分布式结构和查询处理流程

这种分布式结构下的查询处理流程可分为 4 个阶段,其中这种结构的特点体现在(1),(3)和(4).

(1)发起查询:首先,客户端生成一个具有唯一标识 queryID 的初始查询,查询参数是一组 EPC 编码.然后,客户端查询现有的 ONS 服务,将所有 EPC 编码一一交给 ONS 解析(步骤①).ONS 返回 EPC 编码对应的生产商节点的 EPCIS(或 DS 引擎)地址(步骤②).接着,客户端分别向每个已知的 EPCIS 并行发起对相应 EPC 编码的查询,同时客户端也将初始查询交给本地 DS 引擎处理(步骤③显示了 2 条并行查询流).

(2)处理并转发:每个节点的 DS 引擎接收到查询请求后首先查询其本地 EPCIS 的相关动态信息,然后根据本地查询结果生成与其直接上游和直接下游相关的若干新查询请求.然后,每个 DS 引擎并发地向其直接上游或直接下游发送对应的新查询请求(步骤④,⑥),并且每个 DS 引擎并发的将其本地查询结果直接返回给发起查询的客户端(步骤⑤,⑦).这个过程沿着供应链上相关的 EPCIS 递归地进行,直到每条查询流达到供应链边界节点或与另一条查询流"碰撞".图 1 用白色"云朵"代表这些过程.

(3)解决"查询碰撞":DS 引擎会把它已经处理过的查询忽略掉,例如,对图 1 中的灰色矩形来说,若来自左侧的查询早于来自右侧的查询,并且两个查询具有相同的 EPC 编码作为参数,则来自右侧的查询将被忽略.处理这种"查询碰撞"既能避免重复查询,又可避免在局部节点之间形成死锁.

(4)合并所有查询结果:客户端将在预定的等待时间内把从不同节点接收到的查询结果整合、排序,得到最终查询结果.因各节点是并发地返回结果,那么客户端需分辨哪些结果属于同一个初始查询,为此,需要保证所有转发的查询的 queryID 与初始查询保持一致.

3.3　EPCIS 的数据模型

为了增加供应链中物品的可跟踪/追溯性,同时更加全面地描述供应链活动中产生的各种事件,尤其是对那些涉及到将原料加工制成新产品的业务事件,本节对图 1 中每个 EPCIS 中的数据重新进行建模.基于 EPCglobal EPCIS Specification Ratified Standard[10]中定义的事件类型,我们给出了五种事件类型,其中包括一个基类事件和四个派生类事件,如图 2 所示.它们可以表达在各个行业的供应链活动中产生的事件.根据假设(2),我们在 EPCIS 的数据模型中加入了属性"*receiveFrom*"和"*sendTo*",分别表示某节点的直接上游(供应方)和直接下游(接收方).

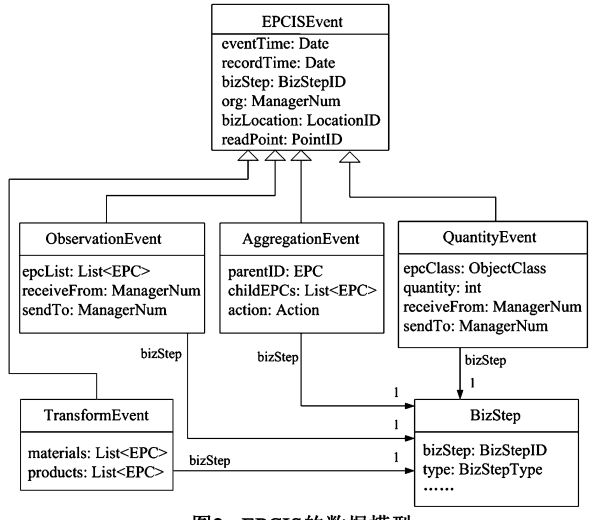


图2　EPCIS的数据模型

(1)EPCISEvent 是所有事件类型的基类,它描述了 EPCIS 所捕获事件的一般特征,包括事件的发生时间,发生地点以及基本的业务背景. *eventTime* 是事件的发生时间;*recordTime* 是事件的记录时间,是一个可选的属性;*bizStep* 是一个 *BusinessStepID*,它记录了事件所属的业务步骤,通过该属性可以实现业务步骤的跟踪;*org* 记录了事件在哪个企业发生;*bizLocation* 记录了事件发生时所处的位置,例如,"warehouse # 1";*readPoint* 记录

了事件在哪个物理位置发生,通常由物理读写器所标识,例如,“dock door #12”.

(2) ObservationEvent 描述了“物品仅被观察到但没被处理”这样一类事件. *epcList* 是一个 EPC 编码列表,它记录了事件中所读取到的所有物品标签编码; *receiveFrom* 记录了物品的直接上游(供应方),它以 *General Manager Number* 的形式唯一标识一个企业; *sendTo* 记录了物品的直接下游(接收方). 在 ObservationEvent 中, *receiveFrom* 和 *sendTo* 两个属性只能有一个有值.

(3) AggregationEvent 表达了物品的包装/组装与解包装/拆卸的发生,例如,“一些零件被装入箱子中”. *parentID* 标识了包装物品的容器,或是由多个部件组装成的新物品; *childEPCs* 是一个 EPC 编码列表,它记录了事件所涉及的所有被组装的部件; *action* 描述了事件中所发生的动作,例如,“ADD”表示包装/组装,而“DELETE”表示解包装/拆卸.

(4) QuantityEvent 描述的事件只关注同一种类物品的数量,而不关注具体的单品. *epcClass* 描述了物品的种类; *quantity* 记录了该类物品的数量.

(5) TransformEvent 刻画了多个物品经过加工处理以后生成新物品的事件,例如,“一块猪肉和一袋面粉加工成一些香肠”. *materials* 是一个 EPC 编码列表,记录了所有的原材料; *products* 也是一个 EPC 编码列表,记录了利用原材料加工成的所有物品.

上述四个派生类事件都通过一个“*bizStep*”属性关联了一个“业务处理步骤类”(BizStep),它刻画了事件发生时所处的业务背景.

3.4 查询处理过程和算法

下面针对 3.2 节的每个阶段给出具体算法,详细讨论查询处理过程. 首先定义算法中必要的数据结构.

```
Neighbor[//描述一个企业的邻居(直接上游或下游)节点信息
    neighborID: ManagerNum,//邻居节点的唯一标识
    epcISaddr: URI,//邻居节点的 EPCIS 地址
    routeType: RouteType,/* 从该企业到它的邻居节点的路由方向,有三种类型“Forward”,“Backward”,“Bidirectional” */
    lastUpdate: Date //该邻居信息的最近更新时间
]
Query[//描述发现服务的查询请求的数据结构
    queryID: int //查询的唯一标识
    epcList: List < EPC > //待查询的所有物品对应的 EPC 编码列表
    queryType: QueryType //查询类型,如“Pedigree”,“Recall”,“BOM”
    routeType: RouteType //与 Neighbor.routeType 含义相同
    routeTo: ManagerNum //该查询将被路由到哪个节点
]
Result[//封装了 queryID 和与该查询有关的一组 EPCISEvents
    queryID: int
    events: List < EPCISEvent >
]
```

(1)发起查询. 这是本文对“跟踪供应链”模式的一点重要改进,利用 ONS 的作用尽可能找到多个查询入口,这样在发起查询时就可以并发执行多个查询流. 下面用算法“startQuery”描述该过程.

```
startQuery(q: Query)
  starts: = {[q, local]} //q 是初始查询,local 是本地 EPCIS 地址
  for each epc in q.epcList
    isAddr: URI: = queryONS(epc) //获得 epc 对应的生产商的 EPCIS 地址
    if isAddr ≠ null
      //为每个 epc 构造一个新查询,其 queryID 与 q 相同
      qstart: Query: = [q.queryID, {epc}, q.queryTpye, bidirectional]
      starts: = starts ∪ {[qstart, isAddr]}
  /* 将具有相同 EPCIS 地址的查询合并,以减少查询数量,具体操作是将这些查询的 epcList 都添加到一个查询中,然后删除其它查询 */
  starts: = combineByIsAddr(starts)
  for each[q_start, isAddr] pair s in starts //并行执行
      //将 qstart 和初始客户端地址转发到对应生产商的 EPCIS 地址
      processAndRoute(s.qstart, s.isAddr, client)
```

(2)处理并转发

该过程用算法“processAndRoute”描述,包括 5 个步骤:第一,检查是否有“查询碰撞”发生;第二,在本地 EPCIS 中执行查询(详见 3.5 节 queryLocal);第三,根据本次查询和本地查询结果,重新生成若干与直接上、下游节点相关的查询(详见 3.5 节 rewriteQuery);第四,并行地将每个新查询转发给相应的上下游节点;最后,将本地查询结果直接返回给初始客户端. 其中,后四步仅在未发生“查询碰撞”时才执行.

```
processAndRoute(q: Query, isAddr: URI, client: URI)
  if checkIfQueriesCollision(q) = true //检查是否有“查询碰撞”发生
      stop
  result: Result: = queryLocal(q) //在本地 EPCIS 上执行查询 q
  //根据本地查询结果,将 q 改写为若干与直接上、下游节点相关的新查询
  qset_new: List < Query > : = rewriteQuery(q, result)
  for each q_new in qset_new//并行执行
      //从邻居列表中查找 q_new 要被转发到的远程邻居节点的 EPCIS 地址
      isAddr: URI: = lookupNeighbors(q_new.routeTo)
      //将 q_new 和初始客户端的地址转发到该邻居节点的 EPCIS
      processAndRoute(q_new, isAddr, client)
  returnToClient(result, client) //直接将本地查询结果返回初始客户端
```

(3)解决“查询碰撞”

采用本地结果缓存机制来解决“查询碰撞”问题,将偶对〈本地收到的查询,对应的本地查询结果〉利用某种机制(如 Least Recently Used (LRU), Time to Live (TTL))缓存起来.DS 引擎在处理一个收到的查询之前,先通过本地结果缓存检查该查询是否已被处理过,判断条件是:该查询与缓存中的某查询具有同一 *queryID* 并且前者的 *epcList* 是后者 *epcList* 的子集.该过程用算法“checkIfQueriesCollision”描述如下.

```
checkIfQueriesCollision(q:Query)
  /* resultCache 缓存了本地的 < query, result > 偶对,它实际上是以某 Hash 算法建立的映射表:其键为 query.epcList 的字符串形式再经过 Hash 算法得到,其映射值为偶对 < query, result > */
  //将 q 的 epcList 作为字符串散列,得到键 key
  key:String: = Hash(q.epcList.toString())
  value:Result: = resultCache[key] //通过 key 查到对应的映射值 value
  /* 检测 q 是否已被处理过,判断条件是:如果 q 与缓存中的某查询 value.query 具有同一 queryID 并且 q.epcList 是 value.query.epcList 的子集 */
  if value.query.queryID = q.queryID /* 确认 q 与 value.query 都是由同一初始查询衍生出来的 */
    and q.epcList⊆value.query.epcList /* 确认 q 查询的每个物品都已被查过.如果有某物品不在 value.query.epcList 中,则不能视为“查询碰撞”,需要重新查询 */
      return true
  else return false
```

(4)合并所有查询结果

既然每个 *DS* 引擎各自将其本地的查询结果直接返回给初始客户端,那么,经过一段预定义的等待时间后,客户端首先从返回的结果集合中选择与初始查询具有相同 *queryID* 的结果,得到一个结果子集,然后根据 *eventTime* 对该子集中所有 EPCISEvents 排序,形成最终应答结果.该过程用算法“combineResults”描述如下.

```
combineResults(queryID:int, results:List < Result > )
  lastTime:Date: = now //初始化上次处理 results 的时刻为当前时刻 now
  beginTime:Date: = now //初始化开始执行时刻为当前时刻 now
  answer:List < EPCISEvent > : = Ø //初始化最终结果
  while now-lastTime < period //period 为预定义的响应等待时长
    sleep(period) //等待一个 period 时长
    //从所有查询结果中识别与初始查询具有相同 queryID 的查询结果
    for each res in results
      if res.queryID = queryID
        answer: = answer∪res.events //将 res 中的事件集合并入 answer
        results: = results-res //从 results 删除 res 以减少 results 长度
    //将 answer 中各事件按照发生时间 eventTime 进行排序
    answer: = sortByEventTime(answer)
    lastTime: = now //更新 lastTime 为当前时刻
    display(answer) //显示最终查询结果
    if now-beginTime > maxDelay //maxDelay 为预定义的最大响应延迟
      break //maxDelay 一般相当于几倍的 period
  return (answer)
```

3.5 processAndRoute 相关细节

本节讨论算法“processAndRoute”中两个关键过程“queryLocal”和“rewriteQuery”.

“queryLocal”负责对本地 EPCIS 执行查询:首先检查本地结果缓存是否缓存了该查询的结果.如果没有缓存结果,则需要查询本地 EPCIS.因为 AggregationEvent 和 TransformEvent 记录了与查询的 EPC 编码具有“包装/组装”、“解包装/拆卸”或“加工变化”关系的其它 EPC 编码,所以先查询这两类事件中与查询的 EPC 编码相关的事件,并找到与这些事件关联的其他 EPC 编码.然后将新发现的 EPC 编码与原查询的 EPC 编码合并成一个集合,并在 ObservationEvent 和 QuantityEvent 这两类事件中查询与该合集中的任何一个 EPC 编码关联的所有事件.最后将本地查询结果添加到本地结果缓存,并返回给初始客户端.具体算法描述如下.

```
queryLocal(q:Query)
  /* 在一定时间内缓存本地的 < query, result > 偶对;检测 q 是否在缓存中(只要判断 q 与缓存中某查询具有相同的 epcList) */
  result:Result: = checkResultCache(q)
  if result = null then
    associatedEPCs:List < EPC > : = Ø
    for each epc in q.epcList
      //从 AggregationEvent 和 TransformEvent 表中查找与 epc 相关的事件
      e1:List < Event > : = findAggregationAndTransformEvent(epc)
      result.events: = result.events∪e1
      /* 从 e1 中寻找通过“包装/组装”、“解包装/拆卸”或“加工变化”关系关联的其他 EPC */
      associatedEPCs: = findAssociatedEPCs(e1)
    /* 对 q.epcList 和 associatedEPCs 并集中的每个 epc,从 AggregationEvent 和 TransformEvent 表中查找与该 epc 关联的所有事件 */
    relatedEPCs:List < EPC > : = q.epcList∪associatedEPCs
    for each epc in relatedEPCs
      e2:List < Event > : = findObservationEventAndQuantityEvent(epc)
      result.events: = result.events∪e2
    addToResultCache(q, result) //将查询结果添加到本地缓存中
  return result
```

根据本地的查询结果,“rewriteQuery”负责将查询重写成与其直接上、下游节点相关的新查询请求.因为 ObservationEvent 和 QuantityEvent 记录了本地节点与其直

接上、下游节点之间的物品收发(from-to)关系,所以需要对这两类事件进行分析:若某个事件具有 receiveFrom 属性,则可生成向直接上游节点转发的新查询;若某个事件具有 sendTo 属性,则可生成向直接下游节点转发的新查询.最后对所得的所有新查询,将具有相同 routeTo 的查询的 epcList 合并,这样多个查询合并为一个查询以减少新查询的数量.具体算法描述如下.

```
rewriteQuery(q, Query, result: Result)
  qset_new: List < Query > : = Ø //初始化一组新查询
  for each e: EPCISEvent in result
    /* 若 e 的类型是 ObservationEvent 或 QuantityEvent,则根据 e 的
       reciveFrom, sendTo 和 routeType 属性,构造一个新查询 q_new
       */
    if typeof(e) = ObservationEvent | QuantityEvent
      if e.reciveFrom ≠ null and q.routeType = Backward | Bidirectional
        //构造一个要被转发到直接上游节点的 q_new
        q_new: = createNewQueryBackward()
      if e.sendTo ≠ null and q.routeType = Forward | Bidirectional
        //构造一个要被转发到直接下游节点的 q_new
        q_new: = createNewQueryForward()
      qset_new: = qset_new ∪ q_new//将新构造的 q_new添加到 qset_new
  //将具有相同 routeTo 属性的新查询合并为一个查询,以减少新
    查询数量
  qset_new: = combineByRouteTo(qset_new)
  return qset_new
```

表 1 RFID 发现服务查询结果的一个实例

序号	事件类型	发生时间	Org.	收发关系	物品的 EPC 编码
1	Observation	2009/5/1	S_A	to M_D	$A_1 \sim A_{3000}$
2	Observation	2009/5/1	S_{B1}	to M_D	$B_1 \sim B_{3000}$
3	Observation	2009/5/1	S_{B2}	to M_D	$B_{3001} \sim B_{5000}$
4	Observation	2009/5/4	M_D	from S_A	$A_1 \sim A_{3000}$
5	Observation	2009/5/5	M_D	from S_{B1}	$B_1 \sim B_{3000}$
6	Observation	2009/5/6	M_D	from S_{B2}	$B_{3001} \sim B_{5000}$
7	Transform	2009/5/6	M_D	n/a	$m: A_1 \sim A_3, B_1 \sim B_4,$ B_{3001} $p: C_1$ C_2
8	Aggregation	2009/5/7	M_D	n/a	$c: C_1 \sim C_{10} p: D_1$
9	Observation	2009/5/9	M_D	to D_D	$D_1 \sim D_{100}$
10	Observation	2009/5/16	D_D	from M_D	$D_1 \sim D_{100}$
11	Observation	2009/5/17	D_D	to R_D	$D_1 \sim D_{50}$
12	Observation	2009/5/20	R_D	from D_D	$D_1 \sim D_{50}$

注:m - 原料,c - 零件,p - 产品

3.6 RFID 发现服务的查询结果

采用 3.2 ~ 3.5 节给出的 RFID 发现服务,能够查询到与给定物品(及其零件、原料)相关的、发生在供应链不同节点的所有 EPCISEvent 集合.下面通过一个典型实例来描述 RFID 发现服务查询结果的结构:一件产品 D 由 10 个零件 C 组装而成,每 2 个零件 C 由 3 件原料 A 和 5 件原料 B 加工制成.D 的制造商 M_D 从原料 A 的供应商 S_A 采购 3000 件原料 A,从原料 B 的两个供应商 S_{B1}, S_{B2}分别采购 3000 件和 2000 件原料 B,用原料 A, B 加工制成一批零件 C,然后把每 10 个零件 C 组装成一件产品 D,最后将 100 件 D 发给 D 的分销商 D_D, D_D 又将 50 件 D 发给一个零售商 R_D.表 1 列出了与产品 D_1 相关的查询结果.

4 算法性能分析

4.1 效率实验分析

为了检验本文提出的分布式发现服务的性能,我们实现了一个原型系统(DDS),还实现了"Theseos"与 DDS 进行对比.这两个系统都是以 webservice 方式实现的,采用 Java 1.5 和 JAX-WS 2.1 技术.节点间的所有通讯都以 webservice 调用方式进行.ONS 系统基于 Berkeley Internet Name Domain (BIND)实现.每个服务单独运行于一台计算机上,每台计算机配置了 Pentium(R) Dual-Core 2.5GHz 处理器、3GB RAM 和相同的 Windows、Microsoft SQL server.实验的测试数据是人工生成的,在特定的供应链网络布局下,给出一定数量的不同物品以及产生这些物品的起始节点集合,然后采用以下数据生成规则:一个物品产生后,有四种可能发生的动作(运送到邻居节点、同另一物品组装起来或从一个物品中分离出来、被加工制成新的物品、不再被移动),动作的选择是随机进行的,并且选择"运送到邻居节点"的概率随着该物品路径长度的增加而减小.为了检验上述两个系统在查询响应时间上的效果,我们生成的测试数据包含一个最大长度为 20 的供应链、0.5 ~ 1 万个物品和 2.5 ~ 5 万条 EPCIS 事件.

我们测试了 Recall 查询的总体运行时间与物品在供应链中移动的最大长度之间的关系.图 3 给出了对 100 个查询的平均运行时间,每个查询都是从供应链的一个前端开始寻找 10 个不同物品的所有 Observation 和 Quantity 事件.实验结果与预期的结果一致,由于查询转发次数和查询结果逆向转发次数的增加,Theseos 方法的响应时间随着路径长度的增加而增长,而 DDS 方法受此影响较小.在物品移动路径的长度 $\leqslant 6$ 时,采用 Theseos 方法的查询效率高于 DDS 方法,这是因为 DDS 在将初始查询分解为多个并行查询流之前,需要查询 ONS 服务,相比之下,Theseos 方法直接发出一个查询

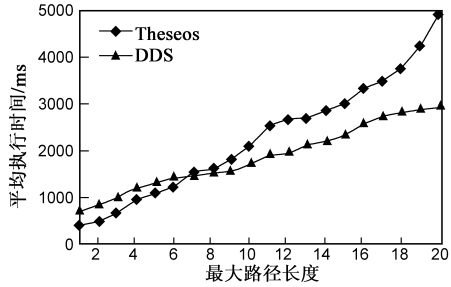


图3 Recall查询的平均执行时间

流,没有访问 ONS 的代价,所以在一定程度上(路径长度≤6)响应时间反而更短一些.但是随着路径的增长,DDS 的优势(多个并行查询流、直接返回查询结果)越来越明显,所以其响应时间的增长率低于 Theseos.

图 4 给出了相同实验配置下对 BOM 查询的测试结果.在这个实验中,物品之间具有 2~3 层的包含关系,并且每个查询都是寻找与物品相关的所有 Observation, Quantity, Aggregation 和 Transform 事件,因此响应时间总体上大于 Recall 查询.Theseos 和 DDS 在响应时间上的对比情况同第一个实验类似,造成的原因相同.

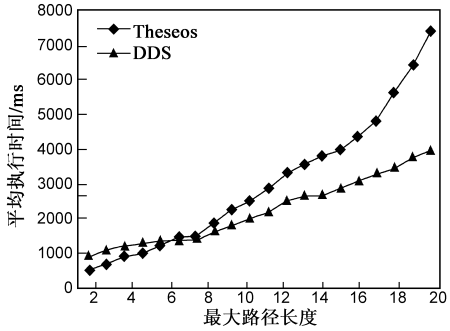


图4　BOM查询的平均执行时间

上述两组实验表明在物品移动路径较长(>6)时,DDS 比 Theseos 方法具有较短的响应时间.因而 DDS 更加适合较大规模的供应链应用,随着生产和物流规模的扩大,供应链将涉及越来越多的属于不同地区、国家甚至大洲的企业,最大路径长度将很少在 6 以内.

4.2　网络消息开销分析

DDS 和 Theseos 两种方法在网络消息开销方面,采用量化分析.设尚待发现的供应链实例 SC:〈n, e〉,其中 n 为供应链拓扑结构图的节点数目,且所有节点均正常工作,e 为节点之间的边数;i 为 DDS 中初始查询首次分解成的并行查询数目;X 为供应链中单次消息转发的网络开销,Y 为 DDS 中单次查询 ONS 的网络开销.因为 Theseos 中递归的查询转发过程和结果返回过程相当于 2 次遍历 SC 的所有边,故网络消息开销为 $C_T = 2Xe$.因为 DDS 中解决了"查询碰撞",避免了重复查询,所以多个并行查询流的转发过程当于遍历 SC 的所有边,而结果返回过程是各节点直接返回给发起初始查询的节点,再加上查询 ONS 和首次发出并行查询流的网络开销,总计

$$C_D = Xe + Xn + 2Yi + Xi = X(e + n + i) + 2Yi$$

又因为供应链 SC 的边数至少构成一棵树,至多任意两个节点均有连接,则 $n - 1 \leqslant e \leqslant n(n-1)/2$,故 $C_D - C_T = X(n - e + i) + 2Yi$,将 e 的边界值代入,得

$$X(n(3-n)/2 + i) + 2Yi \leqslant C_D - C_T \leqslant X(1 + i) + 2Yi$$

由上式分析可知:(1)当 e 较小时,有 $C_D > C_T$,即供应链 SC 的边数较少时,DDS 的网络消息开销比 Theseos 大,但由于 X, Y 是常数,i 通常较小,故开销相差不会很大;(2)随着 e 变大,必然使 $C_D < C_T$,且边数越多,DDS 的网络消息开销比 Theseos 小得越多.综上所述,DDS 的网络消息开销比 Theseos 在一定程度上具有一定优越性.

4.3　可用性分析

在可用性方面,若 DDS 和 Theseos 均不采用额外措施,则在某节点故障时,当前查询均失败,但由于 DDS 中多个并行查询流的存在,其它查询流还可以访问到更多的节点,从而获得比 Theseos 更丰富的查询结果,尚可满足一定的用户需求.当目标节点故障时,源节点可将该查询转发到其所知的某个(些)邻居节点,但不能保证查询能从这个(些)邻居节点延续下去.由于节点之间没有信息冗余备份机制(这是由 EPCIS 的私密性决定的),解决节点故障的策略尚需进一步研究.但可以说,在同等条件下,DDS 比 Theseos 更具有可用性.

5　结论

本文针对大规模的 RFID 应用,提出了一种新的具有分布式结构的 RFID 发现服务,用来在完全分布式的 EPCIS 节点中发现物品在供应链中发生的业务事件.首先,本文扩展的 EPCIS 事件模型适用于对供应链中物品的跟踪和追溯,并且新增加的事件类型"TransformEvent"进一步满足了实际应用需求,用来对那些涉及到将原料加工制成新产品的业务事件建模.本文提出的分布式 RFID 发现服务(DDS),综合了分布式技术和并行处理技术,给出了基于 ONS 的并行查询模式,改进了查询的转发机制并且减少了查询结果返回的路由跳数.通过与 Theseos 方法的对比实验表明,供应链的规模越大(如最大路径越长),DDS 方法比 Theseos 方法具有更高的查询效率.

DDS 系统目前已应用于酒类防伪,跟踪产品在生产商、批发商、零售商等供应链环节的移动细节,追溯产品来源以达到防伪目的;还应用于志愿者卡管理,将 RFID 编码嵌入志愿卡证,可实时监控志愿者所在岗位,以及追溯志愿者从事志愿活动的历史记录,方便对志愿活动的组织和管理.下一步,我们需要进一步研究 DDS 的服务质量(QoS)模型和控制方法,使发现服务的服务质量得以量化和评估.另外,RFID 发现服务的最终目的是让供应链上的各企业充分利用发现的数据集,从而改进自己的关键业务环节甚至供应链结构,因此将来还要研究针对发现服务查询结果的分析方法.

参考文献:

[1] EPCglobal. The EPCglobal architecture framework[S/OL]. http://www. epcglobalinc. org/standards/architecture/archi-

tecture_1_2-framework-20070910.pdf,2007-09.

[2] EPCglobal. EPCglobal Tag Data Standards Version 1.4[S/OL]. http://www.epcglobalinc.org/standards/tds/tds_1_4-standard-20080611.pdf,2008-06.

[3] EPCglobal. The application level events(ALE) specification version 1.1[S/OL]. http://www.epcglobalinc.org/standards/ale/ ale_1_1-standard-core-20080227.pdf,2008-02.

[4] K Vitasek. Logistics Terms and Glossary[S]. Supply Chain Visions, Bellevue, WA, Oct 2003.

[5] H H Bi, D K J Lin. RFID-enabled discovery of supply networks [J]. IEEE Transactions on Engineering Management, 2009, 56(1): 129 – 141.

[6] VeriSign. The EPC Network: Enhancing the Supply Chain[R]. VeriSign Inc, Mountain View, CA, 2004.

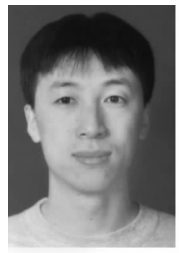
[7] R Agrawal, A Cheung, K Kailing, S Schönauer. Towards traceability across sovereign, distributed RFID databases[A]. Desai B C. Proc. of the 10th Int. Database Engineering & ApplicationsSymposium[C]. Delhi, India, 2006. 174 – 184.

[8] F Lai, J Hutchinson, G Zhang. Radio frequency identification (RFID) in China: Opportunities and challenges[J]. Int. J. Retail Distrib Manag, 2005, 33(11/12): 905 – 916.

[9] ISO. Traceability in the feed and food chain General principles and guidance for system design and development[S]. ISO 22005, 2007.

[10] EPCglobal. EPC information services(EPCIS) version 1.0.1 specification[S/OL]. http://www.epcglobalinc.org/standards/ epcis/epcis_1_0_1-standard-20070921.pdf,2007-09.

作者简介:



赵 文 男,1967 年出生,博士,副研究员,主要研究领域为软件工程、工作流技术和 FRID 相关技术.