适用于云计算的面向查询数据库数据分布策略

文明波 丁治明

(中国科学院软件研究所 北京 100190)

摘 要 为满足海量数据的处理需求,业界提出了多种解决方案。云计算是目前较为热门的一种,它主要用廉价PC组成超大规模集群服务器来进行数据存储和处理。随着云计算技术的发展,越来越多的应用将转移到云中,数据库系统也不例外。但数据库系统要求的ACID特性在数据分布存储时可能导致部分操作性能低下,如连接查询操作。为在数据分布存储下提高数据库系统的性能,提出了一种面向查询的数据分布策略(Selection Oriented Distribution,SOD),即根据数据库的查询情况确定数据的分布算法。该算法适用于云计算,能明显提高系统的查询性能。

关键词 云计算,数据分布,面向查询,SOD

中图法分类号 TP391 **文献标识码** A

Selection Oriented Database Data Distribution Strategy for Cloud Computing

WEN Ming-bo DING Zhi-ming

(Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

Abstract Many methods have been proposed to satisfy the needs of massive data processing, among which cloud computing is an outstanding one. The main thought of Cloud computing is using large number of PCs to compose a huge cluster as a server. With the development of Cloud Computing technology, more and more applications will turn into Cloud, including the DBMS. As Database system requires ACID, when it comes to data distribution, some operations' performance may decline, such as joins. In this paper we proposed a Selection Oriented data Distribution strategy(SOD) to improve the performance of DBMS in Cloud Computing, which works well through the experiments.

Keywords Cloud computing, Data distribution, Selections oriented, SOD

1 引言

当今被称作信息时代,信息的重要性不言而喻。作为信息的载体——数据也一直是研究热点,主要集中在数据的存储、处理和信息提取等方面。

为高效存储和管理数据,20世纪60年代业界就提出了数据库管理系统(DBMS)的概念,并从文件系统发展出了早期的DBMS。自E. F. Codd 1970年发表关系模型[1]的论文以来,关系型数据库逐渐成为主流,并在相当一段时间内很大程度上满足了实际应用的需求。随着应用的扩展,为处理更复杂的数据模型、更复杂的查询操作以及更庞大的数据量,逐渐发展出分布式数据库、并行数据库、数据库集群、数据仓库等系统,它们都在某些方面满足了应用需求,但对海量数据的支持方面都还不够完善。

与此同时,自信息高速公路20世纪90年代提出以来,互联网飞速发展。目前Internet成为庞大的信息库,并且每天都有海量数据产生。要检索其中的数据,就需要海量存储空间和超大规模计算,这是单机型系统无法承受的,而大型服务器则存在性能有限和成本高昂等问题。为此Google采用大量廉价PC组成超大规模集群,并设计实现了GFS[2]分布式文件系统、BigTable[3]存储系统和MapReduce[4]并行编程环境。由此构成了Google的"云计算"环境,其他公司也相继提出和实现了类似的云系统,包括Amazon的EC2、S3和IBM的"蓝云"等。

由于现有单机型数据库系统面对海量数据时性能不足和数据库软件价格高昂,未来必有大量的数据库系统迁移到云中,但数据库系统所要求的ACID特性在数据分布存储时会导致部分操作性能严重降低。另外,目前对云计算中关系型数据库的研究还较少。因此本文主要研究关系型数据库数据的分布策略,以期通过更合理的数据分布策略来提高数据库系统在数据分布存储时的性能。

本文第2节主要介绍相关研究背景;第3节讨论算法思想、模型和实现;第4节为实验和分析部分;最后给出结论。

2 相关研究背景

2.1 云计算相关

云计算是目前热度很高的新名词,主要由Google,Amazon和IBM等公司提出。目前存在多种定义,各从不同角度对云计算进行了阐述。其中较为全面的定义[5]是:"云计算一词用来同时描述一个系统平台或者一种类型的应用程序。一个云计算的平台按需进行动态部署(provision)、配置(configuration)、重新配置(reconfigure)以及取消服务(deprovision)

到稿日期:2009-10-16 返修日期:2010-01-16 本文受国家自然科学基金项目(项目编号:60970030)资助。

文明波 硕士生,主要研究方向为移动对象数据库、分布/并行数据库;丁治明 研究员,主要研究方向为数据库、移动计算、信息检索。

等。在云计算平台中的服务器可以是物理的服务器或者虚拟的服务器。高级的计算云通常包含一些其他的计算资源，例如存储区域网络(SANs)、网络设备、防火墙以及其他安全设备等。云计算在描述应用方面，描述了一种可以通过互联网Internet进行访问的可扩展的应用程序。'云应用'使用大规模的数据中心以及功能强劲的服务器来运行网络应用程序与网络服务。任何一个用户都可以通过合适的互联网接入设备以及一个标准的浏览器来访问一个云计算应用程序。"

尽管对云计算存在多种理解，但其主要有如下3个基本特征：基础设施架构在大规模廉价服务器集群之上；应用程序与底层服务协作开发，最大限度地利用资源；通过多个廉价服务器之间的冗余和软件实现系统的高可用性。

由于云计算是基于互联网应用环境的，因此目前云计算中使用的数据库均为适合于互联网的键/值模型，暂时还没有完整支持关系模型的数据库系统，但也逐渐开始出现这方面的研究，比如微软计划将SQL Data Service更进一步关系化和Brian Aker发起的Drizzle数据库等。

2.2 数据分布策略相关

早期DBMS由于管理的数据集较小，无需对数据进行分布。随着数据集的扩大，单机存储容量有限和存储数据过多时，系统性能低下逐渐显现，因此需要将数据分布存储。数据分布主要应用在分布/并行数据库系统中，目前存在多种分布策略[7-12]，一般都针对表内元组进行分布，主要分为3大类：水平、垂直和混合分布。

所谓水平分布是按特定策略将关系的元组划分成若干不相交子集，每个子集为关系的一个逻辑片段，各片段分布到不同节点上；垂直分布则将关系的属性集划分为若干子集，然后将关系的键和属性子集的值分布到不同节点上；混合分布则是水平和垂直分布两种策略的混合。

由于关系型数据库在设计之初就要求小的冗余度，表的属性集较小，导致垂直分布代价较高，因此目前实际应用的多是水平分布，其主要有如下几种：

Round Robin：即轮转划分。将元组轮转循环分布到各节点上。优点：节点数据分布均匀，负载均衡，查询时只需广播即可实现；缺点：连接查询时节点间需交换大量数据，性能较低，系统伸缩性差。

Range：即范围划分。根据表中某一个或多个属性取值将所有元组划分成若干不相交子集后分布到各节点上。优点：便于精确查询，对连接查询支持较好；缺点：易造成数据倾斜，使节点负载不均，系统伸缩性差。

Hash：即散列划分。根据选定的Hash函数和属性将元组散布到相应节点上。优点：若Hash函数适合，可以实现数据的均匀分布，对非连接查询和等值连接查询支持较好；缺点：很难实现非等值连接查询，系统伸缩性差。

此外，近年来由于面向对象数据库的兴起，使元组中某些属性的数据量极为庞大，关于垂直分布的研究逐渐增多。另外，早在上世纪80年代就有人提出的基于列存储的数据库系统也在不断发展，目前已经有基于列(属性)存储的数据库系统被实现出来，但很少被应用。

3 面向查询的数据库数据分布策略(SOD)

3.1 云计算中数据库响应时间分析

云中数据库的查询操作流程为：

1)用户将SQL查询命令通过网络发送到云端数据库服务器(Server)；

2)Server对SQL命令进行预处理，然后根据数据的分布策略将查询子命令分发到各数据存储节点(Node)；

3)各Node执行接收到的SQL查询子命令，其间不同Node之间可能需要交换数据；

4)各Node将查询结果上传到汇总器(Collector)上进行数据汇总；

5)汇总器将结果返回给用户，完成查询。

以上各流程都会消耗一定的时间，具体如图1所示。

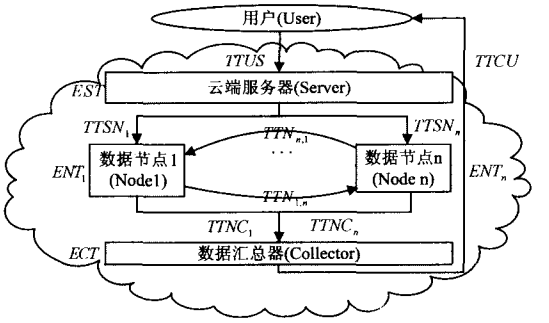

图1 云计算中时间耗费图

图1所示各时间量说明如下。

$TTUS$：用户传输SQL命令到服务器的时间；

EST：服务器解析SQL命令和等待时间；

$TTSN_i$：服务器发送子查询到 i 节点时间；

$TTN_{i,j}$：节点 i 向节点 j 发送数据时间；

ENT_i：节点 i 执行查询时间；

$TTNC_i$：节点 i 向汇总器上传数据时间；

ECT：汇总器汇总数据时间；

$TTCU$：汇总器向用户返回结果时间。

从用户提交查询命令到系统返回查询结果之间总响应时间 T 为

$$T = TTUS + EST + \max[TTSN_i + ENT_i + \max(TTN_{i,j}) + \max(TTN_{j,i}) + TTNC_i] + ECT + TTCU$$

式中，$TTUS$，$TTCU$ 是由查询时网络速度决定的，很难减小；EST 中SQL解析时间是常数量，等待时间则由服务器负载决定。节点 i 所用时间中 $TTSN_i$，$TTNC_i$ 基本不可控，但 ENT_i，$TTN_{i,j}$ 和 $TTN_{j,i}$ 主要由节点间交换的数据量决定，因此可通过减少甚至消除节点间数据交换量来减小 $TTN_{i,j}$，$TTN_{j,i}$，这样也使各节点间 $TTNC_i$ 较为均匀，从而减少最终响应时间 T。对汇总器而言，若要进行连接汇总，ECT 将是一大项，因此汇总器最好仅执行非连接汇总，将连接操作交由各节点执行。通过上述分析可知，总响应时间 T 中可调部分 TM 为

$$TM = \max[TTSN_i + ENT_i + \max(TTN_{i,j}) + \max(TTN_{j,i}) + TTNC_i]$$

3.2 SOD算法思想

从3.1节可知，减少节点间数据交换量能减小 TM，从而提高系统查询性能。但由于每个表的元组会分布到多个节点上，因此进行连接时必须将可能的连接元组进行集中。若其不在同一节点上，则必须进行数据传输。由此若能根据两表

的连接属性将两表联合分布，即将可能的连接元组分布在相同节点上，就能大幅减少甚至消除节点间的数据传输量。

例如假设现有数据库 DB_TEST，其中有以下 4 个表：

T1=(a,b,c,d,e)；

T2=(e,f,g,h)；

T3 =(i,j,k)；

T4 =(l,m)；

通过对数据库 DB_TEST 的查询操作进行统计，发现其中对 T3，T4 两表的查询较为频繁，但很少在一次查询中同时出现；而对 T1，T2 两表的查询则相对较少，但一般同时出现在一次查询中，即多为两表的连接查询。利用该统计结果可以设计出更合理的数据分布策略——按表间查询关联度来分布，即面向查询的分布策略(Selection Oriented data Distribution strategy，SOD)。

数据库中的表自使用 Create Table 命令创建开始，由于新元组的插入会逐渐生长，同时对该表的其他操作还包括查询、更新和删除等。一般情况下，对确定数据库中表的查询模式并不会随其大小而改变，所谓查询模式，是指不同查询的频率分布。例如，对 T1 表的查询有 95%是简单查询，5%是连接查询，即为 T1 的查询模式。

由此可在表创建后，对涉及该表的查询进行统计，确定其查询模式、计算查询频率和表间的关联程度。若两表查询频度都很高，但不同时被查询，则为防止单个节点负载过重，应将两表分布到不同的节点上；反之，若两表经常被连接查询，则为减少节点间数据传输量，应将两表按其连接属性分布到同一节点。此外，同一表内的不同属性被查询的频度也大多不同，查询时一般以某一个或几个属性满足特定关系为条件查询某个或多个其它属性，即属性间存在决定关系。利用该关系可更有效地建立索引和表的垂直分布等工作，为此在统计工作中也可计算属性间的决定关系。

3.3 SOD 算法模型及实现

对 3.2 节中的 DB_TEST，为使系统各节点负载均衡和减少系统响应时间，在数据分布时应考虑数据实际使用的查询模式，将 T1，T2 分离分布，而将 T3，T4 联合分布。为此需对查询进行统计，从中计算出可用于确定数据分布的各种度量值。

由于表在创建后的一段时间内主要进行元组插入操作，其它操作较少，同时对查询统计必然影响系统性能，因此需要在表增长到一定大小，例如 10000 个元组或 60M 数据量时再进行统计。同时要确定统计工作的终点，例如 30000 个元组或 200M 数据量时。具体的开始和结束统计的条件可作为参数确定。

统计开始之后，统计每条查询涉及的表和属性。例如对于一条 SQL 连接查询语句：

```
SELECT *
  FROM T1,T2
  WHERE T1.e = T2.e
```

有表 T1，T2 均被查询 1 次且为连接查询，涉及属性为 a,b,c,d,e,f,g，且由 e 属性确定其余属性，即 $e\rightarrow$other。对每条查询语句都有 $SelDes=(Tables, DeAttrs. TaAttrs)$ 查询描述数据，其中 *Tables* 是查询涉及的表集，*DeAttrs* 是查询条件属性集，*TaAttrs* 则是查询的目的属性集。经过对一定量 SQL 查询统计后得到查询描述集 *SelDesSet*。用 *SelDesSet* 计算数据库中各表的查询频率、表间查询相关度、表内查询决定属性等度量值，利用这些度量值通过相应算法可实现数据的分布优化。上述查询统计数据的结构为：表查询频率矩阵 $TaFreMatrix[N+1][N]$，属性决定关系 *AttrSeq* 为类似于 $a\rightarrow b,c$ 序列，各表的 *AttrSeq* 组成属性对集 *AttrSeqSet*。若两表间关联度超过阈值，则计算其连接属性 *JoinAttr*。

表查询频率矩阵 $TaFreMatrix[N+1][N]$ 形式如表 1 所列。其中 N 为数据库中表的个数，$TaFreMatrix[i][i]$ 中存储 T_i 被单独查询的次数，$TaFreMatrix[i][j]$ 中存储 T_i 和 T_j 被同时查询的次数，*total* 行中存储各表被查询的总次数。

表 1 表查询频率矩阵

tables	1	2	3	4
1	15	189	1	6
2	189	25	5	2
3	1	5	400	10
4	6	2	10	300
total	211	221	416	322

算法 1 统计表的查询频率和相关属性

输入：*SelDesSet*

输出：$TaFreMatrix[N+1][N]$，*AttrSeqSet*，*JoinAttr*

```
BEGIN
  //初始化 TaFreMatrix[N+1][N]各值为 0
  Initialize(TaFreMatrix[N+1][N])
  FOR each tuple in SelDesSet
    //处理只有一表的情况
      IF only i in Tables
        TaFreMatrix[i][i]=TaFreMatrix[i][i]+1
        add DeAttrs→TaAttrs to AttrSeqSet
    END IF
    //处理含连接的查询
    ELSE IF i,j in Tables
        TaFreMatrix[i][j]=TaFreMatrix[i][j]+1
        TaFreMatrix[i][j]=TaFreMatrix[i][j]+1
        JoAttr = getJoAttr(DeAttrs)
      add JoAttr to JoinAttr
      add DeAttrs→TaAttrs to AttrSeqSet
     END ELSE
    END FOR
    //计算各表总的查询次数
    addup all
END
```

为量化两表在一次查询中同时出现的概率，即两表连接查询的概率，定义表间关联度为

$$TaRelRate[i][j]=TaFreMatrix[i][j]/(\sum_{i=1}^{N}TaFreMatrix[i][j]+\sum_{j=1}^{N}TaFreMatrix[i][j]-TaFreMatrix[i][j])$$

式中各数据均取自 $TaFreMatrix[N+1][N]$ 中。采用以上公式的目的在于计算两表间连接查询频率与其非连接查询频率的比例关系，并以此为两表的表间关联度。

算法 2 计算各表间相关度

输入：$TaFreMatrix[N+1][N]$

输出：$TaRelRate[N+1][N]$

```
BEGIN
    initialize(TaRelRate[N][N])
```

```
    initialize(totalNum[N])
  FOR i=1 to N
      FOR j=1 to i
          TaRelRate[i][j]=TaFreMatrix[i][j]/(totalNum[i]+
                          totalNum[j]-TaRelRate[i][j])
      END FOR
    END FOR
END
```

由表 1 中的数据经算法 2 计算所得的关联度矩阵如表 2 所列。

表 2　表间关联度矩阵

tables	1	2	3	4
1	1	0.778	0.001	0.011
2	0.778	1	0.008	0.004
3	0.001	0.008	1	0.014
4	0.011	0.004	0.014	1
frequency	0.180	0.189	0.356	0.275

利用算法 2 输出的表间查询相关度矩阵和相关度阈值 *RaThreshold*，可生成数据库中表的分布方案。主要步骤为：首先将查询相关度高的两表进行联合分布，即将其数据量相加并作为一个整体。如果数据量较大，则使用其连接属性 *JoinAttr* 进行水平分布；其次将剩余表按其查询频率排序，将相关度低但查询频率高的表和查询频率低的表两两组合分别分布到不同节点上。

算法 3　生成分布方案

```
输入:TaRelRate[N+1][N],RaThreshold
输出:DisPlan
BEGIN
//首先处理连接表对
    FOR i=1 to N
      FOR j=1 to i
        IF TaRelRate[i][j]>=
          RaThreshold
          add(i,j) to DisPlan
        END IF
      END FOR
    END FOR
//处理剩余的表,将查询频率较高的表和较低的表配对分布
    sort(1,N) as(1,k)
    FOR i=1 to k/2
        add(i,k-i) to DisPlan
    END FOR
END
```

由表 2 得到的分布计划为(1,2),(3),(4)，即表 T1 和 T2 应该联合分布到相同节点上，T3 和 T4 要分离到不同节点上。之后就可以启动具体的数据移动进程，进行数据的分布迁移了。

3.4　数据分布后的查询算法

在 SOD 算法执行结束之后，数据库中的表都根据其查询情况分布到相应的节点上。当进行查询时，系统的执行分为 3 大部分：服务器端、数据节点端和汇总器端。其中服务器端主要负责：1)接收并解析用户的 SQL 命令；2)查询数据字典并把 SQL 子命令分发到各节点，同时把用户的信息通知给汇总器；3)等待各数据节点和汇总器的确认，否则重发 SQL 子命令。各数据节点负责：1)接收并确认收到 SQL 子命令；2)执行 SQL；3)把结果上传给汇总器并等待确认，否则重传。汇总器负责：1)接收用户信息并确认；2)汇总结果并回传给用户，等待确认，否则重传。具体算法如下。

算法 4　查询算法

```
SERVER 端
    输入:sql_cmd
    输出:Query_result
    BEGIN
      sql = get(sql_cmd)
      sub_sqls = pre_process(sql)
      //分发 SQL 子查询到各数据节点
      spread(sub_sqls,nodes)
      //通知汇总器用户信息
      noticefy(collector,usr_info)
      receive_reply(node)
      FOR node NOT received reply
        resend(sub_sqls,node)
      END FOR
    END
NODE 端
    输入:sub_sql
    输出:sub_qresult
    BEGIN
      sql = get(sub_sql)
      reply(server)
      sub_qreslut = excute(sql)
      RETURN sub_qresult to collector
      IF NOT confirmed(collector)
        resend(sub_qresult,collector)
      END
COLLECTOR 端
    输入:usr_info,sub_qresult
    输出:Query_result
    BEGIN
      get(usr_info)
      reply(server)
      receive(sub_qresult)
      confirm(node)
      Query_result = integrate(sub_qresult)
      return(Query_result,usr_info)
    END
```

4　实验与分析

由于搭建云计算环境较为困难，且由 3.1 节的分析可知，本文的主要目的在于减少 *TM*，因此实验这样进行：使用一台 PC 作为主服务器，对数据库系统进行数据分布控制。另外使用 3 台安装 PostGreSQL 的 PC 作为数据存储节点。以上 4 台机器的配置均为 2.2G CPU、2G 内存、Ubuntu 操作系统。

实验数据是前述 DB_TEST 中的模拟数据。开始时集中存储在主服务器上，统计开始条件为数据库单表数据量达到 50MB，结束条件为数据量达到 100MB。完成统计之后即按 SOD 算法进行数据分布。

数据分布完成后，在主服务器上进行查询和汇总，比较各种数据分布策略在不同数据量时数据库的查询响应时间。实验中参与比对的数据分布策略分别为集中存储(Together)、Round Robin、Range、Hash 和本文的面向查询的分布策略(SOD)。

实验结果如图 2、图 3 所示。图中横轴为单表的数据量，以 100M 为单位；纵轴为查询响应时间，以毫秒为单位。

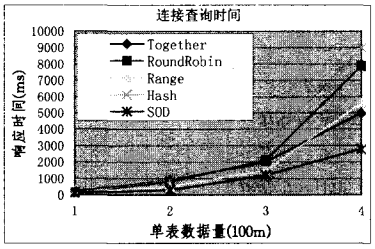

图 2　连接查询时间

从图 2 中可以看到，SOD 分布策略对于连接查询的反应明显优于其他分布策略，同时优于集中存储。原因在于需要进行连接的数据都分布在同一节点上，无需在系统内的节点之间传输数据，从而节省了大量的时间。

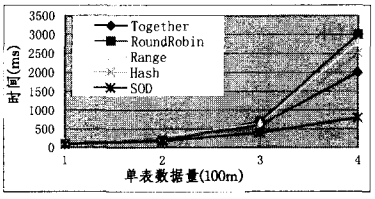

图 3　综合查询时间

从图 3 中可以看出，在进行综合查询时，各种分布策略在不同数据量下的反应时间差别较为明显，且在数据量较大时 SOD 策略明显优于其它分布策略。原因在于节省了大量的连接查询时间，从而提高了系统的查询性能。

结束语　根据云计算中数据分布存储的特点，本文提出了一种面向查询的数据库数据分布策略。根据数据库的实际查询模式进行数据的分布，将通常一起被查询的表分布到同一节点上，同时通过对各表的查询频率统计将经常被查询和很少被查询的表配对，以均衡各节点的负载，最终达到降低系统响应时间、提高系统性能的目的。通过实验证明，本文策略可以明显减少数据库查询响应时间，特别是连接查询的响应时间。

未来工作主要是实验确定对查询时间的统计，以研究对特定数据库的查询操作的响应时间情况，此种统计应该比仅对查询次数统计得到的结果更好；同时对于包含超大数据类型的表进行属性的查询分析，研究相关的垂直分布策略等。

参 考 文 献

[1] Codd E F. A relational model for large shared data banks[J]. Comm. ACM, 1970, 13(6): 377-387

[2] Ghemawat S, Gobioff H, Leung Shun-Tak. The Google File System[J]. SIGOPS Operating Systems Review, 2003, 37(5)

[3] Chang F, Dean J, Ghemawat S, et al. Bigtable: A Distributed Storage System for Structured Data[C]// 7th Symposium on Operating Systems Design and Implementation(OSDI 2006). Seattle, WA, USA, November 2006: 205-218

[4] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters[J]. Communications of the ACM, 2005, 51(1): 107-113

[5] 陈康，郑纬民. 云计算：系统实例与研究现状[J]. 软件学报，2009(5): 1337-1348

[6] Sylvain G, Le G. Using Cluster Computing to Support Automatic and Dynamic Database Clustering[C]// Third International Workshop on Automatic Performance Tuning(iWAPT). 2008: 394-401

[7] Guinepain S, Gruenwald L. Automatic Database Clustering Using Data Mining[C]// Database and Expert Systems Applications, 2006(DEXA '06). 17th International Conference. 2006: 124-128

[8] Zhong Ke, Dutt S. Effective partition-driven placement with simultaneous level processing and global net views[C]// IEEE/ACM International Conference on Computer Aided Design. Nov. 2000: 254-259

[9] Jean-Daniel C, Alain A, Alain A. Criteria to Compare Cloud Computing with Current Database Technology[C]// Dumke R, et al. , eds. IWSM / MetriKon/Mensura LNCS 5338. 2008: 114-126

[10] Fung Chi-Wai, Kamalakar K, Li Qing. Cost-driven vertical class partitioning for methods in object oriented databases[J]. The VLDB Journal, 2003, 12: 187-210

[11] Lee PeiZong. Efficient Algorithms for Data Distribution on Distributed Memory Parallel Computers[J]. IEEE Transactions on Parallel and Distributed Systems, 1997, 8(8): 825-839

[12] Li Wei, Chen C X. Efficient Data Modeling and Querying System for Multi-dimensional Spatial Data[C]// ACM GIS '08. Irvine, CA, USA, November 2008

(上接第 126 页)

实验结果表明，本协议的带宽利用率大于 75%，且随着接收端数目的增加，接收速率较稳定。NORM 协议的带宽利用率小于 50%，并随着接收端数目的增加呈下降趋势。

根据对实验结果的观察，本协议接收端接收数据的速率较稳定，基本消除了抖动。而使用 NORM 协议接收数据，接收端间或会出现 1～2 秒的停滞，这对于流媒体传输来说是不可接受的。

结束语　由于无线网络具有高误码率和高丢失率的特点，现有的可靠组播协议无法满足流媒体传输的实时性需求。

本文基于现有的可靠组播算法，设计了一种以端到端方式运作的轻量级的可靠组播协议。该协议使用 FEC 编码和 NACK 相结合的差错控制机制来实现可靠传输，并应用 TCP 友好的拥塞控制机制与其他传输协议公平共享带宽。该协议无需中间系统提供除基本的 IGMP、路由转发服务外的任何辅助功能，充分利用现有的网络资源，有很强的实用价值。

实验结果表明，在无线网络流媒体传输的应用环境中，相对于同类协议 NORM，本协议在吞吐量、带宽利用率及消除抖动方面上都有较大优势。

参 考 文 献

[1] Adamson B, Bormann C, Handley M, et al. NACK-Oriented Reliable Multicast Transport Protocol 〈draft-ietf-rmt-pi-norm-revised-13〉 Internet Draft, Intended status: Standards Track, Expires, December 2009

[2] Rizzo L. Effective erasure codes for reliable computer communication protocols[J]. ACM Sigcomm Computer Comm unication Review, 1997, 27(2): 24-36

[3] Sisalem D, Wolisz A. MLDA: a TCP_friendly congestion control framework for heterogeneous multicast environments[C]// Proc. of the 8th International Workshop on Quality of Service. Pittsburgh: [s. n.], 2000: 65-74