气象数据检索区域查询优化及并行算法设计

许 婧[1,2] 任开军[2] 李小勇[2]
(国防科技大学计算机学院 长沙 410073)[1] (国防科技大学海洋科学与工程研究院 长沙 410073)[2]

摘 要 随着数值天气预报水平和分辨率的不断提高,气象科学数据呈海量增长趋势,导致气象资料归档与检索系统(MARS)处理大数据服务请求的效率较低。针对此情况,开展了基于MARS检索区域查询方式的优化研究,结合数学补集思想与多路数组聚集计算原理,提出了一种高效的补集转换区域查询方法(CTRQ),从而实现大范围区域查询下的"大数据"计算转换为"小数据"计算。其基本思路是通过超立方体聚集维尺寸与区域查询服务请求的属性值集合大小比较,执行"过半求补"的索引计算操作,利用二次求补实现气象场数据物理存储信息的检索。实验表明,相比原始的索引计算方法,该方法能够有效降低数据检索时元数据索引计算的系统开销。在此基础上,结合并行处理方法,设计并实现了CTRQ并行算法,相比其改进后的串行算法最大获得1.9倍加速比,进一步提高了MARS的检索效率。

关键词 MARS,超立方体,区域查询,元数据索引计算,并行处理
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.03.011

Parallel Algorithm Design and Optimization of Range Query for Meteorological Data Retrieval

XU Jing[1,2] REN Kai-jun[2] LI Xiao-yong[2]
(College of Computer, National University of Defense Technology, Changsha 410073, China)[1]
(Academy of Ocean Science and Engineering, National University of Defense Technology, Changsha 410073, China)[2]

Abstract With continuous improvement of numerical weather prediction technology and resolution, meteorological data shows massive growth trend, resulting in less efficient meteorological archival and retrieval system (MARS) on large data service requests. Aiming at this issue, we carried out the research on optimization for region query based on retrieval in MARS, and proposed an efficient method through complement transform range query (CTRQ) by utilizing the complement ideas of mathematics and calculating principle of multi array aggregation, which transforms "big data" to "small data" in extensive range query. The basic idea is to calculate the rest by comparing the size of aggregation dimension in hypercube with attribute values set in query service request when indexes have more than half, and to second calculate the complement of physically stored information of meteorological data in retrieval. Experiment results show that comparing with the original index calculation method, CTRQ can effectively reduce metadata index computation overhead in data retrieval. On this basis, combining with parallel processing method, we designed and implemented CTRQ parallel algorithm, and attracted 1.9 times at maximum speedup ratio compared with the improved serial algorithm, to further improve the retrieval efficiency of Mars.

Keywords MARS, Hypercube, Range query, Metadata index computation, Parallel processing

1 引言

欧洲中期天气预报中心(European Centre for Medium-range Weather Forecasts, ECMWF)早期研制开发的气象资料归档与检索系统(Meteorological Archival and Retrieval System, MARS)[1]利用联机事务处理(OLAP)[2]中的多维模型[3]和数据立方体技术[4]实现了气象科学数据包括数值预报产品的归档和检索,主要是关于气象自描述文件多维多时空元数据的组织和管理。

世界气象组织(World Meteorological Organisation, WMO)于2004年建设THORPEX(The Observing System Research and Predictability Experiment)项目[5]时将MARS作为数据以支撑构建数据库系统和资料中心。同年,中国国家气象局使用MARS作为本地数据中心的基本存储架构实施国家级气象资料存储检索系统(National Meteorological Archival and Retrieval System, NMARS)项目[6]建设中的各种数据服务。

MARS主要通过对气象元数据核心架构数据立方体进行查询和维护,来完成气象科学数据的归档与检索。随着精细化数值预报的发展,气象科学数据规模快速膨胀,数据立方

到稿日期:2015-12-17 返修日期:2016-04-28 本文受国家公益性行业(气象)科研专项(GYHY201306003),国家自然科学基金资助项目(61572510),国家自然科学基金资助项目(61502511)资助。

许 婧(1991—),女,硕士生,主要研究方向为大数据处理;任开军(1975—),男,博士生,副研究员,主要研究方向为科学工作流、高性能计算;李小勇(1982—),博士,主要研究方向为并行计算、云计算、数据流管理。

体的规模急剧增大。对于大范围区域查询(Range Query,RQ)[7],即 MARS 执行海量数据检索操作时,需要访问数据立方体中大量的基本单元,从而需要对不同维集合进行多次聚集计算[8-9],造成元数据查询响应时间激增,进而导致 MARS 处理海量数据服务请求的效率较低。针对上述情况,本文借鉴数学补集思想,结合多路数组聚集相关原理[10],提出一种更加高效的补集转换区域查询方法(Complement Transformation Range Query,CTRQ)。该方法利用数学思想中的全集和补集的关系来处理各个维集合的索引计算,具体操作采取各个维集合全集"过半求补"的查询方式,并二次求补完成气象场数据物理存储信息的检索,实现降低数据立方体聚集计算的系统开销的目的。实验表明,利用本文提出的 CTRQ 方法执行大范围区域查询时,可以有效降低分组和聚集运算的计算代价。在数据集一定的情况下,区域查询范围与系统检索的响应时间成反相关,即区域查询范围越大,系统检索响应时间越短。另结合并行处理技术[11]实现 CTRQ 方法的并行化操作,进一步降低元数据查询的响应时间,完成海量气象科学数据大范围区域的查询优化。

本文第 2 节简单阐述了 MARS 系统结构以及数据立方体技术;第 3 节主要研究大范围区域查询优化算法和设计其并行算法;第 4 节通过实验验证该优化算法以及并行化操作的高效性;最后总结全文以及给出未来的研究方向。

2　MARS 与立方体技术的应用

2.1　MARS 简介

MARS 由客户端与服务端组成,是典型的 C/S 架构。其中,客户端与应用进程或者 web 页面进行交互,负责接收数据服务请求,并对请求参数进行验证处理后转发给 MARS 服务端。为了方便数据类型以及文件存储系统的扩展,MARS 服务端解除了数据在逻辑视图与物理视图的耦合,并分别交付于 MARS Server 和 Data Server 进行处理,如图 1 所示。

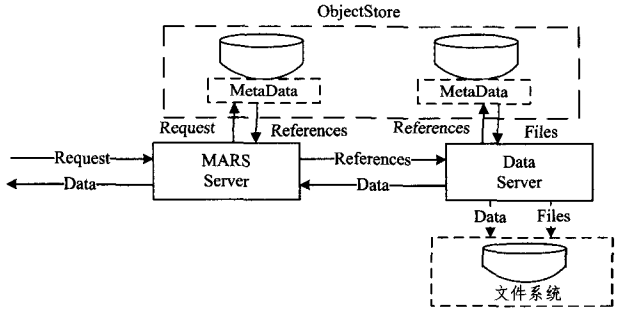


图 1　MARS 服务端架构

MARS 服务端基于面向对象设计方法[12-13]管理数据的逻辑视图与物理视图元数据,利用对象数据库实现持久化存储。在 MARS 中,气象场是能够获取的最小单位对象,由不同的元数据属性定义(见表 1),在 MARS Server 中,系统利用数据立方体技术按照不同属性对气象场元数据进行组合来完成索引计算。数据立方体是多维模型以 n 维空间视角观察数据的表现,其核心是有效地进行多个维集合上的聚集。当数据立方体规模较大时,聚集计算的系统开销也随之增大。为了减少数据立方体规模,降低系统计算代价,MARS 选择具有较强区分能力的属性定义成多维空间的层次结构 MarsTree(见图 2),利用前缀共享和元祖共享技术去除冗余数据[14]。MarsTree 采用类似 R-Tree[15] 的树形结构存储管理非聚集元数据属性,树的中间节点存储非聚集属性值以及指向下一节点的指针。叶节点存储需要进行分组聚集计算的元数据属性及属性值集合的向量链表(Shape)的索引标识(DSReferenceID)。

表 1　一组元数据属性描述气象场

场属性	取值	场属性	取值
Class	EI	Levtype	ML
Domain	G	Param	T
Stream	Oper	Level	10
Expver	0001	Date	20100101
Type	AN	Time	0000

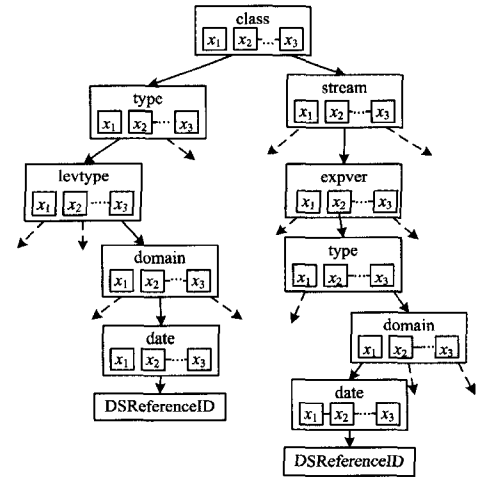


图 2　MarsTree 结构

MARS 将用于构造数据立方体的聚集维及维成员有序地存储在 Shape 结构中(见图 3),通过 Shape 构造数据立方体来进行聚集计算以获得索引值,并根据 DSReferenceID 和计算得到的索引值完成逻辑视图向物理视图的映射。物理视图元数据的存储由 Layout 结构实现,在存储方式上,Layout 采用如图 4 所示的线性数组结构。

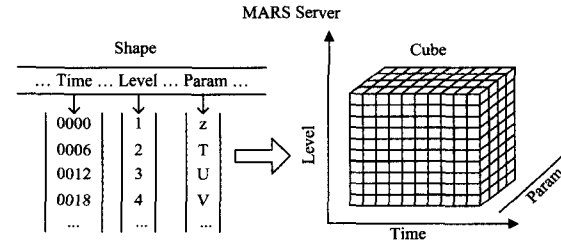
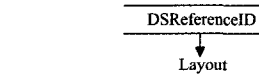


图 3　Shape 构造立方体



Index	Offset	Length	File
0	0	12000	* Ptr
1	12000	27000	* Ptr
2	39000	12000	* Ptr
3	51000	27000	* Ptr
4	0	12000	* Ptrl
5	12000	27000	* Ptrl
6	39000	12000	* Ptrl
7	51000	27000	* Ptrl

图 4　Layout 结构

鉴于数组结构可以直接利用数组下标进行寻址，因此MARS把反映气象场的真实物理存储信息存放其中，包括偏移量、数据大小以及存储路径等，利用在Data Server中计算获得的索引值与Layout数组下标唯一映射完成气象场物理存储信息的检索。

2.2 超立方体聚集计算

在MARS Server中，系统对Shape中的聚集维及维成员直接构造数据立方体，以Time，Level和Param这3个属性为例，构造如图3所示的立方体。在实际业务运行中，Shape存储的聚集维不止这3个，由此组织构造的数据立方体是一个超立方体(Hypercube)。超立方体聚集计算主要包括两部分：坐标转换和坐标聚集计算。

2.2.1 坐标转换

聚集维成员采用整数编码方式把Shape中各个元数据属性值集合转换为Hypercube中各个维的整数坐标。整数编码如下：

定义1 给定任一属性值集合 $A=\{a_0,a_1,\cdots,a_n\}$，构造一个映射函数：$f:A\rightarrow N$，使得 $f(a_i)=i$，其中，$N=\{1,2,\cdots,n\}$，且总是存在 $f(a_{i+1})=f(a_i)+1$。

聚集维成员通过整数编码坐标转换后，再由以下Hypercube坐标聚集计算出物理映射视图关键索引值。

2.2.2 坐标聚集计算

Hypercube在逻辑上采用多维数组的表现形式，其坐标聚集计算过程采用多维空间线性化方法[16]，按照行优先顺序聚集成索引执行坐标映射，定义如下。

定义2 给定Hypercube结构中包含 N 个维，且维尺寸集合为 $D=\{D_1,D_2,\cdots,D_i,\cdots,D_n\}$，其中 $D_i\in N$，且 $D_i>0$；那么，多维空间中某一点坐标 $(x_1,x_2,\cdots,x_j,\cdots,x_n)$ 映射成索引聚集计算公式为：

$$Index_{(x_1,x_2,\cdots,x_j,\cdots,x_n)}=\sum_{1}^{n}(x_j\prod_{j+1}^{n}D_i)$$

在上述索引获取过程中，中间计算结果不持久化存储，MARS采用这种不物化的方式既减少了索引建立时间，又节省了系统空间开销。此时，MARS通过之前检索MarsTree获取的DSReferenceID在对象数据库中取得对应的Layout结构，并结合坐标计算完成物理视图元数据的两级检索。

3 区域查询及优化

3.1 RQ相关问题分析

在气象领域中，针对气象科学数据的检索操作，由于单一的数据不具有实际研究价值，一次或几次的点查询并无意义，因此在使用MARS检索气象科学数据时，最常见的操作是区域查询，其主要流程如下。

1)MARS客户端请求处理。检查服务请求的元数据参数，并向服务端提交。

2)MarsTree结构检索。系统执行MarsTree的层次检索对比直到树的叶节点，获取对应的DSReferenceID。

3)Shape结构检索。根据DSReferenceID在对象数据库中查询对应的Shape结构，并通过Shape结构中不同的元数据属性及其属性值集合构造Hypercube。

4)Hypercube聚集计算。通过2.2节的坐标转换和坐标聚集计算以获得索引值index。

5)Layout结构检索。根据DSReferenceID和索引值index定位Layout结构中的元数据信息，查询真实气象场数据的存储情况，并向客户端发送检索结果和数据。

区域查询是在相应维上的维值在某个区间范围内的一组立方体元祖的聚集值查询，由大量的点查询组成，其一般形式如下：

$$RQ=(d_1,*,\cdots,d_i,\cdots,*,\cdots,\{d_{j1},d_{j2},\cdots,d_{jk}\},*,\cdots,d_n)$$

其中，$*$ 代表某一维的维成员全集。在区域查询操作中，Hypercube聚集计算在整个MARS系统中占据比例最大。当系统检索的气象科学数据呈海量时，区域查询最终会转化为大量的点查询，由于每个点查询都会执行一次索引计算，大范围区域查询无疑会造成系统元数据响应时间激增，导致系统检索效率较低。因此，本文针对此情况设计了CTRQ算法优化区域查询方式，有效地降低了系统计算代价，提高了系统检索效率。

3.2 CTRQ算法设计

Hypercube聚集计算的前提是Shape结构中的聚集维及维集合完成坐标转换。由于整数编码的特性，坐标转换后的Hypercube实质上是一个纯数字立方体。鉴于历史气象数据是比较完整地通过MARS归档到文件存储系统中的，一般不会出现Hypercube聚集的某个离散点数据缺失，因此此时的Hypercube结构是一个完全立方体，即立方体聚集计算获得的索引值都一一映射Layout结构中的元数据项。结合3.1节的问题分析，提出CTRQ算法，其基本思想是在系统执行大范围区域查询操作时，结合数学补集思想与多路数组聚集原理降低元数据计算量，令原始请求“大数据”索引计算转变为求其补集的“小数据”索引计算，如图5所示。此时，气象场元数据存储信息的查询是通过“小数据”计算获得的索引值在Layout结构中二次求补完成的。

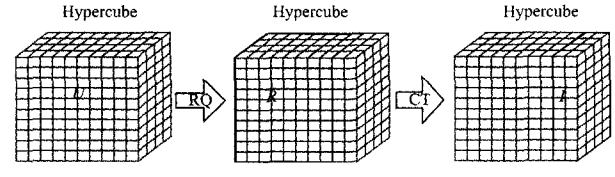


图5 补集转换

假设完全立方体结构中各个维的所有维成员集合为全集 U，且维尺寸集合 $D=\{D_1,D_2,\cdots,D_i,\cdots,D_n\}$。大范围区域查询请求的元数据属性值坐标转换后的集合 $R=\{R_1,R_2,\cdots,R_i,\cdots,R_n\}$，其中 $R_i\leqslant D_i$；令 U 中所有不属于 R 的属性值集合为 I。那么，针对大范围区域查询，通过执行集合 R 中“过半求补”的方式，完成将集合 R 的聚集计算转换成集合 I 的聚集计算。此时，集合 I 完成索引映射需要的坐标聚集计算的次数为：

$$CTRQ\_sum=\begin{cases}R_1\times R_2\cdots\times R_i\cdots\times R_n, & RQ\leqslant\dfrac{U}{2}\\ \sum_{1}^{n}(D_i-R_i)\prod_{0}^{i-1}R_j\prod_{i+1}^{n}D_k, & R_0=1,RQ>\dfrac{U}{2}\end{cases}$$

CTRQ 算法是基于多路数组聚集方法实现的，其实质是划分为多个子立方体的聚集计算。如图 6 所示，数据立方体包含 4 个聚集维，分别为 date，step，level，param，区域查询请求的聚集维成员在图中以灰色标注，将整个数据立方体聚集维主要划分为两部分：查询块和取余块。若区域查询表示为：$RQ=(\{date1,date3\},step1,level,param)$，根据 CTRQ 算法转化为以下 4 个部分：

$$CTRQ_1=(date2,step1,level,param) \tag{1}$$

$$CTRQ_3=(date1,step2,level,param) \tag{2}$$

$$CTRQ_2=(date2,step2,level,param) \tag{3}$$

$$CTRQ_2=(date3,step2,level,param) \tag{4}$$

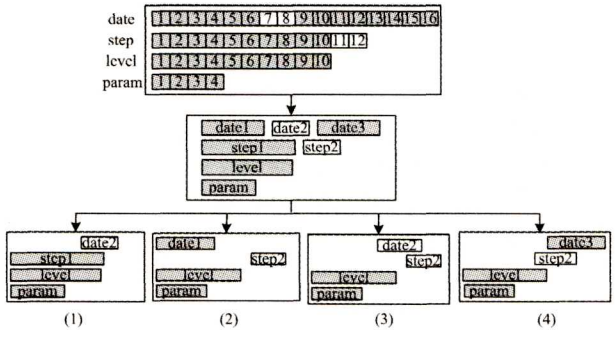


图 6　CTRQ 多路数组聚集

此时，原始区域查询索引映射转换为以上补集查询 4 个部分的计算，再由定义 2 的索引映射公式依次计算块内坐标聚集索引值，但是由此获得的索引值在块内有序，在块间乱序，不利于执行 Layout 结构中元数据信息的二次求补。为了加快 Layout 结构中气象场物理存储信息的检索，必须对索引值结果重新进行排序。

为了调整索引值排序结果，解决索引值乱序问题，CTRQ 算法利用静态数组结构对计算结果进行调整，该结构利用数组下标直接寻址访问的自然设计，使执行结果二次存放到对应的下标位置，这种方式不仅避免了索引值结果重新进行排序的巨大系统开销，而且保证了结果的有序性。具体 CTRQ 算法表述如算法 1 所示。

算法 1　CTRQ 算法

输入：区域查询请求元数据属性集合 R

输出：Layout 结构中的气象场数据物理存储信息

```
Begin
Calculate_Len(R);//计算 R_len:R_len=R1×R2×…Ri×…×Rn;
Calculate_Len(U);//计算 U_len:U_len=D1×D2×…Di×…×Dn;
  if(R_len>U_len/2){
  Dim_ divide(U);//聚集维分块:查询块、补集块
  //在循环 U 中聚集子块并做以下判断;
    if (U_res in R){ // U 中子块属于 R,不做任何操作;
        } else{
  //计算补集子块索引值;
    res[U_len-R_len]=Calculate_Index(U_res);
        }
    for(inti=0;i<res. length;i++){
//调整结果,根据数组下标把索引存放其中,其余置-1;
        Index(res[i],-1);
          }
//检索 Index 元素为-1 且 Layout 对应项非空的元数据对象;
      retrieve(Layout,Index);
        }else{
// 原始请求集合 R 的索引计算;
        R_index[R_len]=Calculate_Index(R);
//直接检索由原始请求集合 R 索引映射的 Layout 数组项;
        retrieve(Layout,R_index);
        }
End
```

3.3　CTRQ 并行优化

为了进一步提高系统的检索效率，改变 Hypercube 结构聚集索引计算的串行模式，本文结合文献[17]提出的基于聚集维共享排序集合划分的并行计算方法实现 CTRQ 算法的并行化。该并行算法主要是在 CTRQ 算法的基础上，结合该程序结构特性以及 CPU 访存特点，对程序核心部分做了高效性能优化，通过重新划分任务和调整数据结构，设计了一种数据级并行算法，该算法采用 OpenMP 并行编程模型[18]实现。

OpenMP 是一种可移植、可伸缩的基于共享内存式的编程模型，它支持多种系统如 UNIX 和 Windows NT 下的 C，C++以及 Fortran 语言。根据 OpenMP 线程执行模型，CTRQ 并行算法采用了一种高效任务分配方式，即静态任务分配方式，初始时为每一个线程分配固定的维成员数量，实行多线程并行执行，以此提高执行速度，最后进行归约操作保证结果的正确性。

由于串行的 CTRQ 算法将消耗很大一部分时间来处理循环计算，因此针对 CTRQ 并行算法设计，采用循环并行化形式基于聚集维共享排序集合来划分实现，如图 7 所示。该并行算法沿用了上述静态数组结构对索引值的临时存储，把原先存储计算结果的数据结构直接调整成为静态数组结构，这样不仅提高了索引的存取速度，而且避免了线程间的访存冲突，保证了结果的一致性和有序性，为下一步检索 Layout 结构元数据节省了重新排序的时间开销。CTRQ 并行化算法如算法 2 所示。

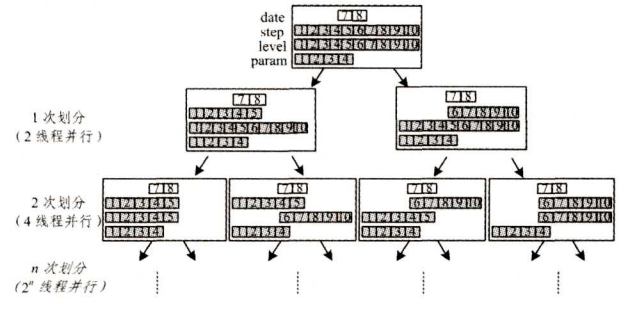


图 7　基于聚集维共享排序集合划分的并行模式

算法 2　CTRQ 并行化算法

输入：区域查询请求元数据属性集合 R；线程数目 Thread_num

输出：Layout 结构中气象场数据具体物理存储信息

```
Begin
  Calculate_Len(R);//计算 R_len:R_len=R1×R2×…Ri×…×Rn;
  Calculate_Len(U);//计算 U_len:U_len=D1×D2×…Di×…×Dn;
  if (R_len>U_len/2){
    Dim_ divide(U);
```

```
  //在循环 U 中聚集子块并做以下判断;
    if (U_res in R){
          } else{
#pragma omp parallel for …
  //OpenMP 并行语句标识,表明以下语句并行执行;
    Index[U_len]=Calculate_Index(U_res);
//利用索引映射公式计算索引值,根据 Index 数组下标直接存放;
          }
#pragma omp parallel for …
    retrieve(Layout,Index);
          }else{
#pragma omp parallel for …
  R_index[R_len]=Calculate_Index(R);
  retrieve(Layout,R_index);
          }
End
```

4 实验及结果

4.1 实验描述

利用所提出的 CTRQ 算法及其并行化算法进行模拟实验,测试平台的主要配置如表 2 所列,在以下模拟实验中,处理器关闭了超线程功能。

表 2 实验节点配置

名称	值
CPU	Intel(R) Xeon(R) CPU E5-2670,8 cores,2.60GHz
CPU 个数	2
操作系统	Linux Red Hat 4.4.5-6
编译器版本	Intel v13.0.0
OpenMP 版本	Intel v3.1

实验 1 使用表 3—表 5 这 3 个不同规模的聚集维元数据集合作为 MARS 已归档数据来执行对比测试。实验采取以下测试方式:Date 维分别选取 16,23,30 个数据进行部分查询,其他维进行全查询;针对实验数据集 1—实验数据集 3,分别执行以下的区域查询操作:

(1) $RQ(Date, Time, Step, Number, Param, Level) = (\{16,23,30\},4,5,5,10,20)$

(2) $RQ(Date, Time, Range, Step, Number, Param, Level) = (\{16,23,30\},4,5,5,5,10,20)$

(3) $RQ(Date, Time, Range, Number, Param, Level, Longitude, Latitude) = (\{16,23,30\},4,5,5,10,20,10,10)$

实验 2 选用表 6 中的实验数据完成 CTRQ 并行实验(包括数据结构调整),分别使用不同粒度的线程执行以下的区域查询:

$RQ(Date, Time, Range, Number, Param, Level, Longitude, Latitude) = (\{16,23,30\},4,5,10,10,60,10,10)$

上述区域查询表示 Date 维成员分别取 16,23,30 个数据查询(数据集 1,数据集 2,数据集 3),其余维全查询。

表 3 实验数据集 1

属性名称	属性值数量	属性名称	属性值数量
Date	31	Number	5
Time	4	Param	10
Step	5	Level	20

表 4 实验数据集 2

属性名称	属性值数量	属性名称	属性值数量
Date	31	Number	5
Time	4	Param	10
Range	5	Level	20
Step	5		

表 5 实验数据集 3

属性名称	属性值数量	属性名称	属性值数量
Date	31	Param	10
Time	4	Level	20
Range	5	Longitude	10
Number	5	Latitude	10

表 6 实验数据集 4

属性名称	属性值数量	属性名称	属性值数量
Date	31	Param	10
Time	4	Level	60
Range	5	Longitude	10
Number	10	Latitude	10

4.2 结果分析

根据以上实验描述,针对大范围区域查询的 CTRQ 算法,实验 1 的测试结果如图 8 所示。

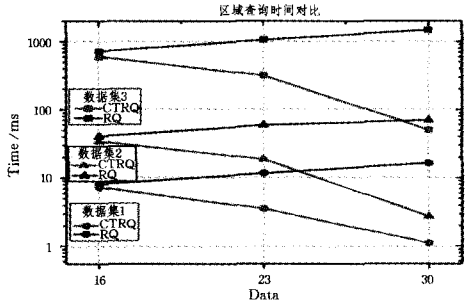


图 8 原始 RQ 和 CTRQ 运行时间对比

图 8 示出了不同规模下的原始区域查询与 CTRQ 算法查询的运行时间,从图 8 中可以看出,在数据集一定的情况下,针对原始 RQ 算法,区域查询范围越大,元数据检索响应时间越长,而对于本文提出的 CTRQ 算法,区域查询范围越大,经补集转换后的元数据检索响应时间越短,其主要原因在于在 MARS 执行区域查询的过程中,元数据聚集索引计算耗时占整个系统耗时的主要部分,而元数据检索的计算量由区域查询的范围确定,大范围区域查询的情况下,原始 RQ 算法获得元数据索引的计算量较大,而 CTRQ 算法是 RQ 算法的元数据索引补集计算,在很大程度上减少了元数据计算量,从而减少了系统检索的响应时间。实验结果表明,区域查询范围与气象场数据检索耗时反相关,且随着问题规模的增加,该算法具备高度的可扩展性。

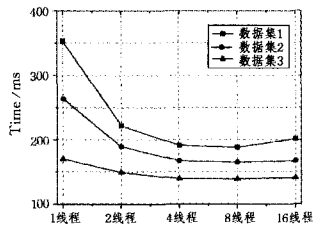


图 9 CTRQ 并行化算法运行时间

图 9 示出了实验 2 在不同规模下的串行 CTRQ 算法(图中 1 线程)与并行化算法的运行时间。从实验结果可以看出,在 CTRQ 算法的基础上,其并行算法进一步降低了系统的响

应时间。图 10 示出了在不同的任务规模下 CTRQ 并行化算法相对上述串行算法的加速比。当线程数为 2 时,最大取得 1.6 倍的加速比;线程数为 4 时,最大取得 1.8 倍的加速比;随着线程数量的增加,加速比随之增大,且在线程数量为 8 时达到峰值,最大取得 1.9 倍的加速比。此后,随着线程的有效计算时间以及线程切换开销占整个运行时间的比例增大,加速比逐渐减小。

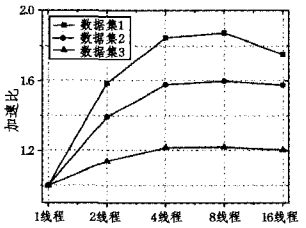


图 10　CTRQ 并行算法加速比

结束语　本文深入分析了 MARS 平台架构,针对该系统执行大范围区域查询时效率不高的问题,设计了一种基于数学补集思想的面向大规模数据的区域查询方法。实验表明,在数据集一定的情况下,区域查询范围与系统响应时间成反相关。另外,本文给出了大规模数据下补集转换区域查询结合 OpenMP 编程模型的并行化算法实现,进一步降低了系统检索的响应时间,提高了数据的查询效率,为基于高性能文件存储系统的 MARS 数据归档与检索奠定了基础,同时,对应用其他领域的超立方体技术优化方法也具有参考价值。

参 考 文 献

[1] RAOULT B. Architecture of the new MARS server[EB/OL]. [2015-06-01]. http://old. ecmwf. int/archive/publications/manuals/mars/server. pdf.

[2] SARAWAGI S, AGRAWAL R, MEGIDDO N. Discovery-driven Exploration of OLAP Data Cubes[J]. Lecture Notes in Computer Science, 1998, 1377: 168-182.

[3] HAN J, KAMBER M. Data Mining: Concepts and Techniques. Second Edition[J]. San Francisco, 2006(1): 1-25.

[4] GRAY J, CHAUDHURI S, BOSWORTH A, et al. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals[J]. Data Mining and Knowledge Discovery, 1997, 1(1): 29-53.

[5] SHAPIRO M A, THORPE A J. THORPEX International Science Plan1[J]. Boletín De La Organización Meteorológica Mundial, 2004, 6(11): 238-242.

[6] SHEN W H, ZHAO F, GAO H Y, et al. The construction of national meteorological archival and retrieval system [J]. Journal of Applied Meteorological Science, 2004, 15(6): 727-736. (in Chinese)
沈文海,赵芳,高华云,等. 国家级气象资料存储检索系统的建立[J]. 应用气象学报,2004,15(6):727-736.

[7] HO C T, AGRAWAL R, MEGIDDO N, et al. Range queries in OLAP data cubes[J]. ACM Sigmod Record, 1970, 26(2): 73-88.

[8] HONG S, SONG B, LEE S. Efficient Execution of Range-Aggregate Queries in Data Warehouse Environments[M]// International Symposium on Requirements for Poultry Virus Vaccines. S. Karger, 1974: 299-310.

[9] AGARWAL S, AGRAWAL R, DESHPANDE P M, et al. On the computation of multidimension alaggre gates[C]// VLDB. 1996: 506-521.

[10] ZHAO Y, DESHPANDE P M, NAUGHTON J F. An array-based algorithm for simultaneous multidimensional aggregates [C]// Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data. 1997: 159-170.

[11] XUE Y S, HUANG Z H, DUAN J J, et al. An Efficient Method for Parallel Multi-Dimensional Join and Aggregation[J]. Journal of Computer Research and Development, 2004, 41(10): 1661-1669. (in Chinese)
薛永生,黄震华,段江娇,等. 一种并行处理多维连接和聚集操作的有效方法[J]. 计算机研究与发展,2004,41(10):1661-1669.

[12] BOOCH G. Object-oriented development[J]. IEEE Transactions on Software Engineering, 1986, 12(2): 211-221.

[13] WU P, XU H P, CHEN H G. Application of Object-oriented for Metadata Research[J]. Journal of Tongji University (Natural Science), 2010, 38(11): 145-151. (in Chinese)
吴萍,许惠平,陈华根. 面向对象方法在元数据研究中的应用[J]. 同济大学学报(自然科学版),2010,38(11):145-151.

[14] SHENG L I, WANG S. Star Cube——An Approach to Implementing Data Cube Efficiently[J]. Journal of Computer Research & Development, 2004, 41(4): 587-593.

[15] GUTTMAN A. R-trees: A dynamic index structure for spatial searching[C]// Proc. of the ACM SIGMOD International Conference on Management of Data. 1984: 47-57.

[16] LI J, ROTEM D, SRIVASTAVA J. Aggregation Algorithms for Very Large Compressed Data Warehouses[C]// Proceeding of the 25th VLDB Conference. 1999: 651-662.

[17] SONG S L, SONG J Q, REN K J. Design of a parallel algorithm for data cube of MARS[J]. Computer Engineering & Science, 2014, 36(12): 2410-2417. (in Chinese)
宋石磊,宋君强,任开军. 气象数据归档与查询系统超立方体结构并行算法设计[J]. 计算机工程与科学,2014,36(12):2410-2417.

[18] SATO M. OpenMP: parallel programming API for shared memory multiprocessors and on-chip multiprocessors[C]// International Symposium on System Synthesis. IEEE, 2002: 109-111.

(上接第 22 页)

[3] WANG M, ZHANG W, DING W, et al. Parallel Clustering Algorithm for Large-Scale Biological Data Sets[J]. PLoS ONE, 2014, 9(4): 13-15.

[4] W W M, KIRK D B. Programming Massively Parallel Processors [M]. New York: Morgan Kaufmann, 2012: 120-128.

[5] EKANAYAKE V, GUNARATHNE T, QIU J. Cloud Technologies for Bioinformatics Applications[J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(6): 998-1011.

[6] BUSTAMAM A, BURRAGE K, HAMILTON N A. Fast Parallel Markov Clustering in Bioinformatics Using Massively Parallel Computing on GPU with CUDA and ELLPACK-R Sparse Format[J]. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), 2012, 9(3): 234-240.