支持移动客户应用的分布式订阅/通知框架*)

闫新庆　尹周平　熊有伦

(华中科技大学数字化制造装备与技术国家重点实验室　武汉 430074)

摘　要　订阅/通知框架是分布式系统中连接数据源和客户应用的一种常用框架。为了支持客户应用在不同的网络连接节点之间的迁移,实现客户应用移动的透明性,本文对通用的消息发布/订阅/通知框架进行了扩展,在客户应用移动时,自动引发通知订阅消息的重新发送、通过系统产生的 Fetch 消息的传递和 Replay 消息的传递,重构移动客户应用通知消息的传递路径,实现系统缓冲消息按照正确时间顺序重新发送给客户应用。同时该框架支持移动客户应用从多个数据源处订阅消息,具有良好的系统可适应性。

关键词　订阅/通知,移动客户端,消息路由,通知路径重构

Distributed Subscribe/Notification Framework with Support of Mobile Applications

YAN Xin-qing　YIN Zhou-ping　XIONG You-lun

(State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

Abstract　Subscribe/Notification is a common used software framework to connect data producers and consumers in distributed system. To support the mobility of client application among different nodes and implement the roaming transparency, the framework is extended and three additional procedures, redelivery of subscribe message, the deliveries of Fetch and Replay Message generated by the system, are introduced. Not only is the new message delivery path for the mobile client application reconstructed, but also the buffered messages for the client application can also be delivered to the mobile application in time order. Besides, the framework also supports multiple data sources for the mobile client applications, and is of excellent flexibility to the network system.

Keywords　Subscribe/notification, Mobile client, Message routing, Notification delivery path

1　引论

订阅/通知是得到广泛研究和应用的一种分布式消息传递框架[1,2],应用系统可以分为数据源和数据消费者。数据源产生数据,并向一个或多个数据消费者发布自己产生的数据。数据消费者则根据需要,订阅一个或多个数据源产生的数据[3]。在订阅数据源产生的数据时,数据消费者还可以指定自己需要数据的属性应该满足的条件,只有当数据源产生的数据满足条件时,数据消费者才会接受到通知消息[4]。某些应用系统可以既是数据源,也是数据消费者。传统分布式应用中,数据消费者和数据源紧密相连,数据消费者需要对数据源产生的所有新数据连续不断接收和处理。这样,不但使两者紧密耦合,增加应用系统开发和维护的难度,不利于系统的重构[5],而且可能会给网络系统带来大量的连接和数据传递负担,严重时导致网络的数据传送性能下降。通过订阅/通知框架,不但可以解除分布式系统中数据消费者和数据源之间在通讯和同步方式方面的紧密耦合,而且可以大大减轻网络数据传输的负担。

但是,目前的分布式订阅/通知框架对应用系统在网络的节点中迁移的问题考虑很少。由于分布式应用的复杂性,一些应用在运行时可能会从一个网络节点迁移到另外的网络节点中(如移动 Agent 等),需要研究一种支持移动客户的分布式订阅/通知框架。

一种方法就是使用移动 IP 的方法[6],来实现支持移动客户应用的分布式订阅/通知机制,但是不但移动 IP 的实现和使用非常不便,而且实现方案不具备通用性。Elvin 在服务器和移动客户之间引入一个集中式的缓存代理服务器[7],这种方式导致了性能瓶颈并且客户的移动范围受限于缓存代理服务。JEDI 使用 MoveIn 和 MoveOut 操作来支持移动客户[8],该算法的缺点是移动要由应用来控制,缺乏透明性,并且不能够保证消息的完整性。Fiege 等人则提出在新老代理之间自动建立路由,以便将消息转发给新代理的思想[9],这种方案相当复杂,对其性能和可实现性需要做进一步的评价。

我们提出了一种新的分布式订阅/通知框架,不但支持应用在网络节点之间迁移,而且保证原来发送到原节点中的通知消息能按照正确的时间顺序,重新发送给迁移后的应用系统,实现了应用在不同网络节点中的无缝和透明移动。

本文的主要结构如下:第 2 节讨论了分布式订阅/通知机制的体系结构,讨论了分布式订阅/通知框架的工作过程,以及网络节点中的消息路由机制。第 3 节提出了通过发送和处理 Fetch 消息和 Replay 消息,使缓冲的通知按照正确顺序重新发送给数据消费者,并且重构数据消费者的通知路由路经。第 4 节给出了算法的实现,并对算法进行了简要分析。最后总结了本文的主要工作。

*)国家 863 计划项目(项目批准号:2006AA04A110)、国家重点基础研究发展计划(973)(项目批准号:2003CB716207)、国家杰出青年基金项目(批准号:50625516)。**闫新庆**　博士后,主要研究领域为 RFID 应用、网格计算、中间件技术等;**尹周平**　博士,教授,博士生导师,主要研究领域为数字化制造、网络制造和电子制造等;**熊有伦**　教授,博士生导师,中国科学院院士,主要研究领域为数字化制造、网络制造和电子制造等。

2 分布式订阅/通知框架的体系结构

支持分布式订阅/通知的网络中的节点我们称为代理节点(broker),分为内部代理节点和边缘代理节点。边缘代理节点负责接收消息源产生的数据,将通知传给消费者,并和其它的代理节点连接。内部代理节点和其他的代理节点连接。连接的代理节点可以发送和接收消息。

2.1 注册和消息路由机制

在代理网络中,存在一个全局的注册机构,存放所有的数据消费者、数据源及其产生的数据概要。同时,该注册机构也保存了节点间的连接关系。每个边缘代理节点有一个局部的注册机构,保存了订阅该节点数据源的所有消费者和订阅条件。在数据源产生数据时,和注册机构中保存的订阅条件进行比较。如果满足该条件,则根据数据形成通知消息,然后把该通知消息路由给相应的消费者。

每个代理节点都有一个路由表,存放经过该节点的所有通知的路由信息。

2.2 消息订阅/通知过程

当数据消费者C通过边缘代理节点N提出自己的消息订阅条件F时,N首先在全局注册机构中注册C和F,然后在全局注册机构中查找符合F的数据源和其边缘代理节点D。根据保存的节点连接关系,计算从N到D的最短路径中的节点序列,形成一个订阅(Subscribe)消息。把节点序列附在Subscribe消息中,按照该路径从N到D路由该订阅消息。

如果存在多个数据源和代理节点能满足订阅条件F,形成多个Subscribe消息,并在代理网络中路由多个Subscribe消息。代理节点收到Subscribe消息时,根据Subscribe中包含的消息,在消息路由表中加入对应C和F的路由信息。某些代理节点可能收到多个对应于客户应用C和订阅条件F的Subscribe消息。此时,对每个数据源,在该节点的路由表中添加一个对应于客户C和订阅条件F的项。

在路由消息到达目标代理节点D时,D除了更新自己的路由表外,还根据客户C和订阅条件F更新自己局部注册机构中的信息。

当边缘代理节点P的数据源产生数据时,P对局部注册机构中的订阅条件进行扫描,找出能满足的订阅条件F和应用C。根据数据形成通知消息(Notification),查找自己的路由表中对应C和F的项,把Notify消息转发给下一个节点N。

代理节点接收到Notification消息时,在路由表找到相应的项,把Notify消息转发给该项中指定的下一个节点。这样,一直把Notification消息传递给边缘节点和数据消费者。

当数据消费者C取消订阅条件F时,边缘代理节点首先删除全局注册机构对应C和F的项,然后沿Subscribe消息的发送路线发送Unsubscribe消息。内部代理节点接收到Unsubscribe消息时,删除自己路由表中的相应项,然后转发给下一个节点。当边缘代理节点接收到Unsubscribe消息时,还需要删除自己的局部注册机构中的信息。

3 通知传递路径的重建

当客户应用在不同的边缘代理节点间移动时,我们需要扩展上面的发布/订阅框架,引入新的消息和机制。

当客户应用接收到来自新的边缘代理节点的心跳消息时,通过订阅/通知框架,客户应用自动通过边缘代理节点重新订阅数据,传递给边缘代理节点自己接收到的最后一个通知消息的编号。重新订阅消息的处理过程可以分为如下的三个步骤:

- 订阅消息的重新发送;
- Fetch消息的发送和处理;
- Replay消息的发送和处理。

通过这些步骤,完成缓冲通知的正确发送,重建通知传递路径。下面,我们讨论这三个步骤的执行过程。

3.1 订阅消息的重新发送

当接收到客户应用C的订阅条件F和应用最后接收到的通知编号num时,边缘代理节点首先更新全局注册表,查找从该节点到目标数据源所在节点的最短路径,形成Subscribe消息,附加通知编号num,路由此Subscribe消息。

接收到Subscribe消息的代理节点,首先判断自己的路由表中是否已经存在对应客户C和订阅条件的项。如果没有,则在路由表中加入对应客户应用C和订阅条件F的项。

如果路由表中已经有了对应客户C和订阅条件F的项,则该节点是原通知传递路径中的一个节点,称为交汇代理节点。订阅消息的重新发送在找到交汇代理节点后结束。交汇代理节点记录原来的路由项,然后更新该项。

此时,发送给应用C的通知就按照新的传递路线,传送到了C所连接的边缘代理节点。但是此时,这些通知还不能传递给应用C,而要在边缘代理节点处暂时缓冲起来。

3.2 Fetch消息的发送

找到第一个交汇代理节点后,在该节点处创建一个Fetch消息,沿着旧的通知传递路线,传递到原来的边缘代理节点。其目的有二个:第一是通知旧的边缘代理节点,把缓冲区的通知重新传递给应用;第二是查看原通知传递路线上的节点,决定哪部分的路线是可以重用的。

旧的消息传递路线上的节点在接收到Fetch消息时,查看自己的路由表中是否有多个对应客户C和订阅条件F的项。如果是,则表示客户C和订阅条件F有多个数据源形成的通知传递经过该节点。该节点也是交汇代理节点,则修改自己的路由表,把对应于应用C和订阅条件F的项更新为新的消息传递路线。然后,修改Fetch消息包含的源代理节点id,设置其为节点自身的id。在Fetch消息发送到原来的边缘代理节点后,这个阶段结束。

3.3 Replay消息的发送

边缘代理节点收到Fetch消息后,形成一个Replay消息,把缓冲的应用通知按照接收的时间顺序附着在此Replay消息中,把Fetch消息的源节点作为目标,沿着Fetch消息传送的反方向传递此Replay消息。然后,边缘代理节点清除保存的客户C和订阅条件F相关的内容。

其它的代理节点接收到Replay消息时,首先判断此消息的目标节点和节点自己的id是否相同。如果相同,则此节点是一个交汇代理节点,把Replay消息中的目标节点id设置为null,按照自己路由表中保存的内容,向客户C发送Replay通知消息。把id设置为null的目的是避免其它交汇代理节点对消息的进一步处理。

如果Replay消息的目标节点和节点的id不同,则此节点不是新的通知传递路径中的节点,此时该节点把Replay消息发送给下一个节点,清除自己路由表中关于客户C和订阅条件F的项。

最后,Replay消息到达给客户C目前所在的边缘代理节

点。该节点把 replay 消息的内容传向客户应用之后，把自己缓冲的消息也传送给客户应用。

这样，通过订阅消息的重新发送、Fetch 和 Replay 消息的发送，实现了客户应用在连接的边缘代理节点之间迁移时通知消息的完整和时序传递。该框架也适用于用客户应用订阅的通知来自于多个数据源的情况。

框架的工作过程如图 1 所示，左边是单个数据源的客户移动时通知路径的重构过程，右边是两个数据源的客户应用移动时通知路径的重构过程。

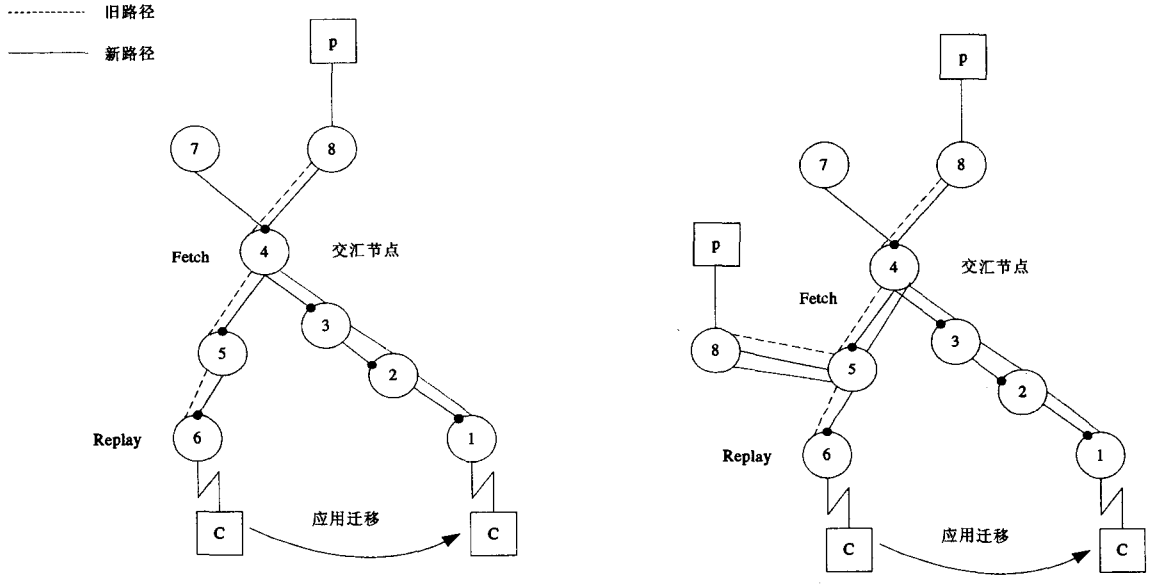

图 1　单数据源和多数据源的客户迁移过程

4　算法及分析

4.1　边缘代理节点中的算法

下面给出了连接移动客户的边缘代理节点中执行的算法。

算法 1　边缘代理节点中移动客户处理方法

```
void receiveSub(C, F, num, Lc){ //从 Lc 连接处接收到 C 的订阅消息,订阅条件为 F,原来接收的通知消息最后的序号为 num
    if (! RoutingTable. Contains(C,F) { //C 和 F 未包含在路由表中,新订阅 routeTable. add(F,C,Lc); //路由表中加入新项
      initializeCache(C,F); //初始化对客户 C 的缓冲
      if (num > 0) {
        //锁定,一直等到路径重分配结束
        RoutingTable. setBlockingFlags(C,F,Lc,true);
    }
    propagate(C,F,Lc); //把 C 和 F 递给下一个节点
  }
  sendCache(C,F); //给 C 发送缓冲的消息
}
//节点从 Bj 处接收到一个 fetch 消息时
void receiveFetch(C,F, num, Bprod, Bj) {
  if (advertisement. numberofMatches(F) >1) {
    routeTable. update(F,C, L,Bj) //把自己路由表中和 F,C 对应项改为 Bj
    } else {
      routeTable. delete(F,C,RouteTable. ALL); //删除路由表中和 F,C 对应的项
    }
    //根据缓冲消息 e1,...,en,形成一个 fetch 消息,并送回该消息
     send( new ReplayEvent(new FetchMessage(F, C, num, Bprod), [e1,…,en], Bj);
  }
  //接收到从 Bj 发来的 replay 消息
  void receiveReplay ( FetchMessage ( F, C, num, Bprod ), [e1,…,en], Bj){
      prependToCache([e1,…,en], C, F);
      routeTable. setBlockingFlag(F,C,Bj,false); //解除锁定
//自动接收以后的消息
}
```

4.2　内部代理节点中的处理算法

下面给出了具有移动客户的分布式订阅/通知框架中，为了处理移动客户在边缘代理节点之间的漫游，内部代理节点应该执行的算法。

算法 2　内部代理节点中的处理方法

```
//内部代理节点接收到一个从 Bj 来的客户为 C 的订阅消息
void receiveSub(C,F,num,Bj) {
    if (! RouteTable. contains(C,F)) {
      //新的通知传送节点,客户和订阅条件未知
      routeTable. add(F,C,Bj); //路由表中添加 C 和 F
      propogate(C,F,Bj); //散布该订阅
    } else { //该节点是交汇代理节点
      borkerOldNext = routeTable. get(C,F); //保存旧的路径
      routeTable. update(F,C,Bj); //更新路由表
      //沿旧的传送路线,形成和发送 fetch 消息
       send( new FetchMessage(C,F,num, Bcurrent, borkerNodeNext);
      }
    }
//接收到来自于 Bj 的 fetch 消息
void receiveFetch(C,F,num,Bprod) {
    borkerOldNext = routeTable. get(C,F) //保存旧路由信息
    routeTable. update(F,C,Bj); //更新路由信息
    if (routeTable. NumberofOccurance(C,F) >1){ //还有其它的数据源
      //修改 fetch 消息,并发送
       send(new FetchMessage(C, F, num, Bcurrent, brokerOldNext));
    }else {
      //保存最后一个交汇节点,继续发送 fetch
       Send (new FetchMessage (C, F, num, Bprod, brokerOldNext));
    }
}
//接收到来自于 Bj 的 replay 消息时
void receiveReplay(FetchMessage(C, F, num, Bprod), [e1,…,en], Bj) {
    brokerNext = routeTable. get(C,F); //取得下一传送节点
      if (Bprod ! = null){ //存在最后的交汇节点
        if (Bprod == Bcurrent){ //当前结点为最后的交汇节点
          //更改 Replay 消息中的 id,设置为 null
          send(new ReplayEvent(new FetchMessage(C, F, num, null),[e1,…,en], brokerNext);
        }else{
          routeTable. delete(C,F,Routetable. ALL); //清理路由表
          //向下一个节点转发 replay 消息
            send(new ReplayEvent(new FetchMessage(C,F,num, Bprod),[e1,…,en], brokerNext);
        }
      }
}
```

(下转第 89 页)

增长。当块大小增长到256kB时，数据的传输率接近140MB/S，而且还有提高的趋势。

从图4和图5中可以看出，以128kB的块大小来传输时磁盘的平均响应时间最短。而且比LINUX系统下标准的以4kB块大小为传输单位的平均响应时间提高了近一倍，同时本系统使用的RAID-H算法比RAID-0的相应时间要快。

结束语 在流媒体服务器中采用RAID-H算法提高了I/O处理的能力，而传输块单位大小的改变则由于充分地利用了磁盘的内部数据传输带宽，使得存储子系统的性能得到了较大的提高。

参考文献

[1] Keeton-k, Katz-r-h. evaluating video layout strategies for a high-performance storage server. multimedia systems, 1995, 3(2):43-52

[2] Lee K, Kwon J B, Yeom H Y. Exploiting caching for realtime multimedia systems//Proc. of sixth IEEE Intl. Conf. on Multimedia Computing and Systems. Florence, Italy, June 1999:85-93

[3] Yokota H, Idoue A, Hasegawa T, et al. Link Layer Assisted Mobile IP Fast Handoff Method over Wireless LAN Networks [A]//Mobile Computing and Networking Conf [C]. 2002:1-5

[4] Patterson D A, Gibson G A, Katz R H. A case for redundant arrays of inexpensive disks(RAID)[C]//Proceedings of the International Conference on Management of Data(SIGMOD). June 1988

[5] Chen P M, Lee E K, Gibson G A, et al. RAID: high-performance, reliable secondary storage [J]. ACM Computing Surveys, 1994, 26(2):145-188

[6] Vadala D. Manag in RAID on LINUX [M]. O'Reilly & Associates, 2003

[7] Perkins C. IP Mobility Support for IPv4 [S]. RFC3344, 2002

[8] Gibson G A. Network attached storage architecture [J]. Communications of the ACM, 2000, 43(11):37-45

[9] Anderson D C, et al. Interposed Request Routing for Scalable Network Storage. ACM Transactions on Computer Systems, 2002, 20(1):25-48

[10] Chen P M, Lee E K, Gibson G A, et al. RAID: high performance, reliable secondary storage [J]. ACM Computing Surveys, 1994, 26(2):145-188

[11] Hennessy J L, Patterson D A. Computer architecture: a quantitative app roach, Third edition [M]. Elsevier Science P te L td, 2003: 392-448

(上接第76页)

下面对算法执行的基本情况进行一些简单的分析。

4.3 算法的基本分析

在边缘代理节点处，导致客户请求建立连接和发布订阅条件的情况可以分为如下四类：

• 一个全新的客户应用，第一次尝试和边缘代理节点B连接并提交订阅信息。这种情况和B已经与客户连接并且已经接受订阅消息的情况稍有不同，因为在后一种情况下，代理节点B已经知道该客户的存在；

• 客户休眠一段时间后，继续工作。这种情况下，需要缓冲由于客户暂时休眠而错过的通知，但是客户连接的边缘代理节点并没有改变；

• 和边缘代理节点Bnew连接的客户C是一个可移动客户，从旧边缘代理节点Bold处漫游到了Bnew处，并和Bnew连接。这时，Bnew接收到客户C中自动发送的一个订阅消息，其中包括客户C从Bold处接收到的最后一个通知的序号；

• 客户在一个边缘代理节点处暂停工作，然后移动，并和另外一个边缘代理节点连接，发送订阅消息，继续工作。这种情况和第三种情况类似，在此不作详细的讨论。

结束语 本文讨论了支持移动客户应用的分布式订阅/通知框架的体系结构和通知传递路径的重构过程，通过订阅消息的重新发布、Fetch和Replay消息的传递，不但重构了新的消息传递路径，而且把原来发送给应用的通知消息按照正确的时序传递给了客户应用。

该消息发布/订阅框架同时也能适应客户应用从多个数据源处订阅数据，具有良好的可扩展性和适应性。

参考文献

[1] Eugster P T, Felber P A, et al. The many faces of publish/subscribe [J]. ACM Computing Surveys, 2003, 35: 114-131

[2] Costa P, Migliavacca M, et al. Algorithms for reliable content-based publish-subscribe: An evaluation [C]//Proceeding of the ICDCS 2004. Tokyo: IEEE Computer Society, 2004:552-561

[3] Pietzuch P R. Hermes: A scalable event-based middleware. Ph. D. Thesis. University of Cambridge, 2004

[4] 马建刚，黄涛，等. 面向大规模分布式计算发布订阅系统核心技术[J]. 软件学报，2006，17(1)：134-147

[5] 汪洋，魏峻，等. 可扩展和可配置事件通知服务体系结构[J]. 软件学报，2006，17(3)：638-648

[6] Ergen M, Puri A. MEWLANA-mobile IP enriched wireless local area network architecture [C]// Proceeding of Vehicular Technology Conference. IEEE Computer Society, 2002: 2449-2453

[7] Sutton P, Arkins R, et al. Supporting disconnectedness transparent information delivery for mobile and invisible computing [C]// First International Symposium on Cluster, Computing and the Grid. Brisbane, Australia, May 2001:277-287

[8] Cugola G, Dinitto E, et al. The JEDI event-based infrastructure and its application to the development of the OPSSWFM [J]. IEEE Transaction on Software Engineering, 2001, 27(9):827-850

[9] Fiege L, Gartner F, et al. Supporting mobility in content-based publish/subscribe middleware [C] // Proceeding of the 4th ACM/IFIP/USENIX International Conference on Middleware. Springer-Verlag, 2003:103-122