基于 MapReduce 的数据立方体分区优化算法研究

张子浪　葛　昂　郑家民
(中电六所智能系统有限公司　北京　100083)

【摘　要】文章利用并行计算框架 MapReduce，探索数据立方体的计算问题。数据立方体的计算存在两个关键问题，一个是计算时间的问题，另一个是立方体的体积问题。随着维度的增加，计算时间将呈现指数级的增长，立方体的体积也是如此。尽管 MapReduce 是一个优秀的并行计算框架，但在处理数据倾斜时，分区算法不够完善，导致一些计算任务时间过长，影响整个作业的完成时间。本文通过数据采样的方式，优化数据分区，实验结果表明，数据立方体的计算的性能明显提升。为解决数据立方体体积过大的问题，在 Reduce 阶段将最终的结果输出到基于 NoSQL 的 HBase 数据库进行存储，HBase 方便水平扩展，同时也便于日后对数据立方体的查询。

【关键词】数据立方体;数据分区;数据分析
【文献标识码】A

The Research on Calculation of Data Cube in MapReduce

Zhang Zi-lang　Ge Ang　Zheng Jia-min
(The 6th Research Institute of CEC Intelligent System Co., Ltd.　Beijing　100083)

【Abstract】In this paper, parallel computing framework,MapReduce, is used to explore calculation of data cube. There are two key issues in calculation of data cube, the first problem is the volume of a cube and the other problem is the computation time. With the increase of the dimensions, the volume of the cube will increase exponentially, so is the calculation time. Since the data volume problem can be well solved using inexpensive disk system, it is very important to shorten the calculation time. Hadoop is an excellent framework for parallel computation, however, partitioning algorithm is not perfect when dealing with data skew, causing some computing tasks a long time and affect the completion time for the entire job. The experimental results show that the performance of cube was significantly improved through optimization of data partitioning based on the data sampling.

【Keywords】data cube;data partition;data analysis

1　引言

在互联网和电子商务领域，一些运营商以及电子商务平台提供商拥有大量的用户，并以云计算的方式向用户提供服务，这些服务响应用户的请求，在后端产生相应的数据，由于数据的集中储存以及用户的频繁请求，使得数据量呈现快速增长。在电子政务领域，一些政府部门根据自身信息化的发展水平及业务发展的需要，将信息化系统集中部署到省级机构，各地、市通过专网访问。这其中信息化化建设步伐更快的政府部门，在省级集中的基础上，实行全国数据的集中。在科学试验领域，科学家所观测的对象也产生了大量的数据，比如天文学当中利用天文望远镜，只需几天的时间，并能扫描半个天空。

就这些领域的数据产生速度而言，一些 IT 系统每天产生 TB 级的数据量，有的则多达 PB 级。有了大量的数据，就会产生数据挖掘的需求，包括对数据进行汇总分析。数据立方体能够很好地表达多维的结构化数据的

汇总分析,传统的联机分析(OLAP)技术对于立方体的计算方法也相对较为成熟。传统的 OLAP 根据数据储存方式的不同,可分为两类:一类是 ROLAP,以关系表进行多维数据的表示和存储;另一类是 MOLAP,以多维数组进行多维数据的表示和存储。ROLAP 计算数据立方体就是利用 SQL 中的 group by 语句对特定维度属性集合的所有子集分别集合,后来引入了 Cube 操作,一个 Cube 等价于多个 Group by。MOLAP 计算数据立方体时基于数组对数据进行集合,一种比较成熟的算法是多路数据聚集算法。

尽管基于传统的数据立方体的并行计算的算法比较成熟,但其不能直接应用于大数据的计算,因为由于这些大数据基于文件系统存储,而不是基于关系型数据库存储或者数组,而且,其数据量也大得多。

针对大数据的分析,Google 提出了 map-reduce 的并行计算框架,结合上千台的廉价 PC 服务器,使得大数据的分析能够在很短时间之内完成,Hadoop 是基于该编程模型的开源实现。利用 Hadoop 进行数据立方体进行计算的研究相对较少,有的利用 Hadoop 计算数据立方体,但没有考虑数据的优化分区,直接采用 Hadoop 缺省的分区方式,这种方式存在缺陷,不能让高度倾斜的数据(少数几个键值出现的次数占据了非常大的比例)均匀分配给各个并行的计算任务,导致某些计算任务的输入数据过多,从而导致其完成时间滞后于其它计算任务,影响整个作业的完成时间。

本文给出了基于 Map-Reduce 计算数据立方体的算法以及分区优化算法,为让数据均匀分布到各个 Reduce 任务,采用数据抽样的方式决定采用何种分区方式,为并行计算立方体提供了一种新方式。

数据立方体有多种,完整数据立方体,冰山立方体,封闭立方体。冰山立方体和封闭立方体考虑了数据立方体的体积,减少不必要的存储。由于封闭立方体或者冰山立方体中的某一个子立方体很可能就是一个完整的立方体,因此,计算完整立方体的过程不可避免,而且,完整立方体也是其它立方体的基础。所以本文研究完整立方体的计算。

在此先介绍并行计算、数据立方体、Hadoop 的相关概念,给出通用的计算数据立方体的 Hadoop 实现,在分析可能由于数据分布不平衡而导致的计算不平衡的基础上,设计基于抽样的分区算法。然后结合实验对算法进行分析。最后对当前工作进行总结,并提出未来的可能研究方向。

2 概念

2.1 数据立方体

实体关系模型主要应用于在关系型数据库中,这样的二维数据模型比较适合事务处理,但是不适合数据的在线分析。在数据仓库当中,往往需要从多个角度对数据进行分析,因此需要多维的数据模型,数据立方体就是用来描述多维数据模型的。

给定基本关系 R(A1,A2,A3,…,An,M),由 R 产生的数据立方体是 R 的属性的所有的可能组合,n 个属性产生 2^n 个组合。A1,A2,A3,…,An 为立方体的属性维,M 为度量维,M 是一个数字函数,描述数据以何种方式进行聚合或者计算。

取 n=3,即基本关系 R(A1,A2,A3,M)产生的立方体由以下分组构成:

{(A1,A2,A3),(A1,A2),(A1,A3),(A2,A3),(A1),(A2),(A3),()}。

度量维常见的聚合函数有 SUM,MAX,MIN,AVG 等。聚合函数可分为三类,分别是分布式,代数式,综合式的。考虑对分组 P 中的元素进行聚合。

分布式的聚合函数:,P_i(i=1,2,3,…n)为 P 的两两不相交的子集,即 $\cup P^n_{i=1}=P$ 并且□ ij,i≠ j,$P_i \cap P_j=\phi$,如果存在函数 G,使得 $F(P)=G(F(P_1),F(P_2),\cdots,F(P_n))$,那么称聚合函数 F 为分布式式函数。COUNT(), MIN(), MAX(),SUM() 都是分布式函数。除了 COUNT 函数外,其余三个几个函数 F=G。对于 COUNT 函数而言,G=SUM,即 COUNT (P)=SUM (COUNT (P_1), COUNT (P_2), COUNT (P_3), …, COUNT(P_n))。

代数式聚合函数:P_i(i=1,2,3,…n)为 P 的两两不相交的子集,$\cup P^n_{i=1}=P$ 并且□ ij,i≠ j,$P_i \cap P_j=\phi$,如果存在函数 G 和函数 H (对于所有的 P_i,H 函数返回一个 k 元组),使得 $F(P)=G(H(P_1),H(P_2),\cdots,H(P_n))$,那么称聚合函数 F 为分布式式函数。AVG 函数就是代数式函数,对每个每个 P_i,H 函数返回一个二元组 (sum_i,$count_i$),G 函数对所有的 sum_i 及 $count_i$ 分别相加,然后相除产生整体的平均值,即 $\frac{\sum_{i=1}^{n} sum_i}{\sum_{i=1}^{n} count_i}$。

整体式聚合函数:既不是分布式的函数以及代数式的函数称之为整体式聚合函数。

在对一个大的数据集进行聚合时,如果聚合函数是

分布式函数或者代数式函数,那么可以采用分而治之的思想,可以将大的数据集为众多小的数据集,然后对每个小的数据集进行计算,最后对中间的计算结果进行汇总,从而得到整体的计算结果。

2.2 MapReduce

MapReduce 是基于非共享的并行计算模型,该模型能够充分利用由多台机器组成的计算、存储、网络资源并行地处理计算任务,适合处理与大数据相关的统计分析。

MapReduce 并行计算模型与其它并行计算相比,主要有两个特点:一是其对串行任务与并行任务的隔离,以及计算任务能够在各个计算节点上独立地进行;二是编程模型简洁,学习成本低。MapReduce 将计算分为两个阶段:Map 阶段和 Reduce 阶段。首先,一个大的输入文件被分割成 M 块,分别由 m 个并行运行的 Map 任务进行处理,每个 map 任务以键值对 <k1,v1> 的形式接收输入,对于每一个记录,将其转为 <k2,v2〉的形式输出,然后系统调用分区函数对 k1 进行分区,map 任务完成后,形成中间的输出文件,每个具有相同 k2 的记录被分组并存放置在相同的分区内。在进行 Reduce 之前,系统将具有形同 k2 的记录进行合并,形成 <k2,list{$v2_1$,$v2_2$,$v2_3$,…}> 的形式,然后由 reduce 任务进行处理,输出键值对 <k2,v3>。

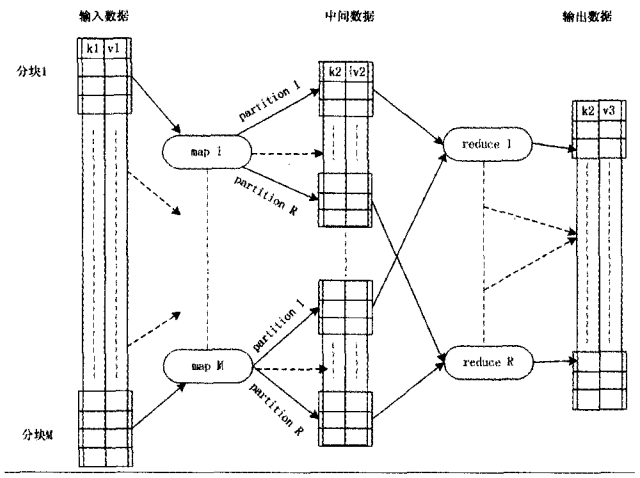

图 1 谷歌的 MapReduce 编程模型——再探

2.3 NoSQL

关系型数据库自 20 世纪 70 年代诞生以来,在企业和政府的信息化建设中得到了广泛的应用,今天关系型数据依然发挥着重要的作用。然而,对于以 PB 衡量的大数据,关系型数据库不能很好地应对。这些数据的特点的是数据类型多样,包括结构化、半结构化,非结构化的数据,另一个特点是数据量大。非关系型的 NoSQL 数据库适合这类数据的存储。NoSQL 具有三个特点:一是以 Key-Value 作为存储模型;二是保证数据的最终一致性;三是在保证应用不间断的情况下方便实现水平扩展。NoSQL 数据库主要包括 Cassandra、HBase、mongoDB 等。这三种作为 NoSQL 数据库中的主流代表,在很多生产系统中得到了应用,在处理大数据时,能保持很好的性能,都是较为成熟的产品。当然,这几种 NoSQL 数据库的系统架构不一样,侧重点也不一样。HBase 的文件系统基于 HDFS,能与 Hadoop 的 MapReduce 并行计算框架无缝集成。由于本文选用的是 Hadoop 的 MapReduce 并行计算框架,因此 NoSQL 数据库采用 HBase。

3 算法

算法除了实现 MapReduce 中的 map 接口和 reduce 接口之外,还实现了 getPartition 分区接口。Map 函数根据关系模式 R 的 n 个属性(A1,A1,A3,...,An),形成 2^n 个所有属性的可能组合,再取得相应属性的值作为键值,这样,在 map 阶段,每条输入记录将产生 2^n 个中间的键值对。为保证每个 Reduce 任务的负载大致相同,分区算法通过抽样的方式,统计每个键出现的频率,以每个键的频率之和度量分区的负载,尽可能让每个分区的负载大致相等。Combine 函数根据具有分布式或者代数式性质的函数 M 对数据进行聚合,把中间结果中具有相同 key 进行合并,形成一个键值对。Reduce 的实现相与 Combine 相似,只是在输出的时候有差异,Reduce 将最终结果保存至 HBase 数据库。

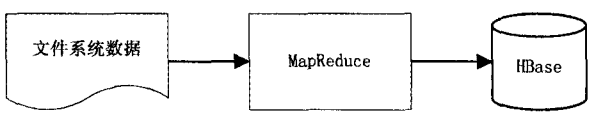

图 2 系统架构

3.1 Map/Reduce 实现

算法形式化描述:

//R 由 n 个属性组成的关系模式

R={A1,A2,A3…,An}

//e 为输入文件中一条记录

```
Map(e)
{
// 根据关系模式 R 和 e 产生中间的 key
// 每个输入元素 e 产生 2n-1 个 key
    emitKeyBySubSetOfR(0,n,R,e)
}
emitKeyBySubSetOfR(begin,end,R)
{
 // 计算包含 R[i]的子集
    for(i=begin;i<end;i++)
    {
  stack.push(R[i]);

   }
    //i 自增,计算不包含 R[i]的子集
    emitKeyBySubSetOfR(i+1,end);
    //I 为 R 的一个子集
    I={}
    for(each o in stack)
    {
   I=I∪o
    }
    // 元素 e 取 I 中的属性形成键值 k
    k=I(e)
    emit(k,e)
    stack.pop();
  }
  // 中间结果保存到文件
  Combine(k, iterator values))
  {
  total=0;
  while(e=(values.nextvalue())
  {
  total=M(total,M(e))
  }
  emit2file(k,total);
  }
计算最终结果并保存至 hbase 数据库
Reduce (k, iterator values))
{
    total=0;
    while(e=(values.nextvalue())
    {
     total=M(total,M(e))
    }
    emit2hbase(k,total);
  }
```

3.2 Partition 实现

每个 Map 任务会输出一系列的以键值对(<K,V>)形式的记录,然后由 Reduce 任务进行处理。由于存在多个 Reduce 任务,具体的一条记录由哪个 Reduce 任务处理,是由分区函数决定的。Hadoop 中缺省采用 hash 函数对 Map 任务输出记录的键值进行分区,由于每个分区只由一个 Reduce 任务处理,因此分区的数量等于 Reduce 任务的数量,每个 Reduce 任务处理一个分区。分区函数的形式描述为:

hash (Hash code (Intermediate-key) % numReduceTasks)

Hadoop 中的 Java 实现如下:

```
public class HashPartitioner<K, V> extends Partitioner<K, V> {
public int getPartition(K key, V value,
int numReduceTasks) {
return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
}
```

Hash 函数能够保证每个分区中键(Key)的数量大致相同。假设有 k 个不同的键,由 r 个 reduce 任务处理,分区函数能够保证每个 reduce 任务处理键的数量为 k/r。然而由于有的键(Key)频繁出现,即很多记录具有相同的键值,显然,分区中包含这样的键其数据量要大的多,所需的计算时间也更长。

建设有 6 个键,分别是 K1,K2,K3,K4,K5,K6,每个键包含的记录数分别为 1,2,3,4,5,6,这 6 个键由 3 个 Reduce 任务处理。采用 hash 分区策略,会形成 3 个分区,Partition1 包含 K1 和 K4,Partition2 包含 K2 和 K5,Partiton3 包含 K3 和 K6。尽管每个分区包含键的数量都为 2,但是每个分区的数据量不一致。Partition1 包含 5 条记录,Partition2 包含 7 条记录,Partiton3 包含 9 条记录。理想的情况应该是每个分区包含 7 条记录。

为了达到图 2 中均匀分区的效果,需要自定义分区函数。分区函数需要事先知道键的分布频率,如果数据

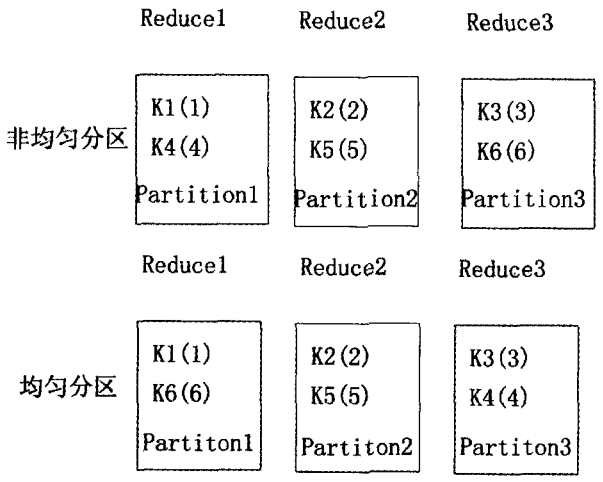

图 3 非均匀分区与均匀分区

集比较大，扫描整个数据集并求出各个键的分布频率，所需的时间比较长。可对数据集进行抽样，只针对一小部分数据进行统计，数据抽样的时间与整个计算任务的时间相比，可忽略不计。

假设一个数据集中包含 k 个不同 Key，键值分别为 K1，K2，K3，…，Kk，其出现的频率为 f (ki)，共有 r 个 Reduce 任务。每个 Reduce 任务的负载定义为其相应的分区的大小，分区大小可用该分区内记录数量大数目衡量，即分区内各个 Key 的频率之和。

$$L_{R_i} = \sum_{k_j \in P_j} f(k_j)$$

分区算法的目标是让各个 L_{Ri} 的值尽可能接近。

```
R=new ArrayList();// 初始时，每个 Reduce 的负载为 0
K = {K1, . . . , Kk};//K 为待分区的 key 的集合
While(K.length>0)// 如果还有 Key 没有分配到某个 Reduce 中
{  // 选取频率数最大的分配给某个 Ri
        kmax = argmaxk∈Kf(k)
   // 从待分区的 key 集合中移除
   K.remove(kmax)
// 如果存在某个 Ri 没有负载
if(R.length < r)
{    keylistofRi={};
     keylistofRi.add(kmax);
     // 直接将该 key 分配给 Ri
     R.add(keylistofRi);
}
// 如果所有的 Ri 都有负载，那么将该 key 分配给目前负载最小的 Ri
else
{
   keyListOfminLRi = argminiRi∈RLRi
    R.remove(keyListOfminLRi)
    keyListOfminLRi.add(kmax)
    R.add(keyListOfminLRi)
 }
}
return R;
```

这样，分区函数 P 能够将 K_i 分配给 R_j 处理，记为 P (K_i, R_j)。通过抽样形成的分区方案存入分布式缓存当中，每个 map 任务按照分布时缓存中的分区方案 P(K_i, R_j)对数据进行分区。

4 实验

为测试数据立方体计算的空间需求和性能，采用 8 个节点组成的 Hadoop 集群，每个节点的 CPU 为 4 核 3.1GZ，内存为 4GB，本地存储为 500G，操作系统环境为 Windows Server 2003。每个节点分别运行一个 Map 任务和 Reduce 任务，输入数据为根据 Zipf 分布人工合成，数据的倾斜程度通过参数 z 值控制，z 的取值范围是[0，1]，较大的 z 值表明更高的倾斜程度。

每个输入 Map 接收 500 万条记录，输入数据共 4000 万条记录。每个记录包含 4 个属性，其中一个属性的类型为数字，作为度量维度，另三个为字符类型，作为属性维。聚合函数采用具有分布式函数特性的 sum 函数。

为在数据抽样的比例和精确性之间进行平衡，通过多次试验，发现以 5%的比例进行抽样时，误差较小。按这个比例进行数据抽样时，数据抽样的完成时间为[5-8]s，这个时间与几百秒的计算任务而言，可忽略不计，以下有关完成时间的描述，均未将抽样时间计算在内。

当取 z=0，即数据均匀分布，分区算法采用 Hadoop 中的缺省分区函数时，最快的 Reduce 任务用时 130s，最慢的用时 131s；当采用自定义分区函数时，最快的用时 132s，最慢的用时 132.5s。

当取 z=0.6，即数据出现较高程度的倾斜，分区算法采用 Hadoop 中缺省分区函数时，最快的 Reduce 任务用时 90s，最慢的用时 331s；当采用自定义分区算法时，最

快 133s,最慢的用时 138s。

随着 z 取更高的值,两个分区算法性能差异明显,一度出现缺省分区函数比自定义分区函数慢 6 倍的情况。

显然,自定义分区算法在输入数据无重复的情况下,性能与默认的分区函数相当,然而当输入大量重复,发生倾斜时,自定义分区函数获得的性能提升非常明显。

当属性维度分别从 3 增加为 6 和 8 时,不管采用何种分区算法,计算时间呈现指数级增长的趋势,这主要和每个输入记录产生 2^n 个中间记录有关。

5 结束语

本文基于开源的 Hadoop 框架对完整数据立方体的计算进行了初步探索。Hadoop 并行计算框架非常优秀,简化了并行计算的编程模型,使得并行数据立方体的计算很容易实现。然而,数据立方体的计算性能非常重要,数据分区是影响性能的一个重要因素,因为并行计算的前提是各个计算任务的负载大致相同。Hadoop 的缺省分区机制在多数场合能够让每个 Reduce 任务的负载大致相同,然而在数据高度倾斜的情况容易导致计算偏斜。

本文从优化数据分区着手,采用抽样方式对分区算法进行了一定优化,当目标问题为分布式或者代数式的集合函数时,能够在一定程度上解决因为数据倾斜而导致的数据立方体的计算性能问题。

在系统架构方面,选用 HBase 存储数据立方体,以便水平扩展,应对数据立方体积快速增加的问题。实验结果表明,性能提升明显。此外,影响 Hadoop 的性能的因素有很多,比如集群的数量和集群中计算节点的数量,每个计算节点中运行的 map 任务和 reduce 任务的数量,以及网络带宽的情况,还有数据复制因子的影响,这些在未来的研究中也会涉及到。

参考文献

[1] Lammel, R.: Google's MapReduce Programming Model - Revisited[J]. Science of Computer Programming 70,2008, 1-30.

[2] Dean, J. and Ghemawat, S. Mapreduce: simplified data processing on large clusters [J]. COMMUNICATIONS OF THE ACM 51,2008.

[3] B. Gufler, N. Augsten, A. Reiser, and A. Kemper. Handling.

data skew in mapreduce. In The First International Conference on Cloud Computing and Services Science,2011,574-583.

[4] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M.Venkatrao,F. Pellow, and H. Pirahesh. Data Cube: A Relational Operator Generalizing Group-By, Cross-Tab and Sub-Totals[J].

Data Mining and Knowledge Discovery, 1996, 29-53.

[5] S. Ibrahim, H. Jin, L. Lu, S. Wu, B. He, and L. Qi. LEEN: Locality/Fairness-Aware Key Partitioning for MapReduce in the Cloud. In Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on,2010,17-24.

作者简介:

张子浪 (1978-),男,中国社会科学院研究生院,MBA,工程师;主要研究方向和关注领域:多维数据聚合。

葛昂,男,北京大学,MBA,高级工程师;主要研究方向和关注领域:数据挖掘、企业架构。

郑家民,男,北京航空航天大学,软件工程,高级工程师;主要研究方向和关注领域:数据挖掘、企业架构。