基于HDFS开源架构与多级索引表的海量数据检索mDHT算法

汤 羽[1] 王英杰[1] 范爱华[2] 姚远哲[1]
(电子科技大学 成都 611731)[1] (西安工程大学 西安 710048)[2]

摘 要 针对大规模能源数据系统的存储与快速检索需求,提出了一种基于HDFS/Hadoop开源平台的云存储架构及多级索引目录体系,以及此架构下的基于多级索引表的mDHT算法,并完成了算法的MapReduce编程实现。基于上述算法完成的4800万条数据的仿真实验表明:在数据量达到1200万~4800万条时,采用多级索引表的mDHT算法较常规的MS SQL Server实现和HDFS/Hive方法在检索性能方面有质的飞跃;与单级索引表检索方法比较,在数据查找时间上也有24.5%~57.8%的显著降低。文中提出的基于多级索引表的DHT算法为构建基于云存储架构的海量数据快速搜索引擎提供了一个关键技术。

关键词 大规模数据处理,云存储,多级索引表,查找算法,MapReduce
中图法分类号 TP393 **文献标识码** A

mDHT: A Search Algorithm to Extra-large Volume of Data Based on Open HDFS Platform and Multi-level Indexing

TANG Yu[1] WANG Ying-jie[1] FAN Ai-hua[2] YAO Yuan-zhe[1]
(University of Electronic Science and Technology of China, Chengdu 611731, China)[1]
(Xi'an Polytechnic University, Xi'an 710048, China)[2]

Abstract Corresponding to the storing and fast searching needs of extra-large scale of energy monitoring and statistics data, we proposed a Multi-indexed Distributed Hash Table (mDHT) algorithm based on the HDFS/Hadoop open platform and multi-level indexing design, and accomplished the MapReduce implementation of the algorithm. The simulation experiment at a scale up to 48 million data records indicates that, when the data volume reaches the scale of 12 millions to 48 millions, the proposed mDHT algorithm presents an outstanding performance in data adding operation, compared to that of traditional MS SQL Server implementation. Even compared to the single-index search application, the mDHT approach reduces the data searching time by 24.5%~57.8%. The multi-level indexed DHT algorithm presented in this paper provides a key technique for developing a fast search engine to the extra-large scale of data on the cloud storage architecture.

Keywords Extra-large scale data processing, Cloud storage, Multi-index, Search algorithm, MapReduce

1 前言

21世纪计算技术的发展及互联网的普遍应用,使得全球数据量呈现暴发性的增长。以生命科学领域的人类基因组学研究为例,2006年DNA碱基数目已超过1300亿,全世界每年生物数据量已超过10^{15}字节(1PB)[1]。国际数据公司(IDC)报告:2011年全球数据产出量更达到了1.8ZB(100万PB),未来10年内将以每两年翻一番的速度增长[2]。面对超大规模海量数据在存储管理、数据挖掘乃至从信息到价值的挑战,传统的基于结构型数据的关系型数据库难以应对超大规模数据、系统可扩展性以及快速检索等方面的更高要求。在区域医疗卫生信息系统、突发应急系统指挥决策、战场无线传感器监测系统、区域大规模能源监控网络等应用中,系统的响应性和可用性很大程度上取决于海量数据的实时检索能力。因此,超大规模海量数据的高效存储与快速搜索已成为工业界和学术界目前关注的一个前沿热点。

Google、Amazon等大型电子商务公司使用GFS (Google File System)[3]、HDFS (Hadoop Didtributed File System)[4],这类分布式文件系统管理海量数据的成功案例引发了研究界对这类非关系型的数据存储系统的兴趣。与传统关系型数据库(RDBS)的结构化存储相比,分布式存储系统采用实体数据(data block)与索引数据(metadata)分离的模式,更适合于非结构化数据的存储和基于目录索引表的快速检索算法。作为开源分布式计算框架Hadoop[5]的核心组成部分,HDFS提供了一种高吞吐量、高容错性、支持TB乃至PB量级超大规模数据的云存储体系。它采用一个NameNode和多个DataN-

到稿日期:2012-04-28 返修日期:2012-08-24 本文受教育部留学回国人员科研启动基金资助。
汤 羽(1964-),男,博士,教授,主要研究方向为分布式系统与云计算,E-mail:yutang@uestc.edu.cn;王英杰(1988-),男,硕士,主要研究方向为计算机应用技术与云计算;范爱华(1971-),女,硕士,副教授,主要研究方向为计算机应用;姚远哲(1973-),男,博士,讲师,主要研究方向为网格计算。

ode 组成的主从(master/slave)结构，实体文件被分成多个相同长度的数据块(data block)存储于不同的 DataNode 上，每个数据块生成的索引信息则存放在 NameNode 上的元数据索引表中。当应用程序需要访问一个实体文件时，HDFS 系统会首先查询 NameNode 上的主索引表，获得相关的索引信息后再对存放实体数据块的 DataNode 直接进行读取，获取所需文件。由此可见，基于 HDFS 架构的元数据索引表的构造以及基于索引表的搜索算法设计，是海量数据快速检索的关键[6]。

目前的海量数据搜索算法主要有 B/B+/B-树检索[7]、哈希检索[8]、分布式哈希表(Distribute Hash Table，简称 DHT)[9]。分布式哈希表(DHT)由 Litwin 在 1996 年首先提出[10]，这种结构将哈希表由单个节点扩展到分布式网络，网络中的每个节点负责一部分哈希表的存储和一定范围内的查询路由。伯克利加州大学和 AT&T 基于 DHT 结构提出了数据查找和节点路由的 CAN 算法[11]，即在已有的网络之上虚拟出一个叠加网络，再将全部节点映射到一个 n 维的笛卡尔空间，然后对每一个节点分配一块空间。而哈希表中的(key，value)经过映射函数的变换之后得到的就是笛卡尔空间里的点，并将其存储在该区域的节点上。CAN 算法具有较好的扩展性，节点状态信息与网络规模无关，但叠加网络增加了计算成本。Rowstron 和 Druschel 提出了基于后缀的 DHT 查找算法 Pastry[12]，即为每个节点分配一个 128 位的节点标识，并基于这一长度为($2^{128}-1$)的标识空间构建 Pastry 叠加网络和查询路由算法。基于前缀的静态 DHT 查找方法则是通过哈希前缀值来分组，把哈希前缀值相同的对象放在同一节点上，再建立路由表，在查找的时候通过路由表来进行快速的定位。基于前缀的动态 DHT 方法与其相类似，只不过动态 DHT 的路由表随着拓扑变化和存储对象的增加而不断更新。基于前缀或后缀的 DHT 算法具有构造简单、维护成本低的特点。但当存储节点数急剧增长，数据规模达到 TB 或 PB 量级时，这类算法使用的单级哈希表规模也急剧增长，从而使得查询效率降低。

本文在 Hadoop 开源架构上搭建了一个基于 HDFS 的区域能源数据分布式云存储体系，在此基础上提出了海量数据查询的多级索引表 DHT 算法，主要工作为：

1)设计了基于 Hadoop/HDFS 云存储架构的区域能源数据系统[13]，建立了水、电、气、煤、油等各类能源数据的特征模型，构建了基于 HDFS 的能源数据多级索引表；

2)基于上述大规模区域能源数据云存储架构，提出了分布式哈希表的多级索引查找算法(mDHT)，从而实现了在海量数据云存储体系中快速准确定位；

3)完成了上述 mDHT 算法的 MapReduce 实现，通过 4800 万条数据的仿真实验对算法性能进行了评估，并与传统的关系型数据库及 HDFS 单级索引表算法进行了对比，在数据读写时间和查找速度两个方面验证了新算法的优越性。

2 基于 HDFS 的云存储体系

在设计基于 HDFS 的云存储体系时[14]，选择区域能源消耗数据作为数据实体。一个省/市区域的能源消耗统计数据量是十分巨大的，水、电、气、煤、油、压缩空气等能源数据每天都在更新，大量的数据需要采集、存储和分析。仅以重庆市的电能数据为例，重庆市区每天的电能消耗(包括民用和工业用电)，仅仅电力数据每天就能达到几十 GB。如果要对重庆市月用电或季用电数据量进行分析，需要处理的数据量将达到 TB 量级。重庆市的区域能源数据监测网络由市、区(县)、企业/机构三级平台组成。企业/机构一级主要为数据采集终端，区域能源数据的存储与分析主要由市、区(县)两级平台承担。

基于 HDFS 云存储体系的区域能源数据集成平台总体架构如图 1 所示，市级能源数据中心支持市级功能平台，同时提供市级区域内的数据备份功能；区(县)数据中心对市级平台而言是实体数据存储节点，存有区(县)范围内的数据块(data blocks)，但对本区(县)而言它又是一个主节点(NameNode)，提供数据查询功能。基于 HDFS 架构的 DHT 查询算法将对市、区(县)两级系统的快速响应性、可用性起到关键作用。

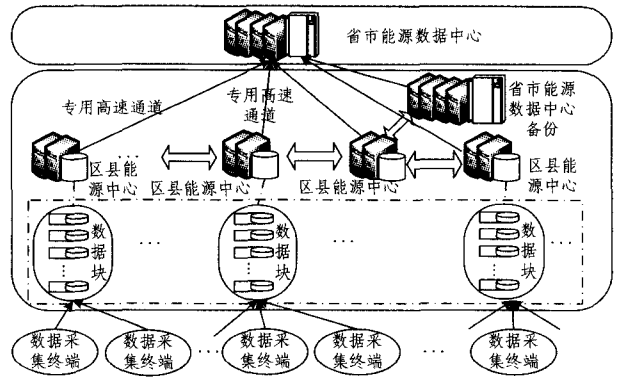

图 1 区域能源数据系统架构图

3 多级哈希算法(mDHT)

3.1 算法设计

基于分布式哈希表 DHT，我们设计了图 2 所示的多级哈希表搜索算法(multi-level indexed DHT，简称 mDHT)。在收到数据查找请求时，首先在第一级哈希索引表中进行检索，根据其(key，value)中 key 对应的 value 的值进入到第二级哈希索引表(二级哈希索引表存储了一级哈希表中分化出来的数据信息)，以此类推直到找到所需数据的地址信息，最后找到该数据为止。

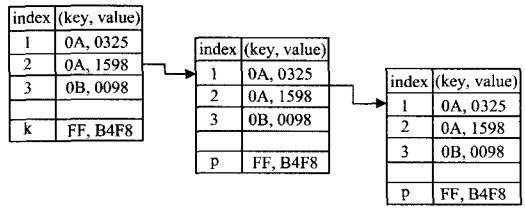

图 2 多级索引表 DHT 算法示意图

这种方式需要把所有的数据对应的信息构成索引存储到多级索引表中，算法的时间复杂度是 O(log ($m1+m2+\cdots$))，其中 $m1$，$m2$，…分别代表了第一级索引的信息条数、第二级索引的信息条数，以此类推。采用这种方式需要遍历各级索引表的 key 列。由于我们设计的多级索引表算法中各表的索引条目数是一致的，因此简化后的时间复杂度表示为：O(log $m*n$)，其中 m 是单个索引表中信息的条数，n 是索引的级数。

常规 DHT 算法中的时间复杂度为：O(log N)＋O(log M)。这里的 O(log N)表示从海量数据表中定位到所需数据的 key 的平均查找时间，O(log M)表示从一条含有各个数据量的记录中获取指定信息的平均查找时间。mDHT 算法的时间复杂度 O(log $m*n$)从表面上看并不优于常规 DHT 算法的 O(log N)＋O(log M)。但应注意到 mDHT 采用了 n 级索引结构，其单个索引表长度 M/n 远小于 M；另外 mDHT 的每级索引表只需要存储一个目录信息，数据量并不大，因而其查找时间也比较小。两种算法的时间复杂度 O(log $m*n$)与 O(log N)＋O(log M)的比较取决于相应的 m，n，M，N 等值，而数据量大小与多级索引设计方式决定了上述参数值。对常规 DHT 算法而言，其搜索成本尚需加入查找路由表的耗时，考虑这一部分后，其总耗时实际上远超出基于多级索引表的 mDHT 算法，后面的仿真实验结果也证实了这一结论。

仿真实验中我们仅采用了电力能源数据，其他类型的数据未纳入本次实验，但可以此类推。图 3 为能源信息表。假定每条电力能源数据包含如下信息：RowKey 为一个能源采集点的编号，例如某公司的电能账户编码。编码具有唯一性，我们采用这个编码作为能源数据表的 RowKey。如图 3 所示，能源信息表内除 Rowkey 外一共含有 4 个列簇，各个列簇用来表示能源数据的具体信息。TimeStamp 是时间戳，每一次的逻辑修改都会产生一个 TimeStamp 关联对应；Info 列簇用来存储能源账户的参数信息；Areas 表示能源的区域；Name 表示名称。以一个地址为重庆市九龙坡区，名为西南铝业集团的公司为例：Areas 对应的数据单元存储的信息为“重庆市九龙坡区”，Name 存储的信息为“西南铝业集团”。Type 列簇用来存储能源消耗的类型，包含能源类型，例如：工业用电，民用电，能源单位例如：千瓦时，度；Cost 列簇用来存储具体消耗数值。

RowKey	Time Satmp	Column Info	Column Type	Column Cost
NoCQ 00023	T5	Infor: Arease		
	T4	Info:Name		
	T3		Type: Energylevel Type	
	T2		Type: Measuring unit	
	T1			Cost:3210

图 3　能源信息表

HBase 中的表根据行键被分成了多个 Region。和 Google 的 BigTable[15] 十分类似。通常一个 Region 的一行都会包含较多的数据，如果以列值作为查询条件，就只能从第一行数据开始往下找，直到查询到相关的数据为止。这样显然是十分低效的。相反，如果将经常被查询的列作为行键、而将行键作为列重新构造一张表，即可实现根据列值快速地定位相关数据所在的行，这就是索引。由此可知我们的能源数据索引表仅仅需要包含一个列，所以索引表的大小和原表比起来要小得多。以下是哈希表的单级索引表和两级索引表的构造方法。

1)单级索引。能源消耗点名字为 RowKey，将原来的 ID 和其他信息作为列，则可以构建单级的索引表。查询时直接根据作为 RowKey 的能源消耗点名字来查询，以提高查询效率。

2)改进的两级索引。当采用多级哈希表构建索引时可以进一步提升查询性能。两级索引在以能源消耗点名字进行索引之前，先进行一个以地区(区县)为基准的索引，原有的消耗点名字索引变为 2 级索引，两级索引结构见图 4。

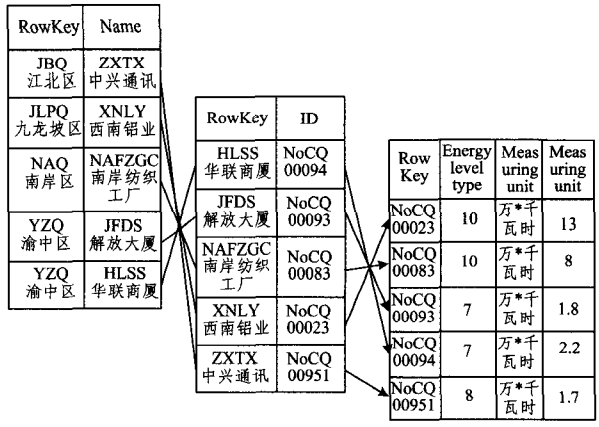

图 4　能源信息表的两级索引结构

3.2　多级索引表 DHT 算法的伪代码

我们提出的多级索引表 mDHT 算法是基于 Map/Reduce 编程模型来实现的。目前对于能源存储体系只做到两级索引，更多级的索引还有待进一步研究。

1)单级索引实现的伪代码：

```
Create Table Index_Name(RowKey,ID)
  Set RowKey=Column Info:Name
For Each Row In Energy_Info_Main_table
{
  // 如果不存在该名字,则直接加入索引表
  If( Row. Name not in Index_Name)
    Index_Name Add Row(Name,ID)
  Else if // 若名字可以重复,后缀序列号加入
    Index_Name Add Row(Name_1,ID)
  Else // 若名字不容许重复,返回错误
    return error
}
```

2)二级索引的实现伪代码：

```
Create Table Index_Area(RowKey,Name)
  Set RowKey=Column Info:Area
Create Table Index_Name(RowKey,ID)
  Set RowKey=Column Info:Name
For Each Row In Energy_Info_Main_table
{ // 第一级索引建立
  If( Row. Name Not in Index_Name)
    // 如果索引表不包含该名字,则直接加入
    Index_Name Add Row(Name,ID)
  Else
    // 名字不能重名,所以返回错误
    return error
  // 第二级索引建立
  If( Row. Area Not in Index_Area)
    // 如果索引表不包含该地区,则直接加入
    Index_Area Add Row(Area,Name)
  Else
    // 若 Area 存在于索引表,则后缀序列号加入
```

```
    Index_Area Add Row(Area_1,Name)
}
```

3)算法的 MapReduce 实现

HBase 的表通常较大,且随着数据量的增加会不断增大,如何高效地遍历索引表是一个问题,为此 HBase 集成了 MapReduce 计算模型用于大规模数据的并行处理[16,17]。我们的 MapReduce 实现主要设计了以下几个关键类(class):

1)InputFormat 类,把输入的数据分割成为 split,然后再进一步拆分为(key,value)键值对,作为 map 函数的输入;

2) Mapper 类,根据上一步输入的键值对产生中间结果;

3) Reducer 类,该类的作用是将中间结果进行合并(merge),生成最后的结果;

4)OutputFormat 类,负责最终结果的输出。

4 算法仿真实验

算法仿真实验所用数据是软件自动生成的模拟数据。生成的数据首先填充到 MS SQL SERVER 数据库中,然后再填充到对应的 Hadoop/Hbase 数据库中,数据格式如表 1 所列。实验环境为:3 台 PC 机器。Intel 2.2G 双核,2G DDR2 内存,Fedora1 5 操作系统;1 台 PC 为 NameNode,3 台为 DataNode(NameNode PC 上同时运行 DataNode)。

表 1 实验数据格式表

字段	类别
ID	INT
AREA	VARCHAR(50)
NAME	VARCHAR(50)
LEVEL	INT
UNIT	VARCHAR(10)
COST	INT
TIME	DATE

4.1 数据存储时间比较

首先进行了相同数据量下 MS SQL SERVER 数据库与 HDFS/Hadoop 体系存储时间的比较,所选取的测试数据量(条)为 5 万、15 万、380 万、800 万、1200 万、2400 万,分别对相应数据的写入时间进行了记录。表 2 列出的是 3 次实验数据的平均值结果。图 5 是基于表 2 的对比折线图。

表 2 数据写入时间比较(单位·秒)

数据量(万条)	WinXP MS SQL Server 单机版	Linux Hadoop 与 HBase 体系	节约时间(秒)
5	11	34	−23
15	34	43	10
380	981	496	485
800	2118	1081	1037
1200	3144	1977	1167
2400	2G 内存溢出-不支持	2630	——

对数据写入时间的分析可看出:在小数据量时(5 万至 15 万),Hadoop/HBase 体系相对于 MS SQL Server 并无优势,反而耗时更多,这是由于 Hadoop 分布式环境下路由分配和数据区块定位的附加成本所致。当插入数据量达到约 50 万时,两种数据库的插入时间差不多达到相同水平。而当数据量达到 380 万条时,Hadoop/HBase 开始展现其优势,插入时间比 MS SQL Server 有显著降低。SQL Server 的插入时间为 981s,而 HDFS/HBase 仅需 496s,后者大约只是前者的一半,在数据存储性能上有显著的提升。

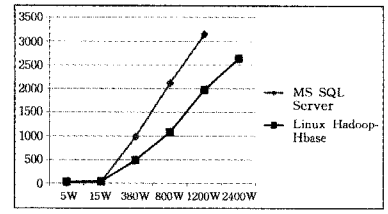

图 5 数据写入时间对比折线图

当数据量达到 2400 万条时,2GB 内存 PC 机 Windows 系统已经出现内存溢出,无法在一次操作中插入这么多数据。然而 Hadoop/Hbase 系统仍然表现优秀,能够非常轻松地胜任这项操作。实验中 Hadoop/HBase 将 3 台普通 PC 机的硬件资源(如 CPU 和硬盘,内存等)集成为一个高性能计算集群来完成大规模数据处理,较单机系统表现出了优异的性能。本次实验限于实验环境,测试的数据量有限,最高只达到 2400 万条数据。相信在更大规模数据情况下(数目达到上亿条数据,数据量达到 TB 或 PB 量级),基于 HDFS/Hadoop 的云计算系统将会展现出更大的优势。

4.2 数据查找时间比较

在对比了传统关系型数据库 MS SQL SERVER 和 HDFS/HBase 的数据写入时间之后,进一步对设计的多级索引表 mDHT 算法与 MS SQL SERVER、Hadoop/HBase/Hive、单级索引表 DHT 算法等方法的查询时间进行了对比。实验中用来进行查找时间对比的基础数据量分别为 20 万、400 万、1200 万、2400 万、4800 万。由于 HBase 不支持结构型数据的条件查询,而只能根据 RowKey 查询,导致了实际使用的不便。我们的解决办法是采用 Hadoop 环境下的 Hive 数据仓库,并建立把 HBase 内部数据表转换为类似于关系数据库的映射模型,在此基础之上支持传统的 SQL 语句查询。在 Hadoop/Hive 后端,则是把传统的 SQL 语句转换为 Map Reduce 的计算步骤,并对 HBase 进行数据转换操作来实现的。

表 3 数据查找时间比较(单位·秒)

数据量(万)	MS SQL Server 单机版	Hadoop 与 HBase(Hive 查找)	Hadoop 与 HBase(单级索引 DHT)	Hadoop 与 HBase(两级索引 mDHT)
20	0.3	1.12	0.87	0.93
400	11.0	27.3	1.20	1.01
1200	31.0	77.0	1.51	1.14
2400	66.7	154.3	2.27	1.73
4800	143.0	301.3	6.28	3.23

注:实验记录取查找 3 次的平均值。实验仅仅计算查找时间,并没有计入建立索引表的耗时。

我们选择了 Win XP 下的 MS SQL Server、Linux 下的 Hadoop/HBase/Hive 实现、单级索引表 DHT 算法 3 种方法与多级索引表 mDHT 算法(两级)进行对比。具体的仿真实验数据结果见表 3。同样,为了直观地展现数据结果,我们给出了对比折线图(见图 6)。

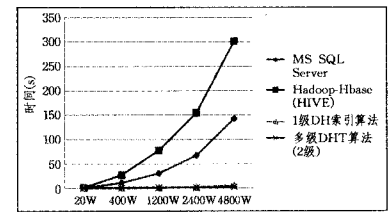

图 6 数据查找时间对比折线图

从图 6 可以看出，与传统的关系数据库 SQL Server 方法比较，采用非关系型数据库的 Hadoop/HBase/Hive 实现方式并无任何优势，表现反而较差。我们分析，这里的主要原因在于 Hive 这种查找方式。为了支持复杂条件查询，Hive 需要把原来的 HBase 数据表的数据结构模拟成另外的对应关系，然后用 Map Reduce 模型把条件查询语句再转换一次执行。这虽然实现了条件查询，但其转换过程耗费的时间过长。因此，对 HDFS/Hadoop 的(key，value)键值对存储结构在应用中进行转换时要慎重，因为转换的附加成本可能会完全抵消原有优势。

接下来对基于索引目录的 DHT 算法和前面两种实现方式的查询时间进行对比。表 3 的结果表明，基于索引目录的两种 DHT 算法(单级索引和多级索引)与 MS SQL Server 比较，其查询时间大幅降低(从 4800 万条数据的 143s 降至 3～6s)，效果非常显著。这是由于除了基于 HDFS/Hadoop 架构和索引目录的 DHT 算法在大规模数据存储检索方面具有性能优势外，还应该看到，我们仿真实验中的 MS SQL Server 算例是运行在单机上，而 Linux/HDFS/Hadoop 算例则是运行在 3 个节点的集群上，这也是后者性能远超前者的一个原因。但即使是将 SQL Server 运行在相同集群上，HDFS/Hadoop/DHT 算法仍然具有优势，因为 HDFS 作为分布式存储系统更适合于高性能集群这样的分布式环境。

表 3 还表明，与 Hadoop/HBase/Hive 实现方式比较，基于索引的 DHT 算法对查询时间的改善也是非常可观的。在 4800 万条数据负载时，查询时间从 HBase/Hive 的 301.3s 减少到 3.23～6.28s。这主要是由于 DHT 算法不需像 Hive 那样作两次转换，而是把列值 Name 作为 RowKey，直接由 Name 查询得到 ID，再回到主表查询出详细结果，这就是单级索引方式的优势。对于多级索引，为了更好地比较单级索引和多级索引的结果，我们把这两种基于索引表的 DHT 算法的结果独立出来，放在图 7 中进行比较。

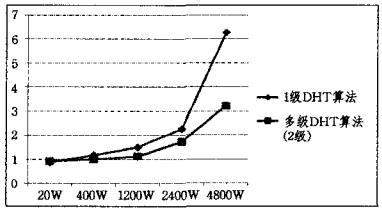

图 7　单级与多级 DHT 算法对比

从图 7 的两种 DHT 算法查找时间的结果可看出：在数据量不大时，采用两级索引 mDHT 算法比单级 DHT 算法耗费的查找时间反而更多。如图 6 中当数据量为 20 万条时，单级索引表算法和两级索引表算法的查询耗时分别是 0.87s 和 0.93s。其原因在于两级索引表算法需要进行两次索引搜索，而单级索引表算法只需一次索引搜索，在数据量不大时，两级索引表算法的索引表长度较短的优势并不足以覆盖多一次索引查询所耗费的成本。

但当储存数据量加大、索引表长度急剧增长、数据量达到 400 万条时，两级索引算法的优势开始显现，其查询时间已较单级索引算法降低 15.8%；当数据量进一步加大到 1200 万时，两级索引算法的优势进一步扩大到 24.5%；当数据量达到 3000 万～4800 万时，其降幅更达到 48.6%～57.8%，效果显著。两级索引算法较单级索引算法在数据查询时间具有优势的原因在于：虽然两级算法多了一次第 2 级索引表检索，但是第 2 级索引检索的结果大大减少了第 1 级索引表的定位时间，从而使得整体性能优化。

结束语　本文工作针对区域大规模能源数据三级平台监测系统设计了基于 HDFS/Hadoop 的云存储架构及能源数据多级索引体系，并基于上述架构提出了多级索引表 mDHT 算法，对云存储环境下的海量数据快速检索提供了一种新的方法。mDHT 算法把常需作为检索条件的列作为行键，而把原来的 RowKey 作为列，并通过构建多级索引目录最终使得第 1 级索引表的定位时间大大缩短，从而达到了海量数据检索性能大大提高的目的。本文还完成了 HDFS/Hadoop 架构下多级索引 mDHT 算法的 MapReduce 编程实现，并通过仿真实验对传统关系型数据库 MS SQL SERVER 方法、Hadoop/HBase/Hive 数据转换方法、单级索引表 DHT 算法以及多级索引表 mDHT 算法的数据写入时间和查询时间进行了对比。实验证实，在小数据量时(实验中 5 万～15 万条数据)，基于 HDFS 分布式文件系统的 Hadoop/HBase 方法较关系型数据库并无任何性能优势；但当数据量达到 1200 万条时，Hadoop/HBase 实现较 SQL Server 的数据写入时间大幅降低，降低幅度达 37.1%。在数据查询时间方面，基于索引目录的 DHT 算法与关系型数据库 SQL SERVER 和 Hadoop/HBase/Hive 数据转换法比较具有极大的优势。而单级索引 DHT 算法与多级索引 mDHT 算法之间的比较则证明：当数据量达到 2400 万～4800 万时，多级索引 mDHT 算法可降低数据查询时间 24.5%～57.8%，效果显著。本文工作证实了基于 HDFS/Hadoop 分布式文件系统存储架构和多级检索体系的 mDHT 算法在海量数据存储与快速搜索方面的有效性，这为进一步基于开源架构的海量数据快速搜索引擎的研究开发提供了基础。

本文工作仍有如下的局限和不足，需要在进一步的研究中完善：

实验数据量仍偏低。最大负载量为 4800 万条数据，数据短小，数据库总共才几个 GB 的容量。对于存储结构和算法在海量数据情况下(TB 量级)的性能检测需要进一步的大负载实验；

仿真实验所使用的能源数据模型只有单一的数值型和字符型数据。为了更好地评估存储体系和搜索算法的性能，应该针对包括数值、文字、医学影像各类异构数据的海量医疗卫生信息系统进行测试；

基于多级索引表 mDHT 算法的设计和实现目前只是一个初步的工作，下一步的研究将集中于异构海量数据存储体系的设计和多级索引算法的优化。

参 考 文 献

[1] 陈润生. 后基因年代的生命信息前沿[J]. 高科技与产业化，2006(8)：17

[2] 2011 年全球数据总量 1.8ZB[R]. IDC 研究报告. http://storage.chinabyte.com/163/12110163.shtml，2011-06-29

[3] Ghemawat S，Gobioff H，Leung S-T. The Google File System[C]//Proceedings of 19th ACM Symposium on Operating Systems Principles. Lake George，NY，October 2003：29-43

[4] HDFS-Apache™. Hadoop：http://hadoop.apache.org/hdfs/

(下转第 234 页)

据集，最小支持度由 0.1%变化到 1%来评估 A-NewMoment 与 NewMoment 算法搜索空间的大小。表 3、表 4 说明，不管在窗体的初始状态还是滑动状态，A-NewMoment 算法的搜索空间都比 NewMoment 算法小，这是因为 A-NewMoment 算法采取了一些搜索策略，极大地减少了搜索空间。

表 3　T20I5D100K 数据集下的比较

algorithm	state of window	support threshold			
		0.1%	0.2%	0.5%	1.0%
A-NewMoment	initialization	1843267	937712	471628	283368
	sliding	286678	218432	106608	51438
NewMoment	initialization	9427678	76432328	3712378	1640276
	sliding	1123642	836608	432218	217402

表 4　T30I20D100K 数据集下的比较

algorithm	state of window	support threshold			
		0.1%	0.2%	0.5%	1.0%
A-NewMoment	initialization	11946487	9872431	4564872	1738237
	sliding	12768	10223	5624	2964
NewMoment	initialization	73755746	73210436	38659311	18433637
	sliding	81134	76236	37542	17464

结束语　由于数据流的高速、无限和不可预测的特点，传统的高效的闭频繁项集挖掘算法无法运用到数据流环境中，而目前经典的数据流闭频繁项集挖掘算法存在因搜索空间大而造成算法时间效率低的问题，针对此问题，本文提出了一种数据流中闭频繁项集挖掘算法 A-NewMoment。其在窗体初始阶段，按照频繁 1-项集的支持度从低到高进行排序，发现由频繁 1-项集所生成的支持度最大的最长闭频繁项集，采用“不需扩展策略”与“向下扩展策略”可避免生成大量中间结果，快速发现其余闭频繁项集。在窗体滑动阶段，提出“动态不频繁剪枝策略”从存储闭项集的 HTC 表中快速删除非闭频繁项集及“动态不搜索策略”动态地维护所有发生变化的闭频繁项集。

参 考 文 献

[1] Golab L, Ozsu M T. Issues in Data Stream Management[J]. ACM SIGMOD RECORD, 2003, 32(2): 5-14

[2] 陈辉. 数据流频繁模式挖掘及数据预测算法研究[D]. 武汉：华中科技大学，2008

[3] Chi, Y, Wang H, Yu P. MOMENT: maintaining closed frequent itemsets over a data stream sliding window[C]// Proc of the 2004 IEEE International Conference on Data Mining, 2004. LosAlamitos, USA: IEEE Computer Society, 2004: 59-66

[4] Jiang Nan, Gruenwald. CFI-stream: Mining closed frequent itemsets in data streams[C]// Proc of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006. Philadelphia, USA: ACM publisher, 2006: 592-597

[5] Ranganath B N, Murty M N. Stream-Close: Fast Mining of Closed Frequent Itemsets in High Speed Data Streams[C]// Proc of the 2008 IEEE International Conference on Data Mining Workshops, 2008. Pisa, Italy: IEEE Computer Society, 2008: 516- 525

[6] Li Hua-fu, Ho Chi-chuan, Lee S-Y. Incremental updates of closed frequent itemsets over continuous data streams[J]. Exper Systems with Applications, 2009, 36(2): 2451-2458

[7] Yen S-J, Lee Y-S. An efficient algorithm for maintaining frequent closed itemsets over data stream[C]// Proc of the 22nd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, 2009. Tainan, Taiwan: Springer-Verlag, 2009: 767-776

[8] Cheng J, Ke Yi-ping, Nq W. Maintaining frequent closed itemsets over a sliding window [J]. Journal of Intelligent Information Systems, 2008, 31(1): 191-215

[9] 熬富江. 数据流频繁模式挖掘关键算法及其仿真应用研究[D]. 长沙：国防科技大学，2008

[10] 宋威，杨炳儒，徐章艳. 一种改进的频繁闭项集挖掘算法[J]. 计算机研究与发展，2007，45(2)：278-286

[11] Lucchesc C, Orlando S, Perego R. Fast and memory efficient mining of frequent closed itemsets [J]. IEEE Trans on Knowledge and Data Engineering, 2006, 18(1): 21-36

[12] http://www.almaden.ibm.com

(上接第 199 页)

[5] Apache™. Hadoop: http://hadoop.apache.org/

[6] 陈勇. 基于 Hadoop 平台的通信数据分布式查询算法的设计与实现[M]. 北京：北京交通大学，2009

[7] Berliner H. The B* tree search algorithm: A best-first proof procedure[J]. Artificial Intelligence, 1979, 12: 23

[8] Wang Wei-yan, Wang Xiao-ling, Zhou Ao-ying. Hash-Search: An Efficient SLCA-Based Keyword Search Algorithm on XML Documents[J]. Lecture Notes in Computer Science, 2009, 5463: 496

[9] Byers J, Considine J. Simple Load Balancing for Distributed Hash Tables[J]. Lecture Notes in Computer Science, 2003, 2735: 80

[10] Litwin W, Neimat M A, Schneider D A. LH*-A Scalable Distributed Data Structure[J]. ACM Trans. on Database Systems, 1996, 21(4): 480

[11] Grover L K. Quantum Computers Can Search Arbitrarily Large Databases by a Single Query[J]. Phys. Rev. Lett., 1997, 79: 4709-4712

[12] Rowstron A, Druschel P. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems[J]. Lecture Notes in Computer Science, 2001, 2218: 329

[13] 周可，王桦，李春花. 云存储技术及其应用[J]. 中兴通讯技术，2010，16(4)：24

[14] Sivathanu S, Liu Ling, Mei Yi-duo. Storage Management in Virtualized Cloud Environment[C]// IEEE 3rd International Conference on Cloud Computing. 2010: 204

[15] Chang F, Dean J, Ghemawat S, et al. Bigtable: A Distributed Storage System for Structured Data[J]. ACM Trans. Comp. Syst., 2008, 26(2): 4

[16] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[C]// Proc. 6th USENIX Symp. on Operating Syst. Design and Impl. 2004: 137-150

[17] Dean J. Experiences with MapReduce, an abstraction for large-scale computation[C]// Proc. 15th International Conference on Parallel Architectures and Compilation Techniques, 2006. Seattle, Washington, USA, ACM, 2006