# 高效的空间大数据分布式拓扑分析算法

## Efficient Distributed Big Spatial Data Topology Analysis Algorithm

**Abstract:** Rapidly growing spatial data and the increasing demand for application have put forward higher requirements for spatial topology analysis algorithms, such as high concurrent real-time request processing and massive data topology analysis. The existing stand-alone or multi-cores space topology analysis algorithm to deal with spatial data is inefficient. We propose an efficient big spatial data distributed topology analysis algorithm. The space-filling curve is used to divide the complex spatial objects into subspace objects. Each sub-space-object has a unique number, CellId. A pre-judgment algorithm is proposed that uses the CellId' spatial information to filter the redundant data and reduce the data in topological analysis. Experiments show that this algorithm can efficiently process the topological analysis of massive spatial data. It is superior to existing stand-alone and multi-core algorithms in most cases, and has a linear acceleration ratio.
**Key words:** spatial database; topology analysis; distributed computing; data division; space fill curve

摘　要: 迅猛增长的空间数据和日趋旺盛的应用需求，对空间拓扑分析算法提出了更高的要求，如高并发请求实时处理、海量数据拓扑分析。现有空间拓扑分析算法针对单机或多核设计，处理空间大数据效率不高。本文提出了一种高效的空间大数据分布式拓扑分析算法。利用空间填充曲线将复杂的空间对象划分为子空间对象，每个子空间对象拥有唯一网格编号；提出一种预先判断机制，利用网格编号包含的局部空间信息过滤冗余数据，减少拓扑分析的数据量。实验表明，该算法可以高效的处理海量空间数据拓扑分析，在绝大多数情况下优于现有单机和多核算法，且具有线性加速比。
关键词:　空间数据库；拓扑分析；分布式计算；数据划分；空间填充曲线

## 1　引言

随着地理信息系统的广泛应用和物联网技术的蓬勃发展，越来越多的软件提供与空间位置信息相关的服务。如：空管系统通过判断航迹相交情况调节航班；Baidu Apollo 无人驾驶项目对附近物体的位置判断包含大量的相交、包含、平行操作。上述判断轨迹相交、物体包含等运算即是针对空间数据的拓扑分析操作。这些应用对空间拓扑分析提出了巨大的挑战：(1)进行拓扑分析运算的空间数据规模大，例如车辆轨迹数据已经达到 TB 甚至 PB 级别；(2)空间拓扑分析运算性能要求高，例如自动驾驶需要毫秒级的碰撞检测速度。现有绝大多数空间拓扑分析算法都是单机算法[1]，分析效率不高。尽管也有利用多核处理器并行计算[2][3]以提高性能的研究，但无法处理海量空间数据。因此，对空间大数据进行并行拓扑分析是目前急需解决的问题。

近年来，大规模分布式系统（例如 Hadoop[4][5]、Spark[6]等）被大量应用于大数据的存储、查询、分析中。工业界和学术界对分布式系统中的传统关系操作和查询，如：join[7][8]、Top-k query[9][10]、knn-join[11][12]进行了深入研究。然而，空间数据与关系数据相比，结构更加复杂，除数值关系之外，数据之间存在空间拓扑关系，且空间拓扑关系难以直接利用传统关系操作进行计算得到。利用分布式集群进行空间拓扑并行分析的研究目前仍处于起步阶段，现有空间大数据查询分析系统如 Hadoop-GIS[13], SpatialHadoop[14]，仅涉及点与点之间的拓扑关系和查询操作，包括最近邻和 KNN 查询[12][15]。分布式领域尚无对复杂空间对象和复杂拓扑关系的研究。

针对空间大数据进行拓扑分析的挑战有两方面：随着测量精度的提高，单个空间对象的数据量随之增大，需要切割单条记录以提高并行度，并保证切割前后拓扑关系运算结果的一致性和正确性；历史数据的积累和新型采集设备的出现，导致空间数据总量庞大且不断增长，需要高效的存储读取数据和减少拓扑分析中的无用数据。

针对上述挑战，本文提出了一种高效的、基于空间填充曲线的分布式拓扑分析算法（**D**istributed **S**patial-Filling-Curve-Based **T**opology **A**nalysis algorithm, DSTA）。该算法利用空间填充曲线将空间对象划分为多个子空间对象，分析过程采用预先判断策略过滤冗余子对象，并保证分布式拓扑分析得到正确的拓扑分析结果；DSTA 算法利用不同分布式存储系统，分层存储数据以实现高效存取。实现了一个原型系统进行实验验证。实验结果表明：与单机串行[1]或并行算法[2][3]相比，本文提出的 DSTA 算法性能更优，具有线性加速比。

本文主要贡献如下：设计了空间对象划分算法（**S**patial **O**bject **P**artitioning algorithm，SOP），利用空间填充曲线和网格划分空间对象；提出了预先判断策略，减少拓扑分析中的冗余数据；设计 DSTA 算法，利用 SOP 划分数据，并在拓扑分析中采用预先判断策略减少冗余。

## 2 相关工作

### 2.1 空间拓扑分析

空间拓扑分析是从原始数据中分析得到空间对象之间的拓扑关系，包括点、线段、区域的相交、平行、包含、距离等。其中判定线段相交操作是空间拓扑分析的基础操作，是分析其他拓扑关系的基石（如求区域交集、线段距离均需得到空间对象相交情况）。现有空间拓扑分析算法主要采用了两种技术：平面扫描和归并遍历，分别适用于不同种类的空间拓扑分析操作。

平面扫描思想[1][2][3]的核心是降维。以二维空间为例，应用平面扫描，将二维空间问题转化为一维问题。通过一条垂直的扫描线(sweep line)，按照一定顺序（通常从左到右）扫过平面上的二维对象，保存扫描过程中得到的信息，利用该信息降维以简化问题规模。常用于处理诸如相交、包含之类的空间拓扑关系。

归并遍历通过遍历所有的空间对象，逐个判断，得到空间对象之间的拓扑关系。常用于解决“两个点集合并”、“判断点集 P 是否在区域集 R 的边界上”等问题，即主要用于处理含有共有元素的部分拓扑分析操作，对两个空间数据对象的组成部分归并后分析以得到结果。

面对空间数据的持续增长，为了提高空间拓扑分析的性能，文献[2][3]提出了基于多核处理器的并行平面扫描算法，但这些算法仅考虑了单机多核并行，无法对空间大数据进行拓扑分析。

### 2.2 大数据系统

近年来，大数据技术发展迅猛，涌现出诸如 Hadoop、Spark、Paxos 等优秀的分布式系统[4][5][14]，利用廉价机器组成分布式集群处理大数据，并且具备分布式存储、计算、容错等机制。

Hadoop 是 Google 分布式系统的开源实现，它为大规模的容错数据处理提供了一个环境。Hadoop 由多个子系统组成，包括分布式文件系统 HDFS（Hadoop Distributed File System）；分布式编程模型 MapReduce；分布式 key-value 数据库 HBase。Spark 是在 MapReduce 的基础上的一种分布式内存计算框架。利用内存的快速读取性能，提高了迭代计算的性能。Spark 提供了 map，group，count 等数据操作，是较 map，reduce 更高层次的抽象，减轻了编程人员的工作。

研究人员利用上述大数据系统对 join、Top-k query、knn-join 等传统关系操作和查询的并行化进行了研究。然而，利用大数据系统进行空间拓扑并行分析的研究目前仍处于起步阶段。

## 3 空间数据划分

空间数据一般包含空间位置信息，时间信息和语义信息。更高的测量精度和更细粒度的应用需求使空间位置信息迅速增加，单条记录数据量成倍扩大，数据总量指数级增长。以香港科技大学给出的上海出租车数据集（http://www.cse.ust.hk/scrg/）为例，每条数据包含了空间位置信息（例如车辆的经纬度）、时间信息、语义信息（例如出租车编号，载客情况和速度等）。由于每秒采集一次位置数据，一辆车每天的轨迹信息包含数万个坐标点，每天收集的数据量超过 1G。积累的历史数据能够达到 TB 甚至 PB 的规模且不断增长。

在单个空间对象数据量和空间数据总量双重增长的情况下，如果分布式算法依旧采用传统分发策略，将单个空间对象数据作为最小数据单元，分发到分布式集群中处理，难以充分利用分布式集群的能力。提出空间对象划分算法（Spatial Object Partitioning algorithm，SOP），首先利用空间填充曲线，将空间划分为均匀的网格（Cell），计算出网格编号（CellId）。然后，将组成空间对象的点按照网格切割，同一 Cell 中的连续点被划为一个独立的子空间对象，每个子空间对象有一个唯一的序号，原空间对象存储子空间对象的序号以还原完整空间信息。最终复杂庞大的单个空间对象被划分为多个简单的子空间对象。数据划分后单条记录的数据量减小，有利于提高并行度、平衡机器负载。同时，划分后的每个子空间对象包含在一个 Cell 中，利用 CellId 的性质，提出一种预先判断机制，利用 CellId 包含的局部空间信息过滤冗余数据，减少拓扑分析的数据量。

### 3.1 划分空间对象

空间对象划分需要保证划分后的多条数据组合后仍可还原空间对象的信息。由空间填充曲线计算得到的 CellId 包含空间信息，每个子空间对象都对应一个 Cell，其 CellId 保留局部空间信息，原空间对象存储的多个子空间对象的序号保留整体空间信息。

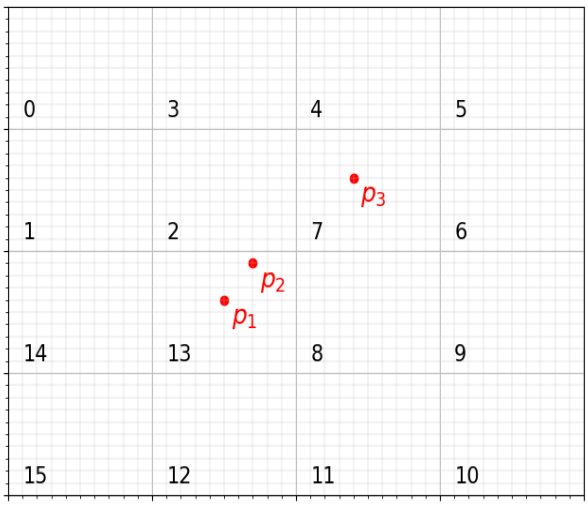
计算机利用离散的点阵描述空间对象连续的位置信息（如轨迹、轮廓）。空间填充曲线具有一维映射多维的性质，可以将高维的空间点映射到一维中[17][18]。高维空间点通过空间填充曲线 f(x,y) 转换后得到的值，记作 id，高维空间相邻点的 id 仍然相近。如图 1，二维平面中的点 $P_1$（1.5，2.3）、$P_2$（1.6，2.1）、$P_3$（2.4，1.7），按 3 阶希尔伯特曲线

转换后的整数部分为 13、13、7，符合高维相邻映射后依然相近。

调整空间填充曲线的阶数，可以得到不同大小的网格（Cell），并计算网格编号（CellId）。将 CellId 和 id 转换为二进制，则一个 Cell 内所有点的 id 具有相同的前缀。图 1 中 $P_1$、$P_2$ 公共子前缀为 0000 1101 B，也就是此网格 CellId 13 的二进制形式。实际应用中，点坐标一般采用 GPS 坐标，需要先将球面上的 GPS 坐标转换为平面坐标，然后根据空间填充曲线 f(x, y)计算得到 id。

Fig.1 CellId

**图 1**　网格编号




CellId 保留了一个 Cell 中子空间对象的局部空间信息；空间对象划分后存储的子空间对象的序号可还原整体的空间信息。图 2 中的空间对象是一条多个点组成的折线，划分的具体流程如下：从起始点 $P_1$ 开始，遍历组成空间对象的所有点，按照 Cell 切割，同一个 Cell 内的点组成一个子空间对象。遍历至终点 $P_{31}$ 结束。划分后得到子空间对象的序号：

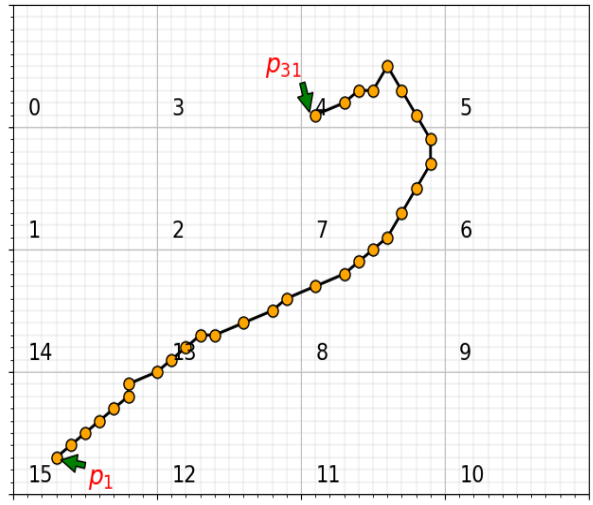
$[obj_{15}, obj_{13}, obj_7, obj_8, obj_4]$

每个子空间对象如下：

$obj_{15} = [P_1, P_2 \dots P_5, P_7]$,
$obj_{13} = [P_8, P_9 \dots P_{14}, P_{15}]$,
$obj_7 = [P_{16}, P_{17} \dots P_{18}, P_{19}]$,
$obj_8 = [P_{20}, P_{21} \dots P_{23}, P_{24}]$,
$obj_4 = [P_{25}, P_{26} \dots P_{30}, P_{31}]]$ 。

Fig.2 object division

**图 2**　空间对象划分




### 3.2　空间对象划分算法

空间对象划分算法（Spatial Object Partitioning algorithm，SOP）的伪代码如算法 1 所示。算法的输入是一个空间对象，输出是该空间对象经过划分后的一组子空间对象。子函数 getCellId() 的作用是：对于给定的空间点 point，返回经纬度转换后的 CellId。函数 SOP() 返回给定空间对象被划分得到的一组子空间对象。该算法首先遍历组成空间对象的点阵，调用 getCellId 函数得到 CellId，以此判断相邻两点是否处在同一个 Cell 内，并将每一个 CellId 加入结果集中，遍历结束后返回该集合，结束程序。getCellId() 的时间复杂度为 O(1)，SOP 中包含一个 for 循环，故总时间复杂度是 O(n)，n 代表点阵中点数量。

**算法 1：SOP**
输入：空间对象 Obj
输出：划分后的子空间对象向量

```
1.  function getCellId(point)
2.      id = point.CellId;
3.      return id;
4.  end function
5.  .
```

```
6.  function SOP (obj)
7.      res = NULL;
8.      Points = obj.getPoints();
9.      m = points.length;
10.     for i = 0 -> m-1 do
11.         id1 = getCellId(point[i]);
12.         id2 = getCellId(point[i+1]);
13.         if id1 == id2 then
14.             res += id1;
15.         end if
16.     end for
17.     return res;
18. end function
```

分布式拓扑分析需要处理大量空间对象，故在 SOP 算法基础之上，提出了分布式数据划分算法(Distributed Spatial Object Partitioning algorithm，DSOP)，将空间对象划分到集群中的不同机器上，对每个空间对象执行 SOP 算法，汇总中间数据，得到最终结果。

**算法 2：DSOP**

输入：空间对象 objs,机器数量 q

输出：子空间对象向量

```
1.  function map(k, v)
2.      res = SOP(v);
3.      return <k, res>;
4.  end function
5.
6.  function DSOP(objs)
7.      objs -> nodes;
8.      res = null;
9.      for i = 0 -> q-1 do
10.         res += map();
11.     end for
12.     return res;
13. end function
```

### 3.3 预先判断策略

DSOP 划分后的数据分发到分布式集群进行拓扑分析。由于每个子空间对象均包含在一个 Cell 中，其 CellId 的数据量远远小于子空间对象的数据量。在拓扑分析前，利用 CellId 之间的关系预先判断，可减少分布式拓扑分析的计算量。

以拓扑分析基础操作相交判定为例，若两个空间对象相交，由定理 1，只需判断 CellId 相同的子空间对象是否存在交点即可。

**定理 1**：空间对象 $obj_1$，$obj_2$ 的交点仅存在于 CellId 相同的子空间对象中。

**证明**：$obj_1$, $obj_2$ 划分后的两个子空间对象为：

$$[obj_{11} \quad obj_{12} \quad \dots \quad obj_{1n}]$$
$$[obj_{21} \quad obj_{22} \quad \dots \quad obj_{2m}]$$

假设存在交点 P，对于 $obj_1$，P 属于子空间对象 $obj_{1x}$；对于 $obj_2$，P 属于 $obj_{2y}$，且 $obj_{1x}$ 和 $obj_{2y}$ 的 CellId 不同。

由于每个 Cell 中的点经空间填充曲线转换后得到的二进制 id 前缀为二进制 CellId。

同一个点经过空间填充曲线转换后 id 相同，故假设不成立，定理 1 得证。

利用预先判断策略，在判定 $obj_1$，$obj_2$ 是否相交时，只需要对 CellId 相同的数据拓扑分析，较传统算法显著减少拓扑分析的数据量，提高算法效率。对于不同的拓扑关系，需要设计不同的预先判断条件。

### 3.4 高效数据结构

预先判断策略利用 CellId 过滤减少拓扑分析过程的无用数据。以相交判定为例，算法只需要分析 CellId 相同的子数据。但是因为分布式拓扑分析的数据量大，而且划分后的子数据被频繁使用，所以划分后数据的查询效率至关重要。针对 CellId 作为查询条件的特点，设计了一种键值对的数据结构表示划分后的数据，实现高效查询。

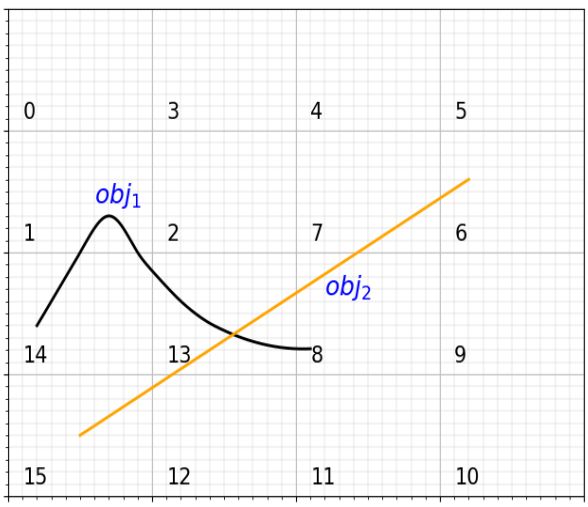
键（key）由 CellId、原空间对象的序号、该条数据在 Cell 中的编号三者组成。CellId 的作用是提

高查询效率。以键值对方式存储数据的存储系统，会按照 key 组织存储，key 相近的物理地址也相邻，这样在磁盘读取时，会节省大量 IO 时间。原空间对象的序号和该条数据在 Cell 中的编号的作用是保证 key 的唯一性。存在两种情况导致不同数据的 CellId 相同：不同空间对象划分后，可能存在多个子对象在同一个 Cell 中；同一个空间对象经过划分后，在一个 Cell 中可能存在不相连的数个子对象。如图 3，两个空间对象 $obj_1$、$obj_2$。$obj_1$ 在 Cell 14 中存在两段不相连的子空间对象；$obj_1$ 和 $obj_2$ 在 Cell 13 中均有子空间对象。

值（value）包含空间信息，语义信息。空间信息如组成该数据的点；语义信息如航班号，序号等。划分后若语义信息失效，可以不保存在划分数据中，仅在原空间对象数据中保存即可。

Fig.3 Single space object division

**图 3** 单个空间对象划分




划分后的子数据以该结构存储在基于键值对的分布式存储系统中，对于指定 CellId 的查询时间复杂度为 O(1)。

## 4 高效的分布式拓扑分析算法

现有研究难以实现面向大数据的高效拓扑分析。基于单核/多核的空间拓扑分析算法不能解决空间大数据拓扑分析需求，提出一种高效的基于空间填充曲线的分布式拓扑分析算法（Distributed Spatial-Filling-Curve-Based Topology Analysis algorithm, DSTA），首先利用 DSOP 算法划分原始数据；然后将划分后数据以键值对结构保存在分布式存储系统中；最后对划分后的子数据进行分布式计算，得到空间拓扑关系。划分、存储过程是高效拓扑分析的基础；拓扑分析过程采用预先判断策略减少冗余计算。

DSTA 算法中划分、存储过程易于分布式实现，本节不再赘述。而由于空间拓扑关系种类繁多，拓扑分析并行化较为复杂。本节首先阐述利用划分后数据并行分析空间拓扑关系的可行性；然后不失一般性，以相交判断为例介绍 DSTA 算法具体流程。

### 4.1 基于数据划分的并行算法

拓扑关系有很多种，如相交、相邻、包含、平行。传统的串行拓扑分析算法通过分析完整的空间对象得到结果。而 DSTA 算法分析划分后的子空间对象，且在分布式拓扑分析过程中，集群中的每个节点处理部分数据，通过汇总每台机器分析后的中间结果数据得到最终结果，首先证明基于前述数据划分策略进行并行拓扑分析的正确性。

**定义 1**：拓扑分析函数 $\varphi(obj_1 \quad obj_2)$。该函数作用是分析空间对象 $obj_1$，$obj_2$ 之间的空间拓扑关系，对于相交判定返回是否，对于求交点返回交点值。

由预先判断策略中的定理 1 可得

**推论 1**：拓扑局部性。存在拓扑关系$\varphi$，可通过分析空间对象的局部信息得到。

即拓扑关系 $\varphi$，空间对象 $obj_1$，$obj_2$

$$\exists\ obj_1' \in obj_1,\ obj_2' \in obj_2$$

$$\Rightarrow \varphi(obj_1' \quad obj_2') = \varphi(obj_1 \quad obj_2)$$

**定义 2：**空间划分独立性（Space division independence，SDI）。这种性质表示空间对象本身或空间对象间的空间拓扑关系能够利用数据划分后的子空间对象得到，即该拓扑关系可由局部信息分析得到。具体描述如下：

空间拓扑关系：$\varphi$

拓扑分析函数：$\varphi(obj_1 \quad obj_2)$

转换函数 $\omega$。由于空间对象之间的拓扑关系可能是多个局部上拓扑关系的组合（如空间对象存在多个交点），转换函数用于汇集局部数据上的分析结果得到最终结果。

**定理 2：**若空间拓扑关系 $\varphi$ 存在转换函数 $\omega$，则可以利用空间对象划分后的子空间对象得到拓扑关系。

**证明：**空间对象 $Obj_1$，$Obj_2$ 划分得到子空间对象矩阵，这里简化证明过程，空间对象仅划分为两个子空间对象：

$$Obj_1 = [obj_{11}, obj_{12}] \tag{1}$$

$$Obj_2 = [obj_{21}, obj_{22}] \tag{2}$$

将(1)(2)代入$\varphi(obj_1, obj_2)$，得到(3)：

$$\varphi([obj_{11} \quad obj_{12}], [obj_{21} \quad obj_{22}]) \tag{3}$$

由定义 2，拓扑分析所有局部信息的组合，转换后得到最终结果，故得到式(4)：

$$= \omega\{\varphi(obj_{11}, obj_{21}), \varphi(obj_{11}, obj_{22}), \varphi(obj_{12}, obj_{21}), \varphi(obj_{12}, obj_{22})\} \tag{4}$$

当转换函数为加法“+”时，上式即为：

$$\varphi(obj_1, obj_2)$$

$$= \varphi([obj_{11} \quad obj_{12}], [obj_{21} \quad obj_{22}])$$

$$= \varphi(obj_{11}, obj_{21}) + \varphi(obj_{12}, obj_{22}) + \varphi(obj_{12}, obj_{21}) + \varphi(obj_{12}, obj_{22}) \tag{5}$$

将(6)式带入(1) (2)，即可得到一般情况：

$$[obj_1 \quad obj_2 \quad \dots \quad obj_n] \rightarrow \begin{bmatrix} obj_{11} & obj_{12} & \dots & obj_{1m} \\ obj_{21} & obj_{22} & \dots & obj_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ obj_{n1} & obj_{n1} & \dots & obj_{nm} \end{bmatrix} \tag{6}$$

很多常见的空间拓扑关系满足空间划分独立性，但是转换函数各不相同。如，拓扑关系“相交”具有空间划分独立性，并且由定理 1 可知其转换函数是 $\cup$，故推论 2 成立：

**推论 2：** $\varphi(obj_1 \quad obj_2)$

$$= \varphi([obj_{11} \quad obj_{12}], [obj_{21} \quad obj_{22}])$$

$$= \varphi(obj_{11}, obj_{21}) \cup \varphi(obj_{11}, obj_{22}) \cup \varphi(obj_{12}, obj_{21}) \cup \varphi(obj_{12}, obj_{22})$$

空间拓扑关系 $\varphi$ 具有空间划分独立性，是拓扑分析函数 $\varphi(obj_1 \quad obj_2)$ 实现并行的关键；而转换函数的复杂程度影响并行的效率。

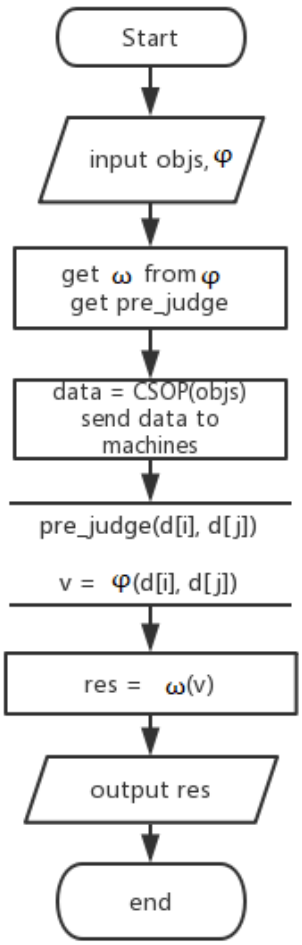
### 4.2 分布式拓扑分析算法 DSTA

基于空间填充曲线的分布式拓扑分析算法（Distributed Spatial-Filling-Curve-Based Topology Analysis algorithm, DSTA）用于分析空间对象之间的拓扑关系，如返回与给定空间对象相交的所有空间对象、查询空间对象的所有相邻空间对象等。算法流程如下：(1)首先根据拓扑关系找到对应的拓扑

分析函数和转换函数；(2)将 DSOP 算法划分得到的数据，按照 CellId 字典序发送到分布式集群上；(3)在分布式集群上运行拓扑分析函数，并且利用预先判断算法减少计算量；(4)汇总中间结果，利用转换函数得到最终结果。算法框图如图 5 所示：

Fig.5 algorithm block diagram

**图 5** 算法框图




**Step1:** 利用 SOP/DSOP 算法划分空间对象。

**Step2:** 得到空间拓扑关系 $\varphi$ 的转换函数 $\omega$；根据$\varphi$ 与 CellId 的关系，设计预先判断条件。

**Step3:** 根据预先判断策略，分布式集群中的机器读取对应数据。

**Step4:** 拓扑分析。

**Step5:** 汇总每台机器的中间数据，利用 $\omega$ 处理得到最终结果。

### 4.3 分布式相交判断算法

利用本文提出的分布式拓扑分析框架，设计实现了一种分布式相交判断算法（**D**istributed **I**nter-section **J**udgment algorithm，DIJ），用来判断空间对象是否相交。其它分布式拓扑分析运算算法同理可得。

给定空间对象 Obj，空间对象向量：

$\mathrm{T} = [t_1 \quad t_2 \quad \cdots \quad t_n]$，

拓扑分析函数：

$\varphi(\mathrm{Obj1}, [t1 \quad t2 \quad \cdots \quad tn])$，

该函数返回空间对象向量 T 中与 Obj 相交的所有空间对象的集合。因为拓扑关系“相交”满足空间划分独立性，由推论 1 可知等式成立：

$$\varphi(\mathrm{Obj}, [t_1 \quad t_2 \quad \cdots \quad t_n])$$

$$= \varphi\left( [obj_1 \quad \cdots \quad obj_p], \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1m} \\ t_{21} & t_{22} & \cdots & t_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ t_{n1} & t_{n2} & \cdots & t_{nm} \end{bmatrix} \right)$$

$$= \bigcup_{i=j=k=1}^{i<p, j<m, k<n} \varphi(obj_i, t_{jk})$$

数据划分之后，$obj_i$，$t_{jk}$ 可以保存在分布式存储系统中，使得每个 $\varphi(obj_i, t_{jk})$ 相互独立，互不干扰。且对于每次 $\varphi(obj_i, t_{jk})$过程，还可以利用 CellId 预先判断：当且仅当$obj_i$和$t_{jk}$的 CellId 相同时，才有可能存在交点，需要计算。据此，伪代码如算法 4 下：

**算法 4：DIJ**

输入：空间对象 Obj，空间对象向量 T

输出：OV 中与 Obj 相交的空间对象集合

```
1.  function getCellId(point)
2.      Id = point.CellId;
3.      return Id;
4.  end function
5.
6.  function intersectPrejudge (obj1,obj2)
7.      id1 = getCellId(obj1);
8.      id2 = getCellId(obj2);
9.      if id1 == id2 then
10.         return true;
11.     else
12.         return false;
13.     end if
14. end function
15.
16. function isCross(Obj, T)
17.     res = NULL;
18.     m = points.length;
19.     for i = 0 -> m do
20.         test = getCellId(OV[i]);
21.         if intersectPrejudge (obj,test) then
22.             res += test;
23.         end if
24.     end for
25.     return res;
26. end function
```

## 5 实验

### 5.1 实验环境

空间拓扑分析包含相交判定、求交集、判断包含关系等，其中相交判定是分析其他拓扑关系的支撑操作。故用相交判定的性能评估空间拓扑分析算法的优劣。设计了多组实验，实验一比较不同情况下，DSTA 算法与文献[3]中的并行平面扫描算法 MPPSW 的性能；实验二通过调整分布式集群的机器数和数据量，测试本文提出的 DSTA 算法是否具有线性加速比；实验三验证 DSTA 算法并发请求性能。

**Table1** Software Environment

**表 1** 软件环境

| item | version |
|---|---|
| Java | 1.8 |
| Scala | 2.11.11 |
| Hadoop | 2.8.3 |
| Spark | 2.1 |

**Table2** Hardware Environment

**表 2** 硬件环境

| item | version |
|---|---|
| Server | HP R710 |
| CPU | Intel(R) Xeon(R) E5620 2.40GHz |
| Memery | 16GB |
| Hard Disk | 1TB |

数据集采用 MPPSW 中的随机数据生成方法，生成不同大小的数据集，数据量从 0.1GB 到 100GB，单条空间数据包含 10000-20000 个点。实验软硬件如表 1 表 2 所示。

### 5.2 实验结果

#### 5.2.1 实验一

首先比较 MPPSW 算法与 DSTA 算法在不同数据量情况下的性能。横坐标是数据集的数据量，纵坐标是运行时间。

图 6 显示的是在数据量为 0.1G、0.2G、0.4G、0.6G、0.8G、1.0G 的数据集上，两种算法的运行时间对比。该情况数据量较小，对于 DSTA 算法，集群间通信的时间占比高，当数据量为 0.1G 时，MPPSW 算法更优；随着数据量逐渐增大，DSTA 算法优于 MPPSW 算法。

Fig.6 experimental result 1

**图 6** 实验 1

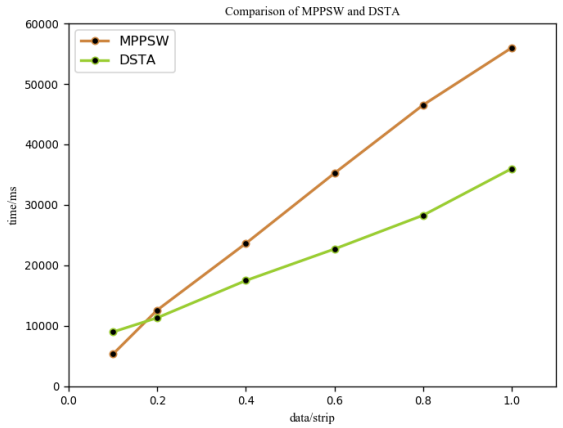
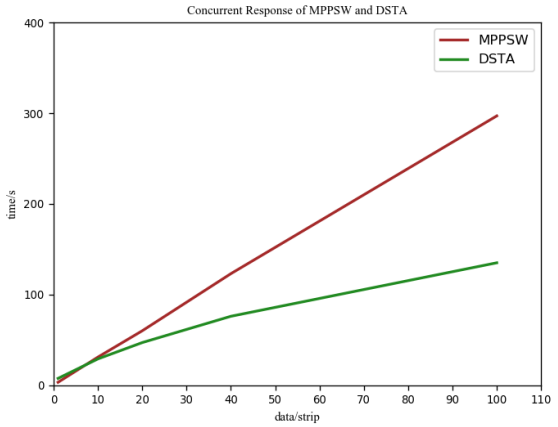



图 7 中系统数据 1G，横坐标表示用户请求数据条数，纵坐标表示运行时间。实验结果表明，当用户并发请求大于 20 条时，DSTA 算法对并发请求处理的性能优于 MPPSW；请求数量增加，效果更加显著。

Fig.7 experimental result 2

**图 7** 实验结果 2

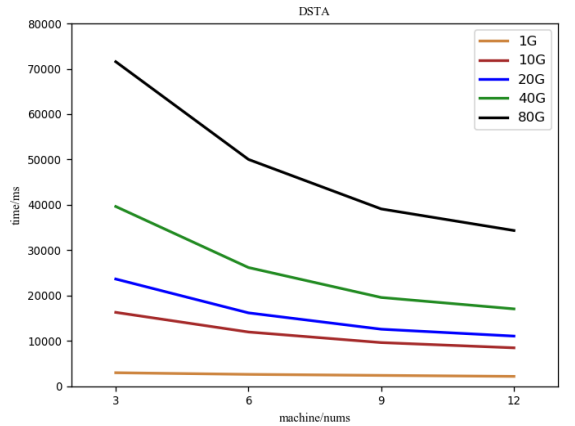



### 5.2.2 实验二

该实验验证 DSTA 算法是否具有线性加速比。分别在数据总量 1G、10G、20G、40G、80G 的情况下，利用 3、6、9、12 台服务器组成的分布式集群进行测试。实验结果如图 8 所示。当分布式集群节点数目增大时，算法具有线性加速比。

Fig.8 linear acceleration ratio

**图 8** 线性加速比

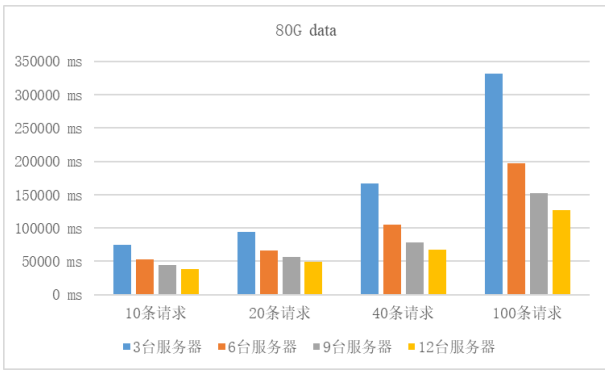



### 5.2.3 实验三

该实验测试 DSTA 算法并发性能，有三个参数：系统数据量、用户请求数量、分布式集群机器数量。每次实验固定系统数据的数据量，分别为 10G、20G、80G；对于每种用户请求规模，均在 3、6、9、12 台服务器的分布式集群进行测试。实验结果如图 9 所示，在系统数据量和用户请求数量不变时，增加集群机器数量可有效减少运行时间，提高并发请求处理性能。

Fig.9 Concurrent Processing Performance

**图 9** 并发处理性能




## 6 结束语

针对现有单机或多核空间拓扑分析算法处理空间大数据效率不高的问题，本文设计了一种空间对象划分算法，利用空间填充曲线和网格划分空间对象；提出了预先判断策略，减少拓扑分析中的冗余数据；设计了基于空间填充曲线的分布式拓扑分析算法，通过数据划分提高并行度，利用键值对数据结构实现快速读取，实验结果表明本文提出的算法性能更优。

**References**:

[1] Nievergelt J, Preparata F P. Plane-sweep algorithms for intersecting geometric figures[J]. Communications of the Acm, 1982, 25(10):739-747.

[2] Mckenney M, Mcguire T. A parallel plane sweep algorithm for multi-core systems[C]// ACM Sigspatial International Conference on Advances in Geographic Information Systems. ACM, 2009:392-395.

[3] Mckenney M, Frye R, Dellamano M, et al. Multi-core parallelism for plane sweep algorithms as a foundation for GIS operations[J]. Geoinformatica, 2017, 21(1):1-24.

[4] White T, Cutting D. Hadoop : the definitive guide[J]. O'reilly Media Inc Gravenstein Highway North, 2012, 215(11):1 - 4.

[5] George L. HBase - The Definitive Guide: Random Access to Your Planet-Size Data.[M]. DBLP, 2011.

[6] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets[C]// Usenix Conference on Hot Topics in Cloud Computing. USENIX Association, 2010:10-10.

[7] Zhang S, Han J, Liu Z, et al. SJMR: Parallelizing spatial join with MapReduce on clusters[J]. 2009:1-8.

[8] Baraglia R, Gianmarco D F M, Lucchese C. Document Similarity Self-Join with MapReduce.[C]// IEEE, International Conference on Data Mining. IEEE, 2011:731-736.

[9] Rohloff K, Schantz R E. Clause-iteration with MapReduce to scalably query datagraphs in the SHARD graph-store[C]// International Workshop on Data-Intensive Distributed Computing. ACM, 2011:35-44.

[10] Nodarakis N, Sioutas S, Tsoumakos D, et al. Rapid AkNN Query Processing for Fast Classification of Multidimensional Data in the Cloud[J]. Eprint Arxiv, 2014.

[11] Lin X. A Top-k query algorithm for big data based on MapReduce[C]// IEEE International Conference on Software Engineering and Service Science. IEEE, 2015:982-985.

[12] Wei L, Shen Y, Su C, et al. Efficient processing of k nearest neighbor joins using MapReduce[J]. Proceedings of the Vldb Endowment, 2012, 5(10):1016-1027.

[13] Eldawy A, Mokbel M F. A demonstration of SpatialHadoop: an efficient mapreduce framework for spatial data[M]. VLDB Endowment, 2013.

[14] Aji A, Wang F, Vo H, et al. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce[J]. Proceedings Vldb Endowment, 2013, 6(11):1009-1020.

[15] Doulkeridis C, Vlachou A, Mpestas D, et al. Parallel and Distributed Processing of Spatial Preference Queries using Keywords[J]. 2017.

[16] Prisco, De R, Lampson, et al. Revisiting the Paxos Algorithm[J]. Theoretical Computer Science, 1997, 243(1):35-91.

[17] Dai H K, Su H C. Approximation and Analytical Studies of Inter-clustering Performances of Space-Filling Curves.[C]// Discrete Random Walks, Drw'03, Paris, France, September. DBLP, 2015:53-68.

[18] Xu P, Nguyen C, Tirthapura S. Onion Curve: A Space Filling Curve with Near-Optimal Clustering[J]. 2018.