doi:10.11887/j.cn.201603029 http://journal.nudt.edu.cn

HBase中半结构化时空数据存储与查询处理*

封孝生,张 翀,陈晓莹,唐九阳,葛 斌
(国防科技大学 信息系统工程重点实验室,湖南 长沙 410073)

摘 要:针对在HBase中如何进行有效的半结构化时空数据存储和查询问题展开研究,对该问题进行形式化描述,并利用半结构化处理方法TwigStack提出HBase的半结构化时空数据存储模型,在此基础上开展了半结构化的时空范围查询和*k*NN查询。在真实数据集中进行实验,与需要硬件配置较高的MongoDB进行了对比,结果表明在普通配置的机器上,所提出的半结构化时空查询算法与MongoDB性能相近,在实际中具有优势。

关键词:时空数据;半结构化;HBase;时空范围查询;*k*NN查询
中图分类号:TP391 **文献标志码**:A **文章编号**:1001-2486(2016)03-174-08

Storage and query processing for semi-structured spatio-temporal data in HBase

FENG Xiaosheng, *ZHANG Chong*, *CHEN Xiaoying*, *TANG Jiuyang*, *GE Bin*
(The Key Laboratory of Information System Engineering, National University of Defense Technology, Changsha 410073, China)

Abstract: A study about how to effectively achieve semi-structured spatio-temporal data storage and query in HBase was carried out. The formal description of the problem was issued; the HBase semi-structured spatio-temporal storage model was proposed by using a semi-structured approach TwigStack. On this basis, semi-structured spatio-temporal range query and *k*NN queries were carried out. An experiment was made in terms of real data and a comparison was made with MongoDB which needs higher hardware configuration, the results show that the performance of semi-structured spatio-temporal query algorithm is similar to MongoDB in the machine with general configuration, so that it has the advantage in the practical application.

Key words: spatio-temporal data; semi-structured; HBase; spatio-temporal range query; *k*NN query

随着遥感、通信等技术不断深入发展与应用,遥感数据规模呈几何级增长,海量遥感数据的高效的面向时空属性的检索对数据库时空查询处理技术提出了挑战。前序工作中大部分都是考虑如何索引和检索时空属性,而包含了检索关键字的工作又仅仅是考虑结构化的情况,然而对于检索遥感数据,问题背景发生了变化。通常,遥感数据本身并不是文字直接表现的数据,如卫星遥感图像、气象云图等,因此检索遥感数据实际上是对描述遥感数据的元数据(也称编目)进行检索,而元数据是用半结构化树形结构,如可扩展标记语言(eXtensive Markup Language, XML)文件进行描述,那么问题背景就变成了如何针对海量的半结构化数据进行时空+半结构化查询语言(如XPath)的检索。

表1显示了一份遥感元数据样例,用户可以通过声明查询地理范围与时间范围以及半结构化查询条件来查询遥感数据。例:查询区域R(以点

表1 遥感元数据样例

Tab. 1 Example of remote sensing metadata

```
<遥感数据编目>
  <类型>卫星遥感图像</类型>
  <传感器>
    <类型>CCD 相机</类型>
    <分辨率>50</分辨率>
    <光谱范围>
      <上限>0.927 μm</上限>
      <下限>0.9 μm</下限>
    </光谱范围>
  </传感器>
  <地理范围>
    <经度>89.223 799, 90.156 234</经度>
    <纬度>28.123 457, 32.734 85</纬度>
  </地理范围>
  <时间跨度>
    1347898654, 1358907896
  </时间跨度>
  <路径>/path/1125.tif</路径>
</遥感数据编目>
```

* **收稿日期**:2016-03-07
基金项目:国家自然科学基金资助项目(61303062,71331008)
作者简介:封孝生(1971—),男,湖北蕲春人,副教授,硕士,硕士生导师,E-mail:xsfeng@nudt.edu.cn

c 为圆心,r 为半径)内,时间为 2 周以内,且内容满足"/遥感数据编目[类型 = '卫星遥感图像' and //类型 = 'CCD 相机']//路径"这样条件的所有遥感数据。需要注意的是,此处半结构化查询是 XPath 中的 twig 查询。

针对这种既有时空查询,又有树状结构 + 内容的混合查询,目前还没有在海量大数据情况下的相关查询处理技术。尽管很容易想到利用 MongoDB[1] 进行存储与时空查询,然而,第一,MongoDB 并不支持时间区间的存储与查询;第二,虽然 MongoDB 面向 JSON(JavaScript object notation)以文档为中心进行存储,但对复杂的 twig 查询(如表 1)效果不佳;第三,虽然 MongoDB 查询效率较高,但这是建立在高内存占用比例基础之上的,对硬件投入较大。

相对而言,HBase[2] 作为高性能、列存储、可伸缩、实时读写的分布式数据库,可支持集群存储海量数据,极大弥补了传统数据库和 MongoDB 的不足。然而 HBase 仅支持键值对(key-value)模式的查询,对多维复杂查询能力支持不足,因此现有的工作基本围绕如何提高 HBase 多维或空间查询性能展开,主要集中研究建立空间数据的索引结构。一方面,这些工作没有考虑空间对象的时间维属性。对于时空对象,时间和空间属性是密不可分的,简单将时间维作为另一种空间维处理,会由于数据性质不同而造成分布不均以致降低性能。另一方面,现有工作基本上受限于 HBase 平台的架构,无法取得性能上的突破。

针对上述问题,本文从存储与查询半结构化时空数据的需求出发,采用 HBase 技术,并为切实提高 HBase 的时空查询性能,提出了充分利用 HBase 中唯一的索引机制 meta 来设计时空索引结构——HBase 的半结构化时空(HBase Semi-Structured Spatio-Temporal, HSSST)数据存储与索引模型,并在此结构基础上,设计了范围查询和 kNN 查询算法,以及这两种查询的并行化算法。

1　相关工作

目前传统的基于结构化查询语言(Structured Query Language, SQL)的时空数据存储与查询处理提供了一个简单的形式结构将信息存储和管理在表结构中。数据存储和检索可以通过简单的表操作来实现,然而,基于 SQL 难以满足高效率的数据插入和查询需求,更难以满足 TB 级的数据负担。相对而言,半结构化的 NoSQL 数据库存储能够支持大规模的时空数据操作。

MongoDB 是一种可扩展、性能高、开源、模式自由、面向文档的数据库,归类为 NoSQL 的数据库。MongoDB 能够直接支持空间类数据存储以及建立索引满足相应的功能需求,例如:GeoJSON 可采用 geospatial (2d) index 对存储在文档中的空间数据坐标对进行索引,随后计算地理哈希(Geohash)值(Geohash 是基于经纬度的地理编码)。其他 NoSQL 数据库虽然并没有直接支持空间数据编码的方法,但可以通过 Geohash 方法进行扩展,而在 MongoDB 中,将时间维度与空间维度结合进行查询是其本身不支持的。

另外一种 NoSQL 数据库——HBase,能够适应大数据时空数据存储和高效率的数据插入,但只支持单一的数据检索模式,而时空数据检索要求返回在特定的空间和时间范围的数据对象,这种检索要求是传统的 HBase 无法实现的。

文献[3]提出了 MD-HBase,一种应用于平台即服务(PaaS)的多维查询方法。它使用 K-D 树和四叉树(quad-tree)进行空间的分割,以及采用 Z-曲线将多维数据转换为一维数据,支持多维范围和最近邻查询。文献[4]提出了一种基于 R^+ 树的键值构建方案,称为 KR^+ 树,并在 KR^+ 树的基础上,设计了空间查询算法——kNN 查询和范围查询,并将其在 HBase 和 Cassandra 上进行运用。实验结果表明,构建的 KR^+ 树优于 MD-HBase[3]。文献[5]提出适用于分布式多维数据的索引——EDMI (efficient distributed multi-dimensional index),其中包括两个层次:上层采用 K-D 树把空间分割成许多子空间;底层中,每一个子空间分别对应一个 Z-order 搭配 R 树前缀(ZPR 树)。ZPR 树可以避免由 R 树节点中多维数据引起的最小边界矩形(Minimum Bounding Rectangles, MBRs)重叠。相较于其他的 packed-R 树和 R^* 树,ZPR 树具有更好的查询性能。基于 HBase 平台上的测试结果表明,EDMI 在点查询、范围查询和 kNN 查询方面都具有优越性。文献[6]提出一种针对 HBase 的数据模型——HGrid。该数据模型基于混合索引结构,融合了四叉树和常规的网格(grid)结构,支持范围查询和 kNN 查询。文献[7]提出一种基于 HBase 的可扩展的数据存储机制——HBaseSpatial,通过与 MongoDB 和 MySQL 作比较,实验结果表明该方法能有效提高大空间数据的查询速度。

上述方法的研究对象为空间数据,而在实际应用中,空间数据的时间属性是不容忽视的。文献[8]从设计 HBase 的模式(schema)出发设计行

键(rowkey)来满足多维或空间查询,并结合 Bloom 过滤器来快速过滤关键词从而实现时空数据的关键词查询。文献[9]则进一步研究了 HBase 的内部索引机制,提出 STEHIX (spatio-temporal HBase index),适合于 HBase 的两级架构,其利用了 meta 链表分别索引了空间和时间,在此基础上设计了时空范围查询和 kNN 查询,以及对应的并行算法。

在诸多的针对时空数据的应用中,移动对象也是常见的研究对象。文献[10]提出了一种基于 HBase、采用 R 树构建空间索引以及用于遍历空间的希尔伯特曲线(Hilbert curve)的混合索引结构。它支持多维范围查询和 kNN 查询,尤其是在不均匀分布的数据中应用效果优于 MD-HBase[3] 和 KR$^+$[4]。虽然此方法考虑了时空条件,但仍然无法满足对时空数据更加多样的时空检索应用。

选择半结构化时空数据的存储方法还应考虑其转化为其他形式数据得以应用的性能,文献[11]在链接开放数据(Linked Open Data, LOD)的研究中,对比了 HBase、逗号分隔值(Comma-Separated Values, CSV)、XML 转化为资源描述框架(Resource Description Framework, RDF)的三种方法的转化性能,结果表明 HBase 的映射转化效率最高。

2 问题描述与预备知识

2.1 问题描述与定义

含有时空信息的半结构化数据可以方便灵活地描述遥感信息产品等数据。一般半结构化时空数据可以描述为 $<\text{uid},(x_l,y_l,x_u,y_u),(t_s,t_e),\text{T}>$,解释如下:

1) uid 是半结构化时空数据文档的唯一标识;

2) (x_l,y_l,x_u,y_u) 是文档 uid 描述的数据对应的空间范围;

3) (t_s,t_e) 是文档 uid 描述的数据对应的时间范围;

4) T 是除去时空信息的半结构化数据,既含有结构信息也含有内容(值)信息。

对于面向半结构化时空查询定义为:

1) 时空范围查询。给定时间范围 (t_{qs},t_{qe}),空间范围 $R_q=(c,r)$ (c 为圆心,r 为查询半径),半结构化 XPath 查询 xq,查询所有满足 $(t_{qs},t_{qe})\cap(t_s,t_e)\neq\varnothing$, $R_q\cap(x_l,y_l,x_u,y_u)\neq\varnothing$,且满足 xq 的文档;

2) kNN 查询。给定时间范围 (t_{qs},t_{qe}),空间位置 $q=(x_q,y_q)$,以及自然数 k 与半结构化 XPath 查询 xq,查询所有满足 $(t_{qs},t_{qe})\cap(t_s,t_e)\neq\varnothing$ 且满足 xq、距离 q 最近的 k 个文档。

2.2 TwigStack 算法

TwigStack 算法是一种有效解决 XML 结构查询的方法,尤其是针对小枝连接(twig join)查询。主要思想是首先将 XML 结点进行区域码(region code)编码,形成的编码格式为:①若该结点为非值结点(非叶子结点),则为(XML 文档 ID, 起始访问编码: 结束访问编码, 层次码);②若该结点为叶子结点,则为(XML 文档 ID, 访问编码, 层次值)。利用访问编码能否覆盖可以判断祖先-后代(ancestor-descendant)关系,再加上层次码就可以判断父子(parent-child)关系。然后将每个标签绑定一个数据流,该数据流是以这个标签为实例化的所有 XML 结点的集合。实施查询时,将查询结构树分解为各个路径,按照每条路径的查询条件从根结点至叶结点的顺序访问各个结点,并将数据流压入栈中,然后根据编码规则判断关系形成各个路径的查询结果,最后再将所有的路径进行合并形成最终结果。

举例来说,图 1 显示了对表 1 中的 XML 进行编码后的结果,因为研究是针对时空条件进行查询,因此对时空属性所在的结果不做编码。假设该 XML 的文档 ID 为 1,根结点"遥感数据编目"的区域码为(1, 1∶24, 1),其中第 1 个数字"1"代表该 XML 文档 ID,第 2 个数字为进行先序遍历的访问顺序号,第 3 个数字表示为访问完最后一个结点"路径"后返回至跟结点的访问序号,第 4 个数字代表处于第 1 层。由此编码之后,给定任意两个结点的编码就可以判断出这两个结点在该 XML 文档中的结构关系。如,"光谱范围"(1,

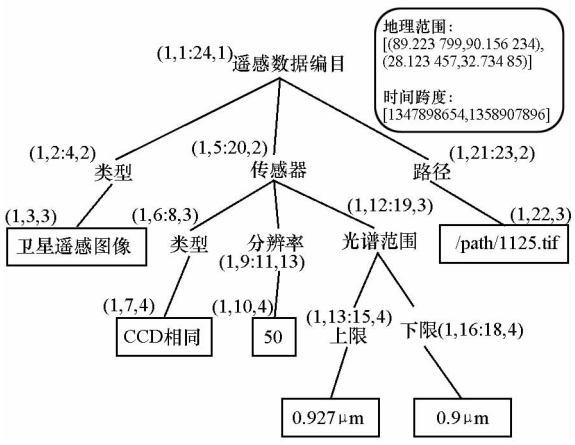

图 1 TwigStack 算法编码示例

Fig. 1 Example of TwigStack coding

12∶19, 3)与“上限”(1, 13∶15, 4),由于 12 < 13 < 15 < 19,所以“光谱范围”肯定是“上限”的祖先结点,再加上层次码仅差 1,则进一步判断“光谱范围”肯定是“上限”的父节点,又如“光谱范围”(1, 12: 19, 3)与“50”(1, 10, 4)则不存在结构上的关系,因为 10 没有被区间(12, 19)覆盖。

对于查询条件“/遥感数据编目[类型 = ‘卫星遥感图像’ and //类型 = ‘CCD 相机’]//路径”,其结构如图 2 所示,查询时,将会分解为 3 条路径分别进行查询,其中第 1 条路径“遥感数据编目”与“类型”是父子结构关系,其余 2 条为祖先 - 后代关系,查询后再根据文档 ID 进行合并,最终形成查询结果。

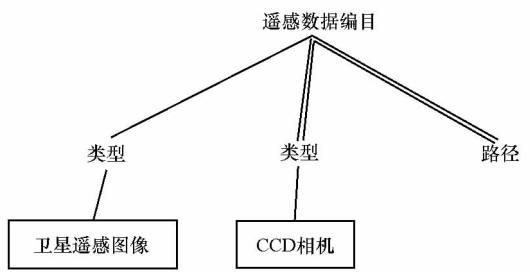

图 2　查询条件结构示例

Fig. 2　Example of query structure

为了便于表述,现将 TwigStack 算法的相关操作表示如下:

1) *encodeNode*(n),将树结点 n 进行编码;

2) *decom*(q),将结构化查询条件 q 进行按路径分解;

3) *queryPath*(p),对路径 p 进行查询,返回相应的 XML 文档 ID 以及命中的部分具体结构;

4) *mergePath*(R),对路径查询结果集合 R 进行合并并形成最终的 XML 结果集。

3　HSSST 存储与索引模型

本节详细描述 HSSST 数据存储模型。经观察可知,半结构化时空数据含有时空信息与半结构化信息两个部分,并且时空信息并不是空间点数据和时间点数据,而是空间范围数据与时间段数据,这增加了存储的复杂性。经分析,本次研究对时空信息和半结构化信息分别进行了存储,索引结构则主要根据时空信息部分建立。

3.1　时空信息存储与索引

空间部分统一采用 Hilbert 曲线进行编码,它采用分形技术,将多维空间中等分的区域(cell)中心连接起来,且每个区域仅进出一次,从而将多维空间区域映射为 1 维区间。对于给定的空间,cell 面积的大小取决于 Hilbert 曲线的阶数 λ,λ 越大,则等分的 cell 数量越多,并且每个 cell 面积越小。实验证明[12],Hilbert 是能最好地保持空间局部邻接性的填充曲线。图 3 展示了 1 阶,2 阶和 3 阶 Hilbert 曲线。

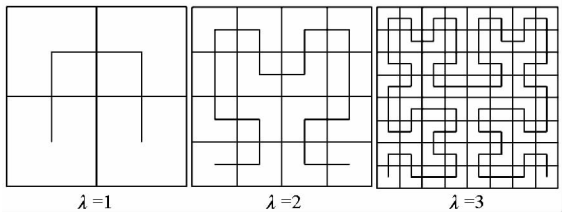

图 3　Hilbert 曲线

Fig. 3　Hilbert curves

时间部分采用周期 + 粒度方式进行索引,即首先设定一个时间周期 T(如 24 h),然后在 T 内按照粒度 t_G(如 1 h)进行划分,那么任何时间段值对 T 取模后都可以对应至周期 T 内,即计算与该时间段相交叠的粒度区间(注意,由于遥感信息的采集时间段是任意的,因此一个时间段可以对应到多个粒度区间)。举例来说,如图 4 所示,周期 T 被划分为 5 段,时间段 t_1 对应于粒度区间 $[0, t_G]$,而时间段 t_2 对应于粒度区间 $[t_G, 2t_G]$ 和 $[2t_G, 3t_G]$,类似地,时间段 t_3 对应于粒度区间 $[2t_G, 3t_G]$,$[3t_G, 4t_G]$ 以及 $[4t_G, T]$。

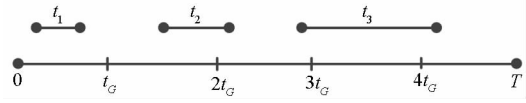

图 4　时间索引示例

Fig. 4　Example of time index

建立索引时(如图 5 所示),首先将经过 Hilbert 曲线划分的空间编码构建 HBase 的 meta

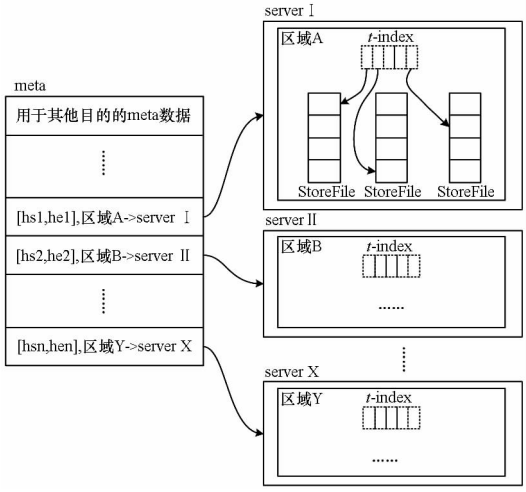

图 5　时空信息索引

Fig. 5　Spatio-temporal data index

索引,即每个 meta 的 entry 中第 1 项为 Hilbert 值,第 2 项为该 Hilbert 值所在的子空间对应的区域服务器(region server)的编号。然后在区域服务器内部建立时间索引,具体来说为:在每个区域服务器所辖的区域内将上述时间索引结构实现,每个粒度区间对应一个存储文件(StoreFile)的 item。这样,meta →区域服务器→时间索引结构→存储文件的查找顺序建立起来。

3.2 半结构化信息存储

半结构化信息的存储依靠区域码对其进行编码。首先调用 *encodeNode*( )函数将每个树结点进行编码,例如,tn_1 编码为 ec_1,然后将 tn_1 作为 HBase 中列的列键(column qualifier),ec_1 作为时空信息部分与 tn_1 形成 cell 的值。

表 2 示例了时空半结构化信息在 HBase 逻辑表中的存储方式。其中行键即为 meta 索引,存储 Hilbert 值,列中的列族(column family)用 hssst 代表,space 和 time 这两个列键对应的 cell 中存储文档中标注的空间范围和时间区间,接下来的列键就是该文档中的树结点,所对应的 cell 中存储该树结点的区域码编码。注意,逻辑表中反映不出时间索引。

表 2 时空半结构数据在 HBase 的逻辑表示

Tab. 2 Logic table of semi-structured spatio-temporal data in HBase

行键	hssst				
	space	time	tn_1	tn_2	…
Hilbert 值	(x_l, y_l, x_u, y_u)	(t_s, t_e)	ec_1	ec_2	…
……	…	…	…	…	…
……	…	…	…		

4 查询处理算法

4.1 时空范围查询

对于给定范围查询——时间范围(t_{qs}, t_{qe})、空间范围 $R_q = (c, r)$(c 为圆心,r 为查询半径)、半结构化 XPath 查询 xq,基本处理流程如算法 1 所示:首先计算空间范围 R_q 与 Hilbert 曲线的交集,即所相交的 Hilbert cell 值集合 H_q,然后将 H_q 作为条件查询 meta 索引,获得对应的区域服务器列表,然后在区域服务器中利用时间索引查询条件(t_{qs}, t_{qe})获得存储文件上对应的 item,每个 item 就是逻辑表中的行健 + 列族 + 列键 + cell 值,形式化为(rowkey, column family, column qualifier, cell),然后利用 TwigStack 算法判断 item 中半结构化信息是否符合 xq 查询条件,若符合且时空属性均符合则输出该文档作为结果。

算法 1 时空范围查询算法

Alg. 1 Algorithm for spatio-temporal range queries

Input: (t_{qs}, t_{qe})//时间条件
$R_q = (c, r)$//空间条件
xq//XPath 查询条件
Output: *xFileSet*//查询结果列表

```
1.  Begin
2.    H_q = intersectHilbert(R_q);
3.    serverList = lookupMeta(H_q);
4.    foreach server in serverList
5.      itemSet←server. lookupByTimeIndex(t_qs, t_qe)
6.    endfor
7.    P = decom(xq)
8.    foreach path in P
9.      xFile = queryPath(path)
10.     if xFile. isSatisfied(t_qs, t_qe, R_q) then
11.     xFileSet←xFile
12.     endif
13.   endfor
14.  xFileSet = mergePath(xFileSet)
15.  return xFileSet
```

对上述算法解释如下:第 2 行函数 *intersectHilbert*( )为计算空间范围 R_q 与 Hilbert cell 的交集;第 3 行为利用 meta 索引,根据 Hilbert 值集合 H_q 查找对应的区域服务器;第 4 行至第 6 行为每个服务器通过时间索引查找存储文件中对应的 item,形成 item 集合;第 7 行将 XPath 查询条件分解为多个路径的集合 P;第 8 行至第 13 行为依次根据 P 中每个路径利用 TwigStack 算法查找符合条件的文档,并判断该文档的时空属性是否符合查询条件;最后,第 14 行将符合条件的文档进行过滤合并形成结果返回。

4.2 *k*NN 查询

采用串行的方法实现 kNN 查询算法,原因如下:并行的方法对 k 值的大小有一定要求,必须超过一定值后,并行的方法才有性能提高的优势,然而通常情况下,kNN 查询时,k 值并不大;另外,并行的方法对数据分布不均匀的情况处理不好,因为这种方法只能均匀地扩大查询边框,造成大量的与结果无关的数据进入候选集,导致查询性能

下降。串行的方法由于采用优先队列技术,因此可以很好地解决数据分布不均匀的情况。

设计了增量式返回最近邻居的方法,利用优先队列依据与查询点距离的升序保存 cell 或半结构化文档,通过不断的 *enqueue*( ) 和 *dequeue*( ) 操作,返回 k 个最近的文档。下面给出距离定义。

定义 1(两点距离) 给定空间中两点 $p=(x_p,y_p)$ 和 $q=(x_q,y_q)$,距离 $d(p,q)$ 为欧式距离,即

$$d(p,q)=\sqrt{(x_p-x_q)^2+(y_p-y_q)^2} \quad (1)$$

定义 2(点与矩形之间的距离) 给定空间中点 $p=(x_p,y_p)$ 以及矩形 $R=(x_l,y_l,x_u,y_u)$,若 p 在 R 内(包括 R 的边沿),距离为 0;否则,距离 $d(p,R)$ 定义为:

$$d(p,cell_c)=\begin{cases}\min(|x_p-x_l|,\ |x_p-x_u|), & y_l<y_p<y_u\\ \min(|y_p-y_l|,\ |y_p-y_u|), & x_l<x_p<x_u\\ \min(d_1,d_2,d_3,d_4), & \text{otherwise}\end{cases} \quad (2)$$

式中,$d_1=d[p,(x_l,y_l)]$,$d_2=d[p,(x_l,y_u)]$,$d_3=d[p,(x_u,y_l)]$,$d_4=d[p,(x_u,y_u)]$。

对于给定时间范围 (t_{qs},t_{qe}),空间位置 $q=(x_q,y_q)$,k 以及半结构化 XPath 查询 xq,kNN 查询的基本思路如算法 2 所示:首先将 q 所在的 Hilbert cell 插入优先队列,该优先队列按照文档中空间范围(即矩形)或 cell(也是矩形)到 q 的距离升序排列元素,不断取出第一个元素并判断,若元素为 cell 则将 cell 中符合时间要求的文档从 HBase 中取出并插入队列,然后再将该 cell 的邻居 cell 插入队列;若元素为文档,则利用 TwigStack 算法判断该文档是否符合 xq 查询要求,符合的插入返回结果列表,并判断结果数量如果达到 k 个,则停止循环返回结果列表。

算法说明如下:第 2 行初始化优先队列 PQ,第 3 行定位查询点 q 所在的 cell,第 4 行将该 cell 压入队列 PQ,度量值为点 q 与该 cell 之间的距离,从第 5 行开始,不断将 PQ 中的元素取出(第 6 行),然后判断,若是 cell 类型(第 7 行),则利用时间索引获取该 cell 内的全部符合时间条件的文档(第 8 行),形成集合 *xFileSet*,然后将 *xFileSet* 中的每个文档压入 PQ(第 10 行),然后将当前 cell 的邻居 cell 全部压入 PQ(第 12 至第 15 行);若出队列的元素类型是文档,那么首先利用函数 *isSatisfied* 判断该文档是否满足 XPath 查询 xq(第 17 行),将满足条件的加入结果集 *Qlist*,若 *Qlist* 大小为 k,则结束循环。

算法 2　kNN 查询算法

Alg. 2　Algorithm for kNN queries

```
Input:(t_qs,t_qe)//时间条件
      q=(x_q,y_q)//空间点
      k//返回的结果数量
      xq//XPath 查询条件
Output:Qlist //查询结果列表
1.  Begin
2.  PQ=∅ //初始化升序优先队列
3.  cell_initial=coorToCell(x_q,y_q)
4.  PQ.enqueue(cell_initial,d(q,cell_initial))
5.  while PQ≠∅
6.    element=PQ.dequeue();
7.    if element is typeof cell then
8.      xFileSet=getXFilesbyTimeIndex(element,(t_qs,
        t_qe));
9.      foreach xFile in xFileSet
10.       PQ.enqueue(xFile,d(q,xFile.R))
11.     endfor
12.     CellSet=getNeighborCells(element.center);
13.     foreach cell in CellSet
14.       PQ.enqueue(cell,d(q,cell))
15.     endfor
16.   else//element is typeof xFile object
17.     if element.isSatisfied(element,xq) then
18.       Qlist.add(element);
19.       if Qlist.size()==k then
20.         return Qlist
21.       endif
22.     endif
23.   endif
24. endwhile
```

5　实验分析

实验采用 HBase - 0.98.6 搭建了一个由 11 个节点构成的 HBase 集群。每个节点为 Intel Core i3 2.40 GHz×2 CPU,2 G 内存,240 G 硬盘,操作系统为 CentOS release 6.5 64 bit,网络带宽为 100 Mbps。

MongoDB 是面向文档的数据库,它的查询优势在于将索引大部分装载于内存,从而提高检索的速度,然而这样的性能需要配置较高的硬件来支撑,在实际运行中增加了投入成本。将 HSSST 算法与 MongoDB 进行对比,目的是证明在配置一般的计算机集群上,HSSST 查询性能与对配置要求较高的 MongoDB 相近。MongoDB 的硬件环境为 1 台 Intel Xeon 3.60 GHz×4 CPU,32G 内存的机器。按照 MongoDB 的设计理念,每个半结构化

文档存储为 MongoDB 中的一个文件(document),利用自带的查询方法进行时空半结构化查询。

实验数据集选用真实遥感元数据集,数量为 100 万,类别分为气象、海洋、可见光成像等,数据中的空间范围描述呈偏斜分布,时间范围呈均匀分布。

5.1 时空范围查询实验

时空范围查询实验中分为变化选择率和变化 HBase 的节点数量。变化查询条件的时间范围、空间范围和 XPath 的选择率形成综合选择率作为测试变化点,选择率的默认值为 15%,节点数量默认为 3,首先测试变化选择率,实验结果如图 6 所示。

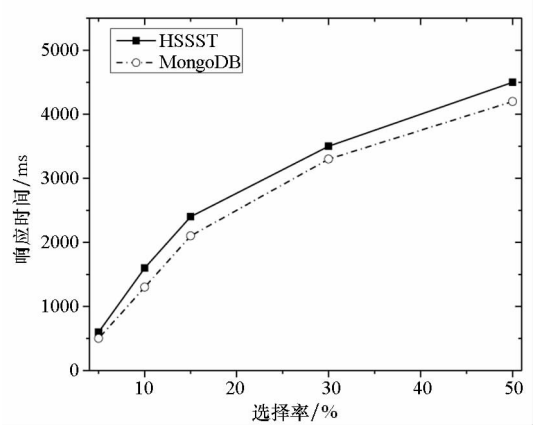

图 6 时空范围查询实验结果(变化选择率)

Fig. 6 Results of spatio-temporal range queries (change selectivity)

变化选择率从 3% 至 50%,两种方法的查询响应时间都在增长,这是因为随着查询的时空窗口与 XPath 选择率的增大,所涉及的文档数量不断增长,HBase 和 MongoDB 中所查询的行数不断增加引起查询延时增加。从趋势上分析,由图中发现,HSSST 与 MongoDB 的性能曲线趋势相似。这是因为,HSSST 利用 HBase 的 meta 结构,该结构是一种树状索引,通过 Hilbert 曲线值查找对应的区域服务器;MongoDB 是一种面向文档的 NoSQL 数据库,并且针对地理信息维度建立 Geospatial 索引,该索引也是一种树状索引,但并没有使用 Hilbert 映射方法,而是通过类似 B - 树查找的方法深入到叶结点。虽然 MongoDB 充分利用内存将查询大部分都在内存中完成,但其索引结构与 Hilbert 曲线映射相比较并没有优势,而 HSSST 通过 Hilbert 曲线映射并且将第二阶段过滤查询分散到各个区域服务器上,降低了时间开销。因此,二者在性能上比较相近。对比两种方法可见,在配置一般的计算机上,HSSST 仅使用了 3 台机器就接近了 MongoDB 的查询性能(基本上相差 500 ms),这主要是因为 HSSST 不但使用了空间信息作索引,而且还使用了时间信息作索引结构,并且时间索引占用内存较小,空间索引利用 meta 结构分布到各个机器上。

然后测试节点数量变化对查询性能的影响,变化 HSSST 所用的节点数量从 3 至 11,MongoDB 机器 1 台不变。图 7 显示了实验结果。可见节点数量增加时,HSSST 查询时间在不断降低,这是由于机器数量增多,查询被分散,吞吐量增加,提高了性能。进一步,由图可见,HSSST 机器数量为 5 时,性能就超过了 MongoDB,可见 HSSST 中时空索引和半结构化查询算法十分有效。

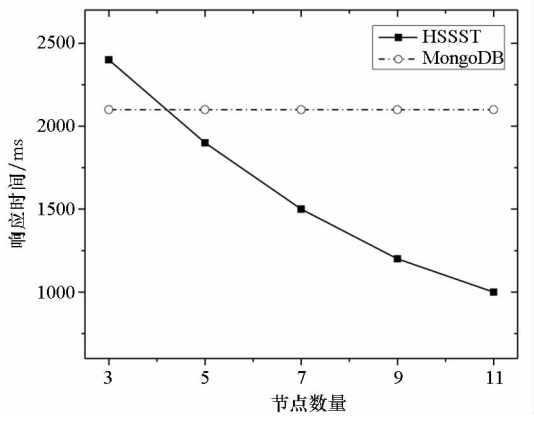

图 7 时空范围查询实验结果(变化节点数量)

Fig. 7 Results of spatio-temporal range queries (change node numbers)

5.2 *k*NN 查询实验

对 *k*NN 查询算法进行测试,类似地,也分为 k 值变化测试和节点数量变化测试,k 值默认为 10,节点数量默认值为 3。图 8 展示了 k 值变化的实验结果。

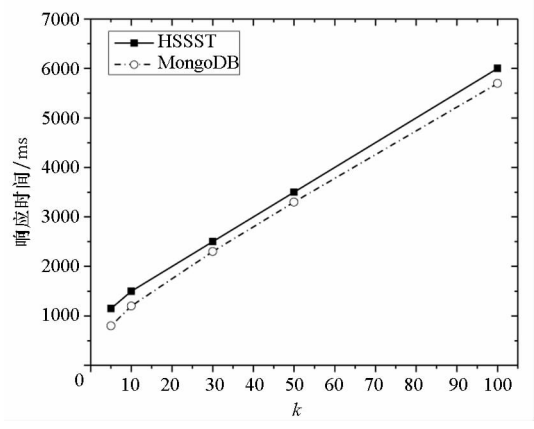

图 8 *k*NN 查询算法实验结果(变化 k 值)

Fig. 8 Results of *k*NN queries (change k)

由图 8 可见,k 值不断增加引起访问的空间

范围也不断增多,两种方法的查询性能都在降低。趋势上,二者的性能接近,原因同上一小节讲述的。

图 9 显示了 kNN 查询中节点数量变化的实验结果。性能和原因与时空范围查询类似,当节点数量增大时,HSSST 的查询能力超过了 MongoDB 的。

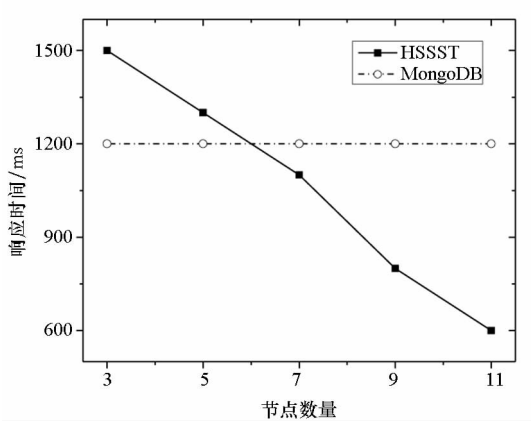

图 9　kNN 查询算法实验结果(变化节点数量)

Fig. 9　Results of kNN queries (change node numbers)

6　结论

随着各种探测技术的不断发展与各类时空数据的深入应用,对时空数据的有效管理与利用将会越来越受到关注。针对半结构化的时空数据存储与查询这一目前尚处起步阶段的问题进行研究,面向时空范围和 kNN 查询,提出了问题的形式化描述,设计了存储与索引结构模型 HSSST,并给出了范围和 kNN 查询的算法。实验在真实数据集上进行,与需要配置较高硬件设备的 MongoDB 进行比较,结果表明所设计方法的查询性能在配置一般的集群上就能接近 MongoDB 的,当节点数量增加时,其性能还会超过 MongoDB。

进一步的工作将专注于将结构化与半结构化的时空数据统一建模管理,并进一步提升查询性能。

参考文献(References)

[1] MongoDB. The MongoDB 3.2 manual[EB/OL]. [2016 - 05 - 10]. https://docs.mongodb.com/manual/.

[2] Apache Software Foundation. HBase[EB/OL]. [2016 - 03 - 12]. http://hbase.apache.org.

[3] Nishimura S, Das S, Agrawal D, et al. MD-HBase: a scalable multi-dimensional data infrastructure for location aware services [C]//Proceedings of the 12th IEEE International Conference on Mobile Data Management (MDM), 2011, 1: 7 - 16.

[4] Hsu Y T, Pan Y C, Wei L Y, et al. Key formulation schemes for spatial index in cloud data managements[C]//Proceedings of the 13th International Conference on Mobile Data Management (MDM), 2012: 21 - 26.

[5] Zhou X, Zhang X, Wang Y H, et al. Efficient distributed multi-dimensional index for big data management [C]// Proceedings of 14th International Conference, WAIM, 2013: 130 - 141.

[6] Han D, Stroulia E. HGrid: a data model for large geospatial data sets in HBase [C]//Proceedings of the Sixth International Conference on Cloud Computing (CLOUD), 2013: 910 - 917.

[7] Zhang N Y, Zheng G Z, Chen H J, et al. HBaseSpatial: a scalable spatial data storage based on HBase [C]// Proceedings of the 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2014: 644 - 651.

[8] Chen X Y, Zhang C, Shi Z L, et al. Spatio-temporal keywords queries in HBase[J]. Big Data and Information Analytics (BDIA), 2015, 1(1): 81 - 91.

[9] Chen X Y, Zhang C, Ge B, et al. Spatio-temporal queries in HBase [C]//Proceedings of the IEEE International Conference on Big Data (Big Data), 2015: 1929 - 1937.

[10] Du N B, Zhan J F, Zhao M, et al. Spatio-temporal data index model of moving objects on fixed networks using HBase[C]// Proceedings of the International Conference on Computational Intelligence & Communication Technology (CICT), 2015: 247 - 251.

[11] Vahdati S, Karim F, Huang J Y, et al. Mapping large scale research metadata to linked data: a performance comparison of HBase, CSV and XML[M]. USA: Metadata and Semantics Research (MSR), Springer International Publishing, 2015: 261 - 273.

[12] Moon B, Jagadish H V, Faloutsos C, et al. Analysis of the clustering properties of the Hilbert space-filling curve[J]. Knowledge and Data Engineering (KDE), 2001, 13(01): 124 - 141.