分布式系统下民航时空轨迹数据的管理与分析

**摘要（Abstraction）：**

随着定位技术的发展和大规模位置感知应用的广泛应用，导致了不断产生大量的时空轨迹数据。特别地，由于民航运输业的快速发展，民航信息化程度不断提高，民航应用领域产生了海量的航班轨迹数据。高效地处理时空数据对于制定航班计划策略和空域和流量策略有着至关重要的作用。为了有效地支持复杂的民航空中交通工作，例如基于时空大数据下的空中交通流量管理的研究，需要一种高效、可伸缩、时空处理操作丰富、且易于使用的时空数据管理与分析应用。

尽管目前有一些时空轨迹数据管理研究的原型系统，然而这些系统仅支持有限的范围查询操作,不能有效处理大规模复杂的时空数据。本文提出了一种时空轨迹数据处理和分析应用，基于Spatialhadoop框架开发,可以高效便捷地管理大时空数据。我们的工作将支持时空轨迹数据的存储，并丰富了一些时空数据处理操作。我们提出了一个高度并行化的查询处理框架来高效地计算时空多边形范围查询，并创造性地设计了时空k-NN查询算法,提高了时空查询速度。此外，我们还设计了一种针对空域交通流量的统计策略，能够为民航流量策略的制定提供分析基础。我们用真实数据集和合成数据集进行实验评估。并与主流的时空数据处理系统ST-Hadoop进行了对比。实验显示，我们比现有的所有技术都有显著的优越性。

1. **介绍（Introduction）**

**思路：时空数据来源（规模时空应用的使用）——导致时空数据激增（体量大、结构复杂）——引发大量时空数据研究——时空数据存储与查询处理——范围查询的场景及价值——提出时空多边形范围查询——空域流量统计、航路点、航班计划**

近年来，随着定位技术的广泛应用，物联网传感器、GPS穿戴设备、智能手机、卫星等应用设备在全球范围内广泛使用，这些不同的应用来源不断产生海量的时空数据，以指数级不断增长。例如，在民用航空领域，空中交通管理每天产生约10PB大量的具有时间和空间属性的空管大数据，且数据量日益增大，类型繁多。比如，雷达每3秒一次采集航班飞越扇区的航班轨迹点ADSB数据，一天之内一个扇区内的数据集高达1TB。飞行量的快速增长给空管行业带来越来越大的挑战, 日益增长的空中交通流量必然会加剧空中交通拥堵，加重航班延误问题，正点率下降，并且由此造成旅客、航空公司、机场等各方的巨大损失。大数据的泛滥，给数据管理和处理系统带来了前所未有的性能和可伸缩性挑战。大量的时空数据超出了原有系统的存储、处理和分析的能力。

这些时空数据对于民航空域管制具有重要的作用。对航迹数据的处理能够发现好很多有用的信息。例如，航迹点ADSB数据和扇区静态数据可以用来统计一个时间段内的空域流量，对于合理规划空域,科学管理空中交通流量具有非常大的帮助。航班和扇区、航路点的数据关联，能够用来对空域与交通性能/特征的关联关系进行分析与挖掘。

然而，对海量异构数据进行高效管理并非易事。原因如下：时空数据体量大，种类繁多。在二维空间中，空间数据类型只是简单的对象结构，如单点、线和简单区域。在三维空间中，时空数据类型使用户能够描述空间对象随时间变化的动态行为。时空数据是连续生成的，可以具有巨大的规模。对这类高维数据的管理是个极大的挑战。时空数据蕴含丰富的数据关联特性，时空数据本质上具有至少三个维度的复杂性，即纬度、经度和时间信息。

民航时空数据蕴含着丰富的信息，各个属性之间又包含大量复杂的、可挖掘的关联关系，能够应用于各种空中交通管制策略应用。但时空数据关联紧密、体量大、结构复杂、多样性、价值密度低,难以被高效存储、管理和分析。单台机器无法应对海量数据的场景，而分布式查询处理框架如Spark，Hadoop等，缺乏有效的时空索引和时空分析算法，因此很难高效地处理时空数据。

目前大部分空间数据处理系统，多是基于Hadoop或者Spark进行扩展的系统。如Spatialhadoop、ST-Hadoop、HadoopGIS等。这些系统不能识别时空对象特性，缺乏时空索引，也不能支持时空范围查询算法。现有的时空数据处理平台如ST-Hadoop，虽然在查询时间方面比较高效，但建立索引的时间开销巨大。而且，由于ST-Hadoop本身支持不同粒度的时间切片，导致hdfs物理块数据急剧膨胀，建立多层索引引起的磁盘存储空间可增至2~3倍。

范围查询通常用于从空间的特定部分 提取信息。目前现有的研究很少关注时空条件下多边形范围查询的问题。这个问题很有挑战性，因为真正的多边形(如县、市、空域等)是由非常复杂的边界构成的。多边形的查询范围是一组不重叠的复杂多边形，通常以数千为数量级，而输入是一个包含数亿甚至数十亿空间点的大型数据集。当底层数据集很大时，获取满足查询条件的所有点代价十分高昂，因为满足查询的点可能有很多。对这些时空数据点执行数据分析任务或计算聚合所需的CPU成本会增加额外的开销，并且随着时空数据的增加，系统可能无法很好地扩展。因此，等待准确的分析或聚合结果会花费较长的时间。

基于以上的讨论，本篇论文介绍了NHhadoop，一个面向民航空中交通的时空数据处理和分析应用。我们的论文工作贡献分为以下几个方面：

1、我们扩展了Spatialhadoop，支持时空数据结构，基于时空索引机制，建立一套时空数据管理机制并实现了一个整体的分布式系统，通过这个平台用户可以方便地管理大规模的时空数据。

2、我们提出了一个高度并行化的查询处理框架来高效地计算时空多边形范围查询，减少范围查询时间，提高查询速度；提出一种时空knn范围查询方法，能够判断航迹和航路点周围区域的位置关系，极大了丰富了时空分析操作算法的种类，提高了时空查询的效率。

3、我们使用真实的扇区静态数据集和雷达采集的ADSB数据集进行性能实验。实验表明，它在查询效率和功能的可扩展性方面都优于其他先进的系统。

本文的其余部分组织如下。在第二节中，我们首先回顾时空数据管理系统的相关工作。第三节介绍了范围查询相关的数学定义。第四节描述了两种范围查询算法。第五节详细描述了统计空域流量的具体流程和数据构成，包括空域扇区数据以及ADSB数据。我们在第六节中给出了系统性能评价结果，并在第七节中总结了本文的研究工作。

1. **相关工作(Related Work)**

**移动对象数据库系统**MOD。MOD是一个能够存储、查询和管理移动对象数据的数据库系统。有一些原型系统：比如SECONDO、Hermes和DEDALE。这三个系统都是管理移动对象轨迹的系统。大部分将在单个节点服务器上运行。故而无法应对海量数据的场景。尽管可以优化这些MOD系统，以加速时空范围查询处理，但仍然是不可扩展的。并且，这些单一架构的系统支持的时空轨迹查询处理操作十分有限。

**可扩展的移动对象数据库。**由于上述这些MOD原型系统是不可伸缩的，因此不能支持大规模的时空数据处理，所以需要向分布式数据库方向扩展。例如，ParallelSECONDO 是使用Hadoop的SECONDO的一个版本。Hadoop被用作通信管理器，用于调度和协调工作节点之间的任务，每个工作节点运行一个常规的SECONDO实例，并包含数据的完整副本。

GeoMesa是一个用于大规模存储、查询和转换时空数据的时空数据库。它建立在Apache Accumulo之上，后者是建立在Hadoop之上的一个键值存储。GeoMesa使用geohash和时间戳来组织数据。键是作为geohash和时间值的组合生成的。

同时，针对现有的轨迹数据管理框架存在可伸缩性问题，只能支持有限的轨迹查询，京东等人的工作提出了一个整体的分布式NoSQL轨迹存储引擎，TrajMesa。TrajMesa采用了一种新的存储模式，极大地减小了存储容量。针对各种轨迹查询，设计了新的索引键模式和多种剪枝策略，使系统具有良好的查询效率和可扩展性。TrajMesa能够有效支持多种轨迹查询算法，包括:ID时态查询、空间范围查询、相似性查询和k-NN 查询。

**对Hadoop加以拓展时空操作的系统**。与MOD系统不同，有些工作在Hadoop（或者其他大数据处理框架）上扩展时空支持。

SpatialHadoop在Hadoop核心中内置空间类型和操作符，包括点和区域的基本GIS类型。并且实现了两层索引，一个全局索引作用于主节点，一个局部索引作用于每个工作节点的数据片。SpatialHadoop支持一系列空间索引技术:Grid、R-tree和R+-tree等等。在空间索引的基础上提供了一组空间数据查询处理操作。Spatialhadoop不能识别时空对象特性，缺乏时空索引，也不能支持时空范围查询算法。

ST-Hadoop是SpatialHadoop在时间维度上的扩展。它实现了一个时空类型STPoint，是(纬度、经度、时间)的三元组。ST-Hadoop可以处理大量的STpoint，并计算它们的选择性和连接谓词(例如，overlap和within\_distance)。ST-Hadoop的全局索引由一个时间切片器扩展，该切片器根据给定的时间粒度(日、周、月)将输入数据分割成多个切片。在每个切片中，构建了类似于SpatialHadoop的索引。相应地，运算符将先按时间筛选，获取相应的切片，并使用SpatialHadoop空间操作的时间扩展进行处理。ST-Hadoop构建的是多层索引：首先根据时间维度构建索引，再为每个时间点或者时间段构建一个空间索引。

ST-Hadoop，虽然在查询时间方面比较高效，但建立索引的时间开销巨大。对时间点的查询效率十分有效，时间段的查询效率低，会产生大量重复的数据记录。同时，因建立时空索引而引起的延迟，会让系统处理数据的能力大大下降。而且，由于ST-Hadoop本身支持不同粒度的时间切片，导致hdfs物理块数据急剧膨胀，建立多层索引引起的磁盘存储空间可增至2~3倍。**更重要的是，ST-Hadoop无法表达轨迹的概念，相应地，也无法支持轨迹级别的查询操作，如速度、相交等。**它仅限于处理离散时间点集。ST-Hadoop和我们的工作之间的详细比较将在第6节中讨论。

HadoopTrajectory，一种用于时空数据处理的Hadoop扩展。HadoopTrajectory主要支持轨迹数据，该扩展将时空类型和时空操作添加到Hadoop核心中。同时建立基于轨迹的索引，每个节点包含一条轨迹的片段，同一个轨迹的节点被链接在一起，保证了时间连续性；但缺点同时也存在，不同轨迹但空间邻近的片段将被存到不同的节点，即破坏了空间邻近性，不利于空间近邻查询。

**对Spark或NoSQL加以拓展时空操作的系统**。Apache Spark是一个分布式内存计算系统。与Hadoop相比，Spark的效率更高，因为它尽可能多地将数据缓存在内存中。然而，Spark本身并不支持空间/时空数据管理。GeoSpark通过提供支持空间范围查询、kNN查询和空间连接的空间RDDs (SRDDs)。然而，GeoSpark缺乏全局索引，这限制了它的性能。SpatialSpark对空间数据采用固定Grid分区、二分空间分区、平铺分区，支持空间范围查询。LocationSpark提供了一组空间查询操作符，包括空间范围查询、k-NN查询和空间连接。它使用各种空间索引，例如Grid，R树，四叉树，IR树。它还设计了一个动态内存缓存框架，将访问频率较低的数据刷新到磁盘。

文献提出了一种基于NoSQL的JD时空数据引擎——JUST，利用GeoMesa组件建立时空索引，HBase作为底层存储，可以方便地有效地管理大时空数据。该工作还设计了Z2T和XZ2T两种新的索引策略，以弥补原生的GeoMesa建立的索引不适合进行时空范围查询的不足，极大地提高了查询效率。并引入了压缩机制，通过减少磁盘IO开销，降低了存储成本，提高了查询效率。 NoSQL数据存储，如Cassandra、HBase，可以实现每秒数百万次的更新，并提供高可扩展性和随机访问的数据管理。许多工作提供了基于NoSQL的空间或时空数据访问。如XXX。大多数基于nosql的系统都很难使用。用户需要深入研究并实现自己的时空查询操作。

表1：基于Hadoop或Spark的不同DSDMSs所支持的功能比较

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| DSDMS | Architecture | Spatial Index | Spatial Query | Scalability | S or ST |
| Spatialhadoop | Hadoop | R-Tree,Grid,Quadtree,kd-tree | range query, kNN, spatial join | Yes | S |
| ST-Hadoop | Hadoop | ST-index | ST-rangequery, ST-join, kNNQ | Yes | S/ST |
| Hadoop-GIS | Hadoop | Grid | Range query, spatial join | Yes | S |
| Hadoop-Trajectory | Hadoop | Grid,kd-tree | Pass,Traj,WindowIntersect | Yes | S |
| SpatialSpark | Spark | Grid,kd-tree | Range query, spatial join | Limited | S |
| GeoSaprk | Spark | R-tree,Quadtree | range query, kNN, spatialjoin,  distance join | Limited | S |
| LocationSpark | Spark | R-tree,Grid,Quadtree,IR-tree | range query, kNN, spatialjoin,  distance join, kNNJQ | Limited | S |
| Simba | Spark | R-tree | range query, kNNQ, distance  join, kNNJQ | Limited | S |
| STARK | Spark | R-tree,Grid | range query, kNN, spatial join | Limited | S/ST |
| JUST | Spark | Z2T,XZ2T | ST-rangequery, kNN | Yes | S/ST |
| OURS | Hadoop | R-Tree,Grid,Quadtree,kd-tree | ST-rangequery, ST-join, kNNQ | Yes | S/ST |

1. **系统架构Architecture**

**Indexing & Storaging layer：**

在本节中，我们将描述我们的HDFS扩展，我们的数据存放在HDFS上，以分块的形式存储。在Spatialhadoop中提出了全局索引的思想。虽然时空数据管理框架处理的信息主要是时空对象（移动空间对象），但本质上仍然是空间数据，大部分对时空对象的空间操作分析算法均可建立在空间对象的操作算法之上。在这里，我们将在时空数据的环境下重用空间索引的全局索引和本地索引。

由于时空数据存储在HDFS的物理块上，而Hadoop只能存堆文件，本身不能提供索引。所以需要建立在HDFS的不同分区技术的时空索引。建立索引的过程分为分区、本地索引、全局索引。在分区阶段，根据块大小和数据的局部性，确定分区数量。进而确定分区的边界并对数据按照边界进行划分。根据具体的索引形式(如Grid索引、Rtree索引)，建立一个分区的本地索引，最后建立整个数据集上的全局索引。沿用Spatialhadoop内嵌在Hadoop core的几种空间索引，分别为Grid,Rtree,Quadtree,Kdtree。

**Mapreduce layer：**

我们在Mapreduce层重写了SpatialInputformat和SpatialRecordReader两个函数。Hadoop在FileInputFormat类提供了一个FileSplitter类，用于将输入拆分为n个分片，并将它们分发给所有映射器。它是运行在Hadoop集群主节点上的进程。SpatialInputFormat是Hadoop FileInputFormat的扩展，它可以读取时空数据点的文件格式。Hadoop并不能按行读取时空数据格式。所以我们提供的函数能够弥补这一点。SpatialInputFormat还可以读取在存储层建立的全局索引，基于全局索引对文件进行分片。

SpatialRecordReader是一个运行在每个映射器上的进程。同样地，SpatialRecordReader扩展了Hadoop 中的RecordReader类，使其能够识别时空数据文件格式。同样，我们实现了多个SpatialRecordReader类来处理不同可能的范围查询的格式。所有的FileSplitter和SpatialRecordReader都以两种形式实现:一种用于时空矩形范围查询，另一种用于时空多边形范围查询。

**Operation layer：**

我们将原有框架的范围查询扩展到了三维层面，以支持时空数据处理操作。具体的时空范围查询算法包括时空矩形范围查询、时空多边形范围查询以及时空knn查询、时空连接、时空圆范围查询等算法。在本篇论文中，我们主要关注使用最广泛的三种时空查询操作：Spatio-Temporal Range Query, Spatio-Temporal Polygon Range Query,Spatio-Temporal k-NN Query。

对于多边范围查询技术，目前有以下几种：

1. 集中式技术
2. Filter-define

Filter阶段根据多边形近似条件检索数据子集，然后细化这个子集，其中每个数据点都使用多边形中的点检查对精确的多边形几何进行测试。最常用的方法是获取已知多边形查询范围的最小外包矩形(MBR)，一般用表示。利用矩形方便快捷的查询方式获取几个近似的分区，然后再对这些分区进行更为细粒度的精确查询，判断点是否在多边形内。其他方法如下：

Using polygon MBR

Using more precise polygon approximations

Using multi-step filtering with consecutive polygon approximations

Using rasterization-based polygon approximation

多边形内点运算依赖于多边形的点的个数，对于复杂多边形来说，计算成本很高。精确的近似结果是一个紧凑的候选集，但但它们并没有减少每个点在多边形检查的计算复杂度。

1. Polygon decomposition

原始多边形被分解成规则形式，如凸多边形、三角形、梯形、矩形与三角形的组合、梯形、矩形与三角形的组合，甚至更小的不规则多边形使用均匀网格。然，每个较小的多边形被单独处理。该方法的局限在于，多边形分解技术尚未在分布式大空间数据系统中得到应用。

1. 分布式并行技术

这些技术主要依赖于跨机器集群或GPU内核进行数据分区，这样一个查询就会随着数据分区一起进行分区，然后查询会在多个有相关数据的节点/内核上执行。每个节点/Core使用filter-refine方法来处理多边形。

1. **PRELIMINARY**

**Spatiotemporal data types:**

实验采用雷达采集的飞行点轨迹数据，其中每一个航迹点数据的值包含两部分：时空属性和其他属性。时空属性：由GPS点的经度、维度、时间属性构成。其他属性：包含移动对象的ID等信息。

我们定义航迹点数据集为一系列点集的集合，其中每一条时空记录,表示经度，表示纬度，表示时间属性，表示点在空域中的高度属性。为了在空间中定位对象和计算点之间的距离，我们使用欧几里得空间计算距离。空间中两个点的距离我们用欧氏距离表示。每个航迹点的经度和维度，我们抽象为二维平面上点的坐标。二维平面上两点的欧氏距离为，三维空间上两点的欧氏距离为。

虽然时空数据库处理的的信息主要是时空对象(移动空间对象)，但本质上任何仍然是空间数据，大部分对时空对象的空间操作分析算法均可建立在控件对象的操作算法之上，其中时空对象的拓扑关系分析可基于对空间对象的拓扑关系。把时间因素看作第三维，时空拓扑关系便可被看作存在于三维空间之中。

1. **时空范围查询算法(Spatio-Temporal Range Query)**

**定义1：时空范围查询**

时空范围查询旨在查询在时空数据集中满足空间范围和时间间隔条件的所有记录。时空范围条件在很多基于位置的应用中都很有用，比如我们可以利用时空矩形范围查询一个飞机在某个时间段内是否在特定的空域中。

**定义1：**设为(d维欧几里得空间)中的一组点数据集,和一个矩形空间范围，和一个时间间隔T=，STRQ返回满足以下条件的集合：

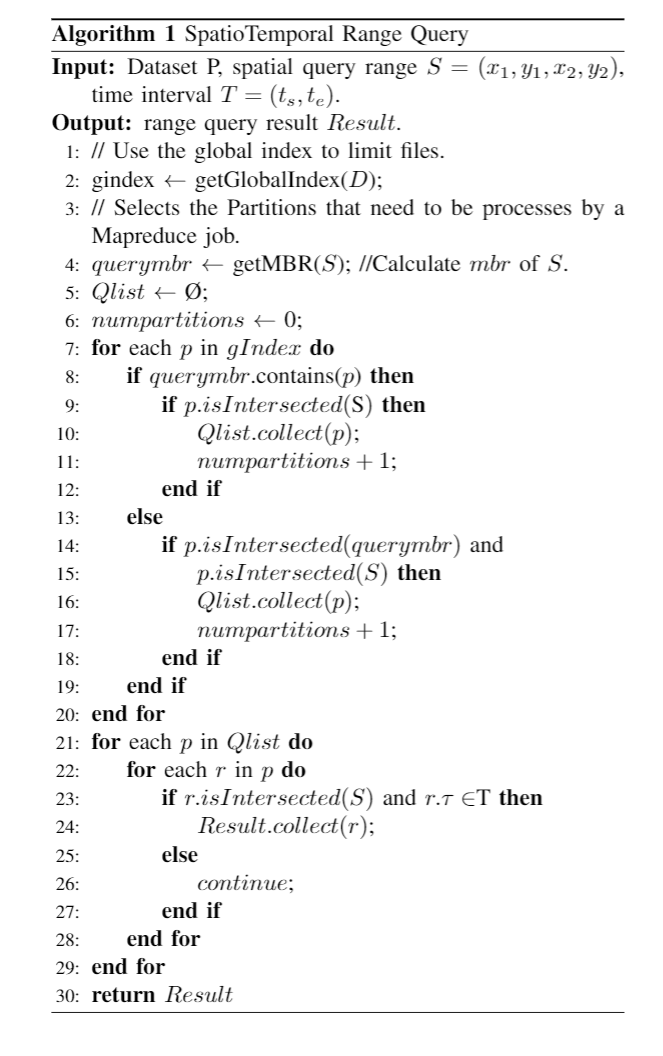




**算法描述**

算法1描述了矩形范围查询的处理流程。在第1行，我们获取源数据集的索引以限制文件大小。在第4行，我们获取最小外包矩形(MBR)，最小外包矩形一般用表示。第6-19行，我们利用全局索引选择只与最小外包矩形相交的分区，存入Qlist中。以过滤不含查询结果的数据块。第21-29行，我们对相交的分区中的所有记录进行筛选，仅当每个单独的记录在查询范围内（包括时间范围）才返回结果。

**伪代码**



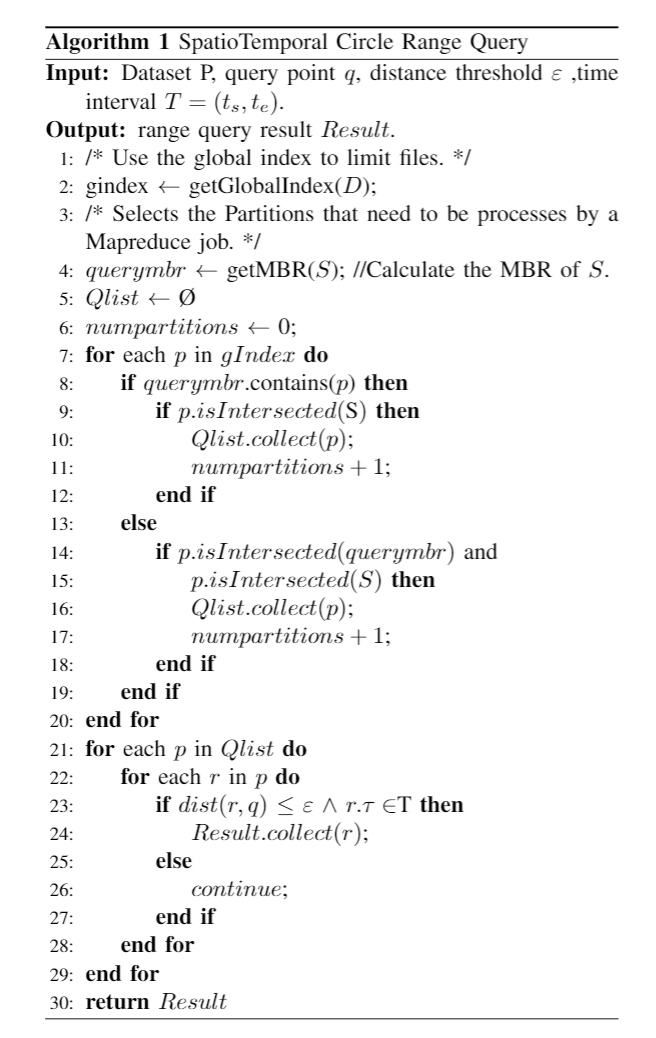
1. **时空圆形查询算法（Spatio-Temporal Circle Range Query）**

STCRQ在给定时间范围内，查询所有落在圆形区域内的点，圆形区域用一个查询点q和一个距离阈值ε表示。

**定义2：**设为(d维欧几里得空间)中的一组点数据集，q为中的一个查询点，距离阈值和一个时间间隔。STCRQ返回满足以下条件的集合：

**伪代码**



1. **时空多边形查询算法(Spatio-Temporal Polygon Range Query)**

给定一组时空点集，一组多边形范围和一段时间范围，STPRQ查询所有落在一组多边形区域范围内的点集合。

**定义3：**设为(d维欧几里得空间)中的由时点对象构成的一组时空数据集，对于每个对象，都有。多边形查询范围q都由一组空间点组成它的边界。通常chan用一个正整数n，一组x轴坐标集合，一组y轴坐标集合表示。时间间隔T=表示。对于时空多边形范围查询，我们有如下定义：

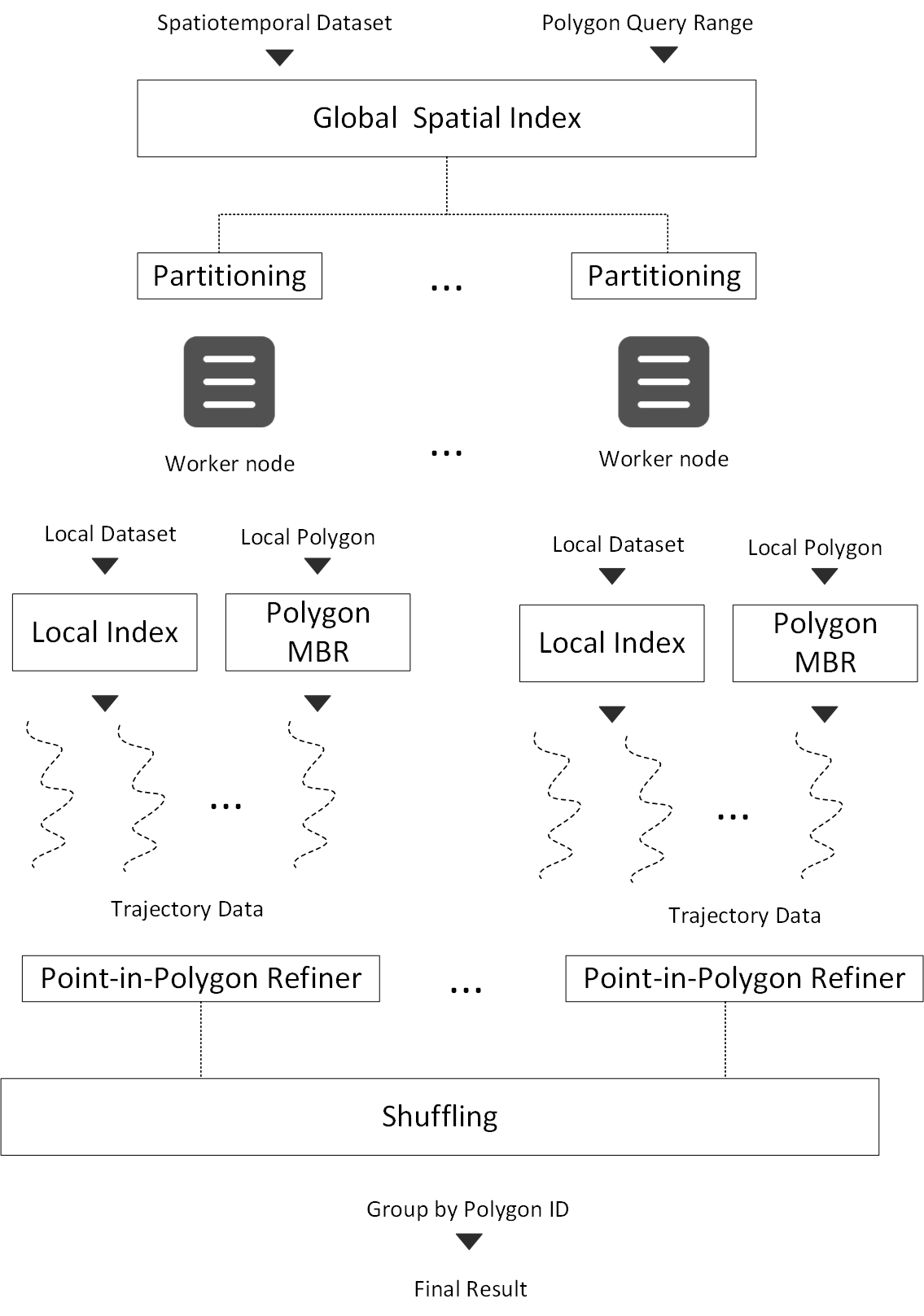
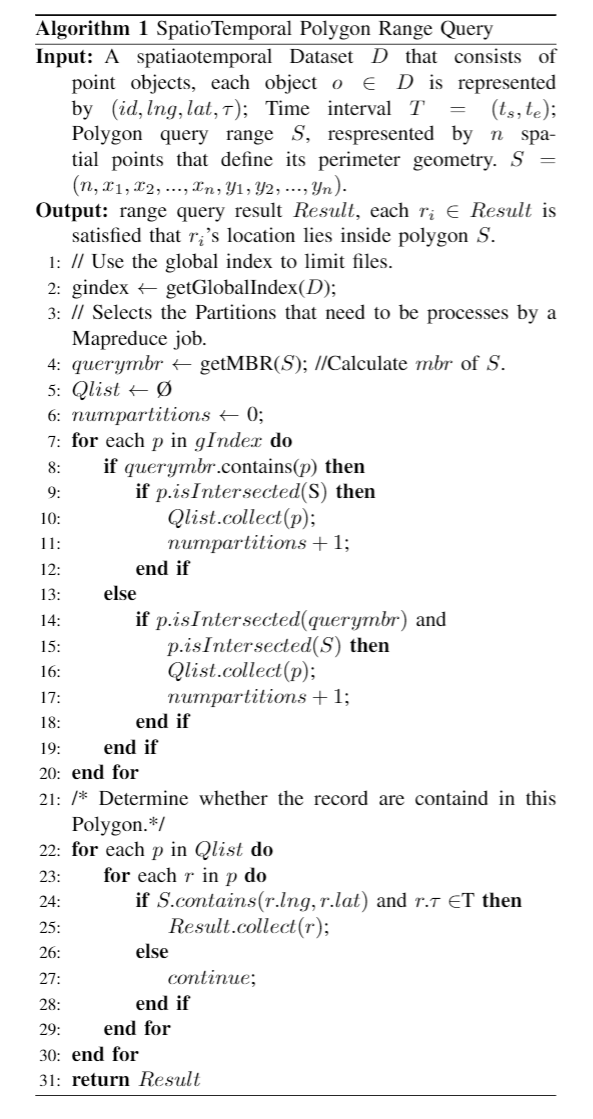




**算法描述**

算法1描述了时空多边形范围查询的处理流程。在第1行，我们获取源数据集的索引以限制文件大小。在第4行，我们获取已知多边形查询范围的最小外包矩形(MBR)，一般用表示。第6-20行，我们利用全局索引选择只与最小外包矩形相交的分区，共分为两种情况：内含于MBR且与多边形查询范围S相交的partition；对于与MBR相交，且与S相交的partition。这样，我们可以过滤掉不包含查询结果的数据块。第21-29行，我们对相交的分区中的所有记录进行筛选，仅当每个单独的记录在多边形区域内且时间满足时间查询条件才返回结果。

**伪代码**

****

1. **时空kNN算法(Spatio-Temporal k-NN Query)**

kNNQ获取在给定时间内，与给定查询点q最近的k个点。

**定义4:** 设为(d维欧几里得空间)中的一组点数据集，q为中的一个查询点，正整数和一个时间间隔。kNN返回满足以下条件的集合：



这里的dist(pi,q)是两个几何图形之间的距离函数。我们这里主要研究基于点的k-NN查询算法。本系统还支持其他基于非点集的记录，如线或多边形。

**算法描述**

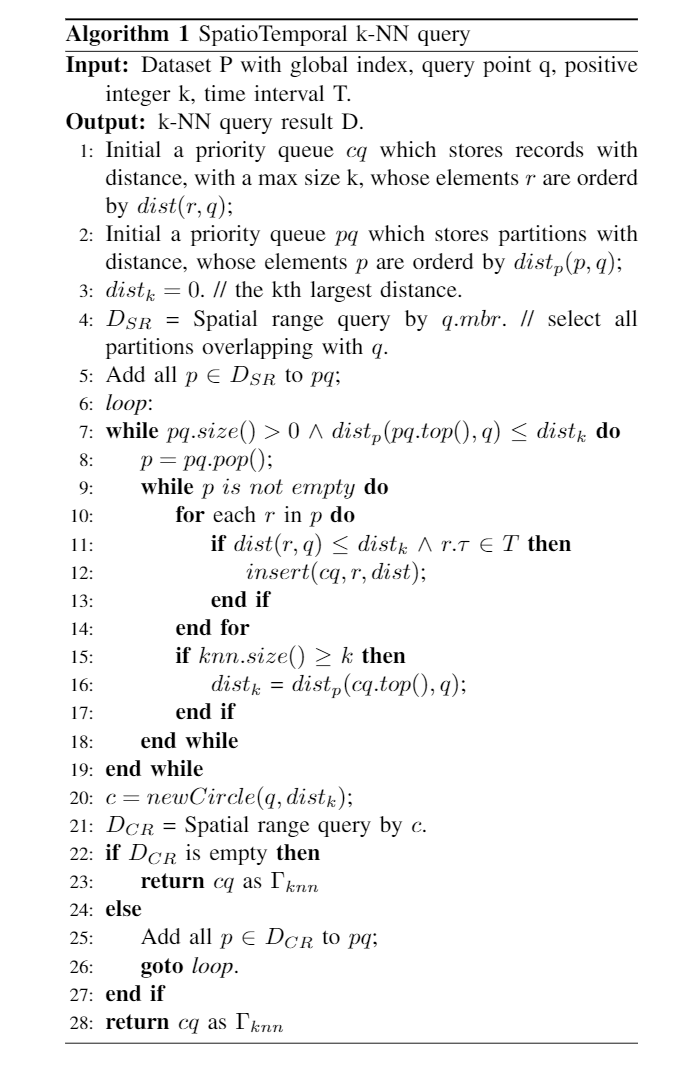
算法1给出了时空k-NN查询算法的伪代码。分为以下4个步骤：

1. 初始化(1-3行)。这一步初始化了一些变量。rq优先级队列用来存储满足knn条件的候选记录。pq是一个记录将要查询遍历的所有数据块的优先级队列。distk存放的是目前在rq队列中与查询点q距离第k个近的距离值。我们定义查询点q到分区p的距离为点到分区的最小距离，用如下公式定义：



1. 生成初始化答案(4-19行)。我们首先使用范围查询找出所有与查询点相交的分区，并将所有分区加到优先级队列。接着，对优先级内的分区进行条件匹配，当分区距离小于当前distk时，才将分区从优先级队列弹出，并遍历分区中的每条单独的记录，时空knn的特点是，当前记录的时间要满足时间范围,即。这一过程将产生k个初始结果，并更新distk。
2. 正确性检测(20-23行)。这个步骤将检验k个初始结果是否是最终结果。我们以测试点为圆心，distk为半径生成一个测试圆C，如果C不与其他分区相交，那么k个初始结果即为最终结果，直接返回cq。否则，需要进行最终结果的确定步骤。
3. 确定最终结果(24-28行)。我们启动一个范围查询们获取测试圆C中的所有点记录，回到第六行的循环，对结果进行遍历，最终确定k个在时间范围内离查询点最近的点。

**伪代码**

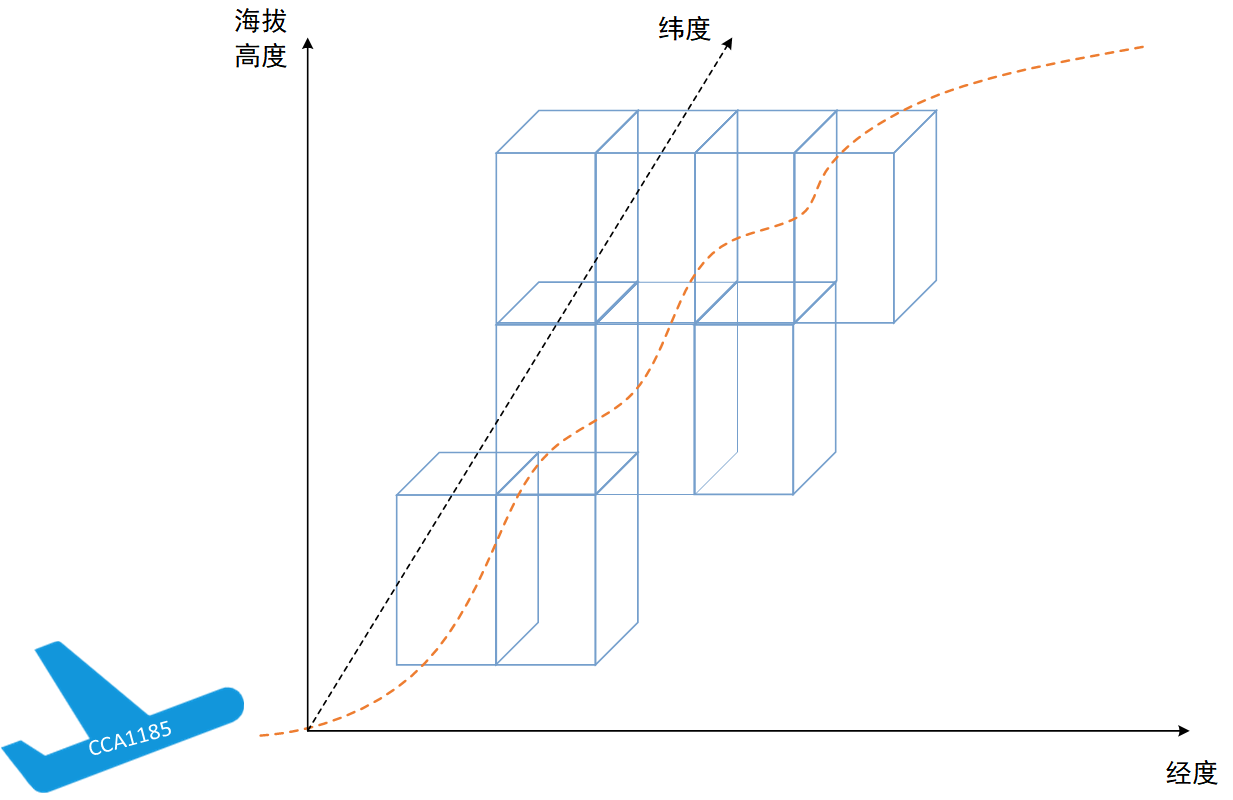


1. **分析算法**
2. **统计空域内飞行器流量的分析算法**

**应用场景描述**

统计空域内飞行器流量的分析算法旨在统计某个时间段内某个空域的飞行流量，从而为后期制定流量管理与控制策略提供数据支撑。飞行器的航迹点经过的区域多为扇区/终端区，扇区通常由多个规则的空间柱体组成，柱体包含由高度属性和一个多边形的截面组成，是一个三维范围区域。我们此前讨论的多边形范围查询算法只涉及二维平面区域内的范围搜索算法。如果要对空域内的飞行器流量进行统计，则需要三维层级的范围查询算法。

我们定义三维空间范围域用表示，其中。其中，S表示柱体的水平截面，该水平截面有多个点组成，表示柱体的上界和下界值，构成柱体的高度信息。



**扇区空域数据的数据结构描述：**

扇区空域数据是静态的，因此可以作为范围查询的参数。空中扇区的表面通常凹凸不平，高低起伏，呈不规则分布。但扇区可以拆分成多个规则的空间柱体，柱体包含由高度属性和一个多边形的截面组成。数据描述如下图所示：一个扇区包含扇区ID和一系列Volume集合，还有能唯一确定扇区的id属性。柱体Volume又由高度下界lower、高度上界upper和一系列点集构成，Volumename属性唯一确定一个柱体。最后，点集合构成一个多边形平面，点包含经度、纬度等属性信息。

数据描述如下所示：

Area: <Id, List<Volume>,...etc >

Volume: <Id, lower,upper,List<Point> ,...etc>

Point: <pointname,longitude,latitude,...etc>

**雷达采集的ADSB数据描述：**

飞行器的航迹一般是一系列轨迹数据，由许多位置连续、时间连续紧密的航迹点组成。我们的航迹点由雷达以时间间隔为3秒的频率进行一次采样获得。航迹点的属性包括经度lng、纬度lat、时间time三个维度的属性，也包括航班号ID，飞行高度altitude等其他属性。数据描述如下图所示。为了系统的灵活性，我们的扩展还支持读取其他类型的数据结构。

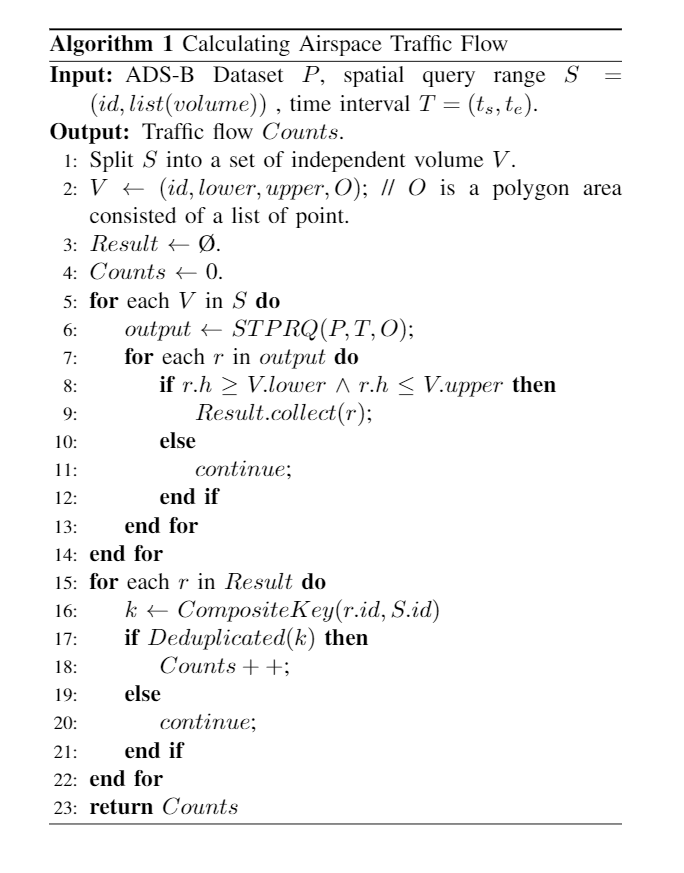
TrajectoryData: <Id, lng,lat, time,altitude,...etc>

**算法描述：**

我们的输入包括ADS-B数据集，空间查询范围S是由一系列柱体组成的集合，还有时间范围T。我们预期的输出是交通流量的统计值Counts。

在算法第1行，我们将范围拆分为一个个独立的空间柱体V，我们在第2行定义了柱体的数据属性。第3-4行初始化了一些变量，Results用来存放中间输出结果，Counts用来存放流量统计的最终结果。在5-14行，我们利用STPRQ算法结合柱体的高度条件进行筛选，查询在ADSB数据集在S内所有记录。在15-23行，我们对中间输出结果进行是数据去重的操作，保证飞行器经过某个扇区的次数仅出现一次，确保结果的真实性和准确性。

**伪代码：**



1. **统计航班关联计划的分析算法**

**应用场景描述**

**算法描述**

**伪代码**

1. **性能实验**

在这个部分，我们首先描述数据集和实验配置，然后给出实验性能的评价结果。

1. **数据集实验配置**

**数据集：**为了评估NHhadoop的实验性能，我们使用一个飞机的轨迹数据集、一个合成数据集和一个空域范围数据集：

1. **ADSB航迹点数据**：雷达采集的半年内包含1000万条航班轨迹记录
2. **合成数据集**：通过复制和采样得到的航迹点数据，来测试系统的扩展性。我们使用一个合成轨迹数据集进行了一组实验。
3. **扇区静态数据**：详细描述了民航空域中扇区静态数据的结构，是一个三维的空域空间边界数据，用来在统计空域流量的算法中精准的描述空间范围边界。

表2：数据集参数

|  |  |  |
| --- | --- | --- |
| Attributes | Traj | Synthetic |
| # Records | 10million | 100million |
| Raw Size | 379MB | 10.3GB |
| Time Span | 2019/01/01-2019/07/01 | 2019/01/01-2019/07/01 |

**实验配置：**

我们测试了NHhadoop的时空范围查询、时空多边形范围查询和时空k-NN查询的效率。表3给出了我们的数据集实验配置。表4给出了查询参数配置。表5给出了实验中的硬件版本信息。

为了实验结果的真实性，我们随机选取10个不同的查询参数，每个查询执行一次，取其响应时间的平均值作为最终结果。

为了保证实验的高可用性和副本不易失性，所有的实验在一个6个节点的集群上进行，其中1个master节点，5个slave节点，每个节点配置如下：CentOS-7， 8-core CPU，16GB RAM，和200GB的磁盘。每个节点的block大小是64MB，复制因子为3个副本数量。

表3：数据集大小和时间查询窗口、空间查询窗口的设置

|  |  |
| --- | --- |
| Parameters | Settings |
| Data Size(%) | **20**,40,60,80,100 |
| Time Window | 10d,**1m**,2m,4m,6m |
| Spatial Window | 10\*10,20\*20,30\*30,40\*40,**50\*50** |
| K for kNN | **50**,100,150,200,250 |

表4：实验环境版本设置

|  |  |
| --- | --- |
| Software | Version |
| Hadoop | 2.10.0 |
| JDK | 1.8 |
| Idea | 2020.2.1 |
| Maven | 2.6.0 |
| Python | 3.8.4 |

Baselines：我们选取st-hadoop作为对比算法，因为st-hadoop可以同时支持时空范围和时空k-nn算法。为了验证索引策略的有效性，对于每一种时空范围查询算法，我们选取两种索引进行横向比较，即str和kdtree。对于对比算法st-hadoop，我们为其构建不同的索引作为对比，即按两种不同的时间粒度month、week构建str索引。

Our-str:使用str树建立空间索引

Our-kdtree:使用kdtree树建立空间索引

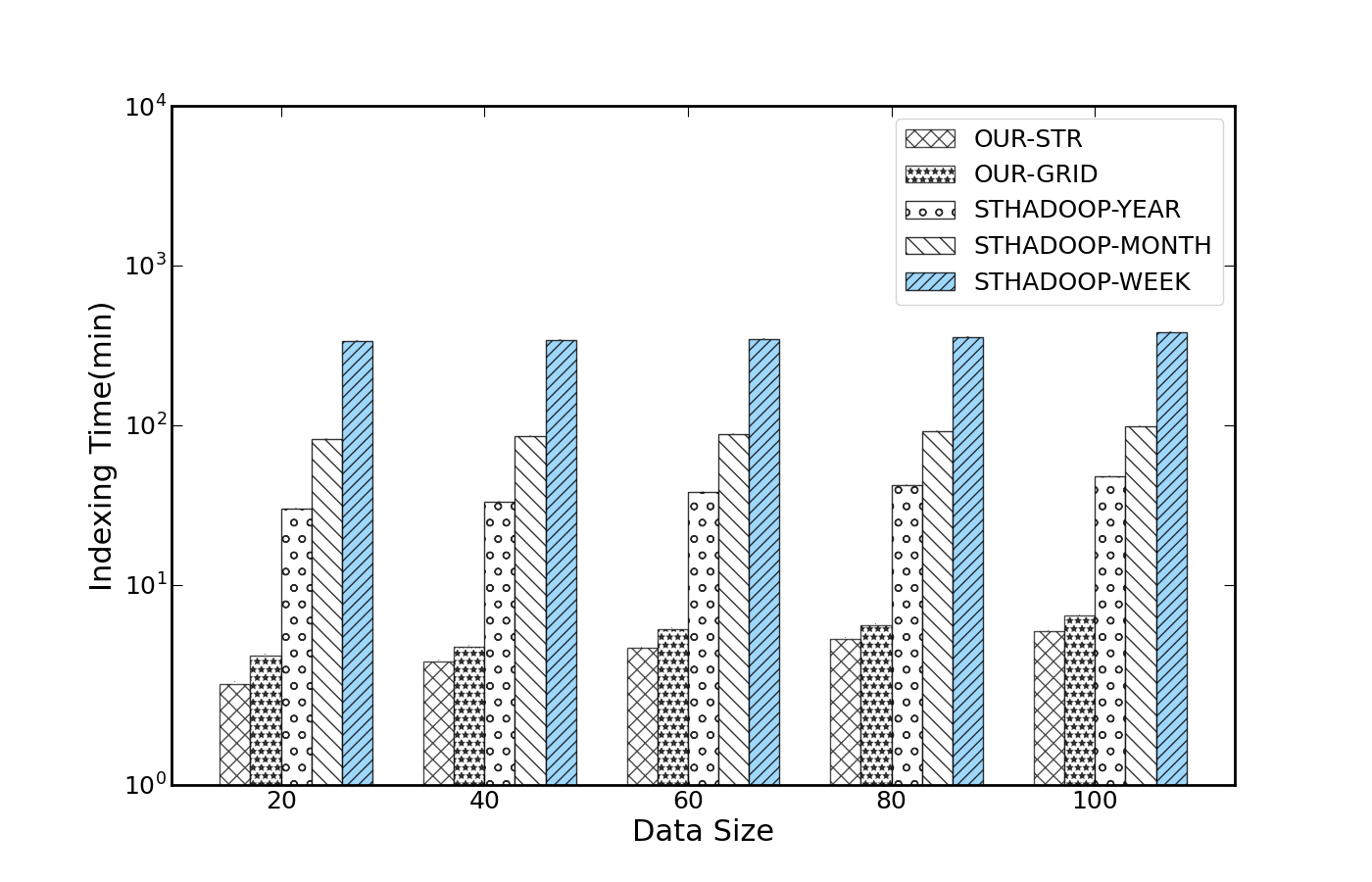
STHadoop-week:使用str树建立空间索引、以week为粒度的时间切片

STHadoop-month:使用str空间索引策略、以month为粒度的时间切片

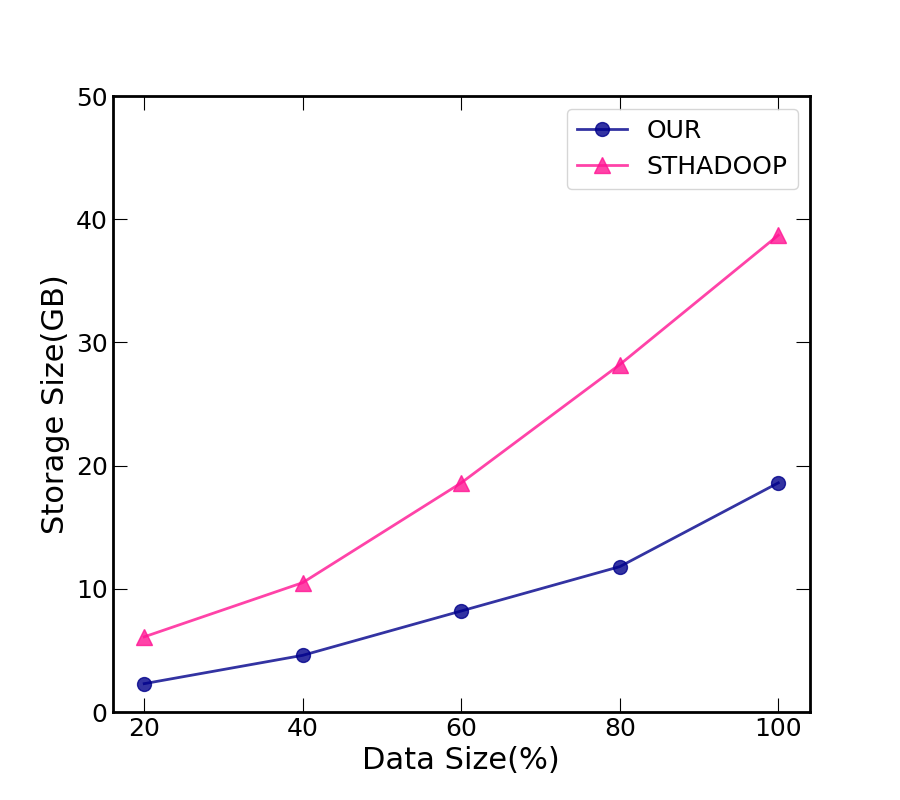
对比算法：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| System | SRQ | STRQ | kNN | ST k-NN | STPRQ |
| ST-Hadoop | √ | √ | √ | √ | √ |
| Spatialhadoop | √ | × | √ | × | × |

1. **索引的性能**



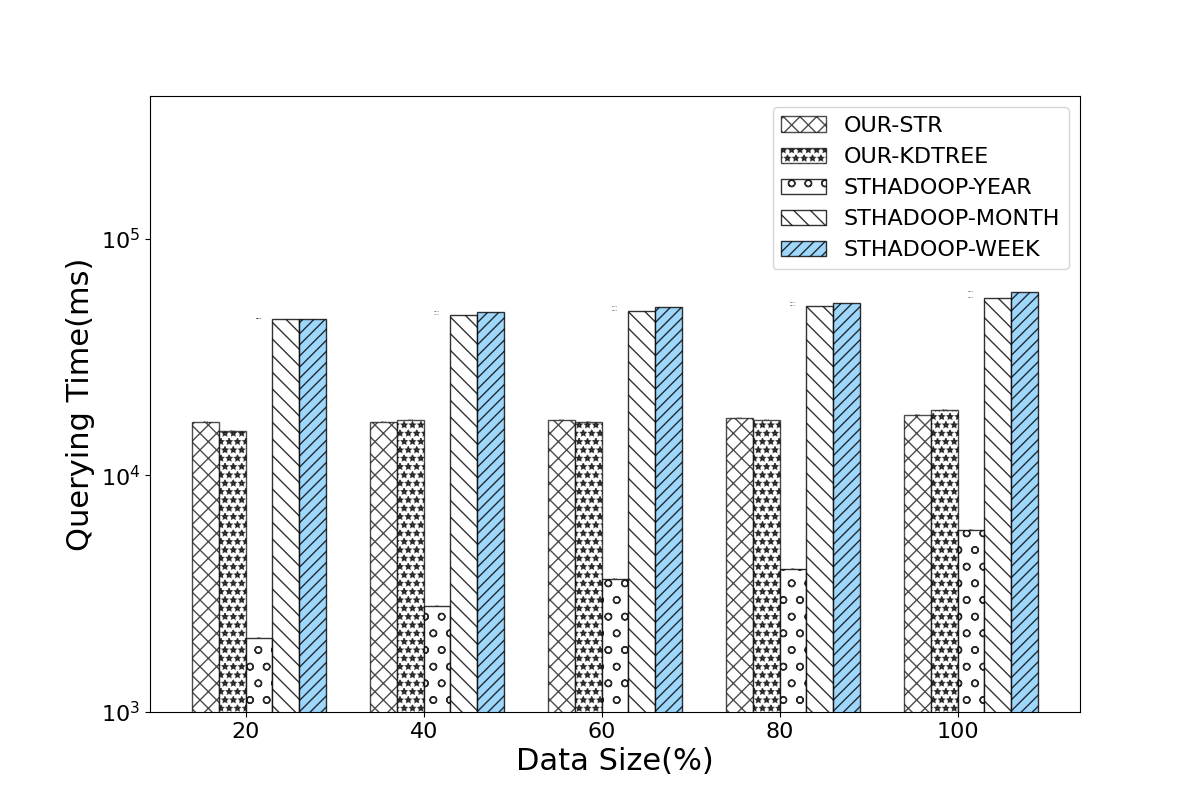
图中所示的索引时间为创建和存储索引的时间开销。可以看到，随着数据量的增长，建立索引的时间不断增加。图中包含五个对比算法: OUR-STR, OUR-KDTREE,STHADOOP-YEAR,STHADOOP-MONTH, SYHADOOP-WEEK。图中可以看出，STHadoop建立索引的时间比OURS要多出200%-300%，这是由于STHadoop本身采用了层级结构的索引机制,先根据支持不同时间粒度进行时间片划分,再对每个时间片进行空间上的索引,因此索引时间大大增加。

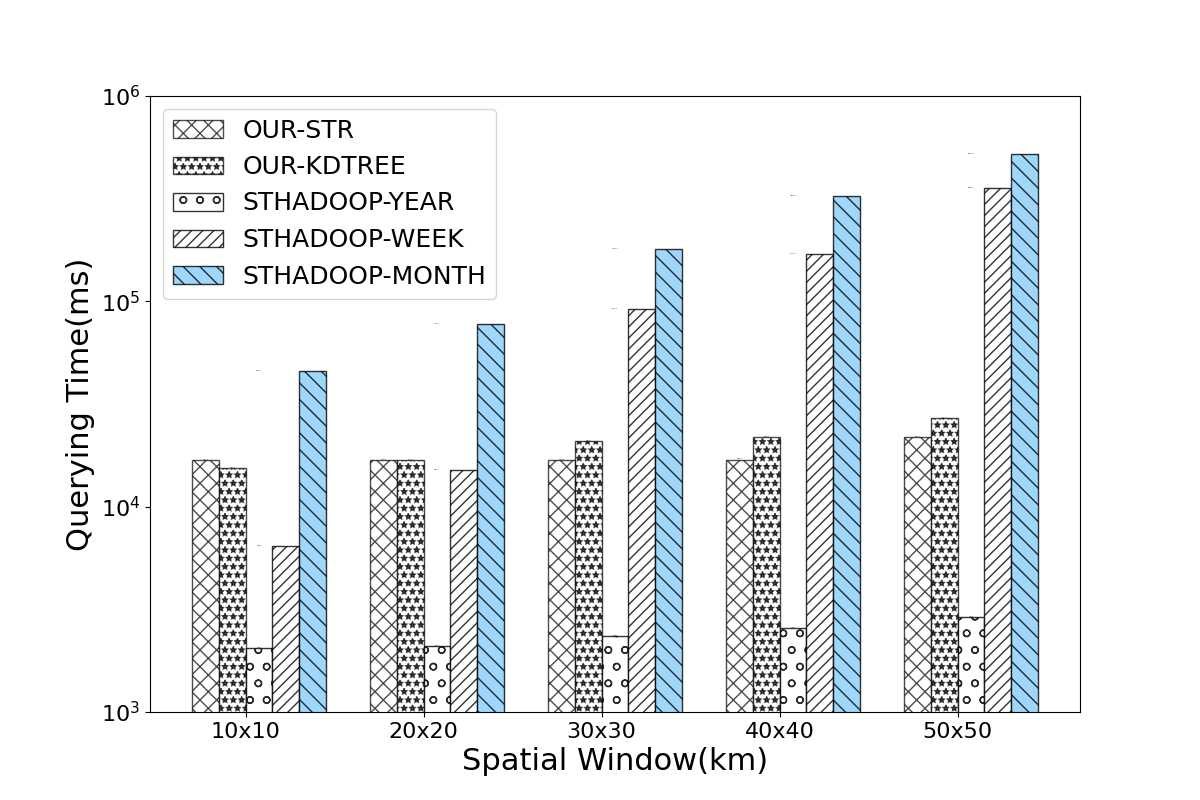


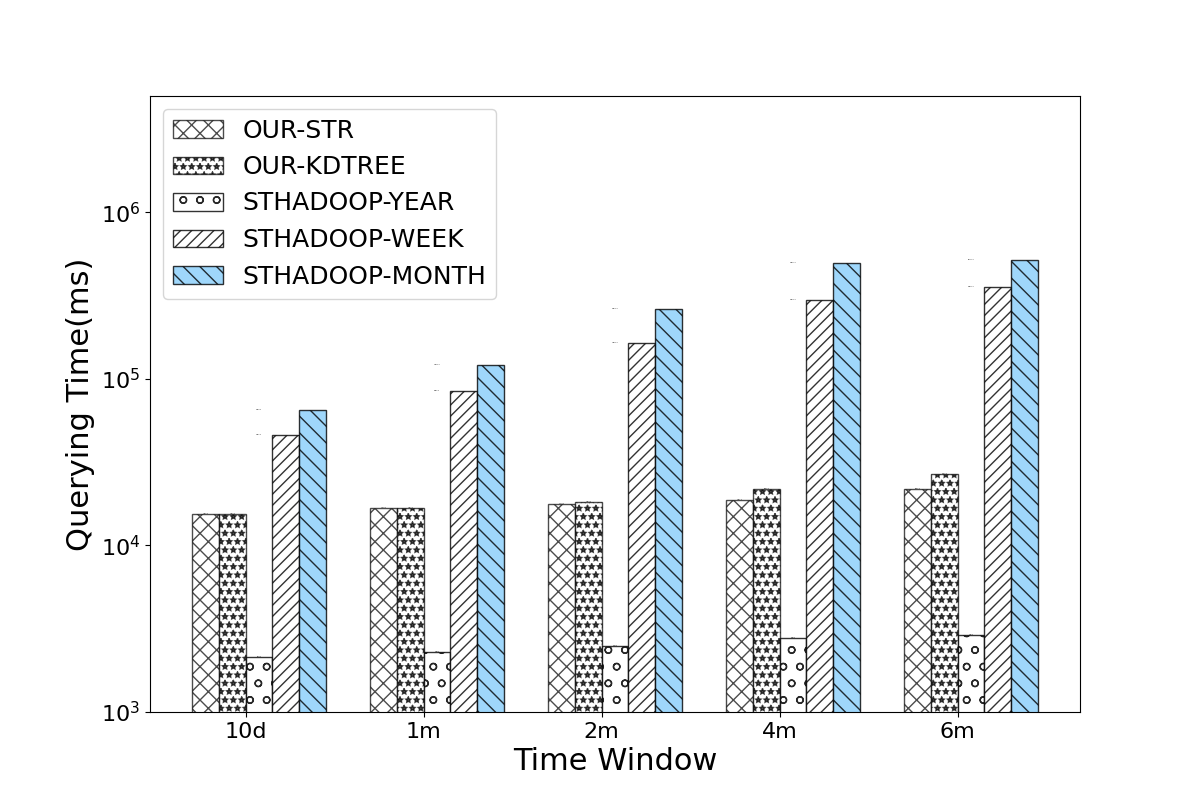
这里的存储大小指的是索引的存储成本，包含索引的元数据和各个数据块。

随着数据量的增长，索引占用的存储空间越来越大。因为我们将数据存储在HDFS上，每个数据的复制因子为3，因此会不可避免的导致数据膨胀。ST-Hadoop花费大量的时间进行序列化和存储索引。为了应对不同粒度的时空范围查询，必须要对数据进行不同时间片的切分，这种牺牲空间换取时间的操作，使得存储空间大大增加。

1. **时空范围查询算法的性能**







有以下几点发现：

**不同的数据集大小：**所有时空查询的时间随着数据大小的增大而增加，因为扫描和返回符合条件的轨迹数据越多，触发的磁盘I/O越多。

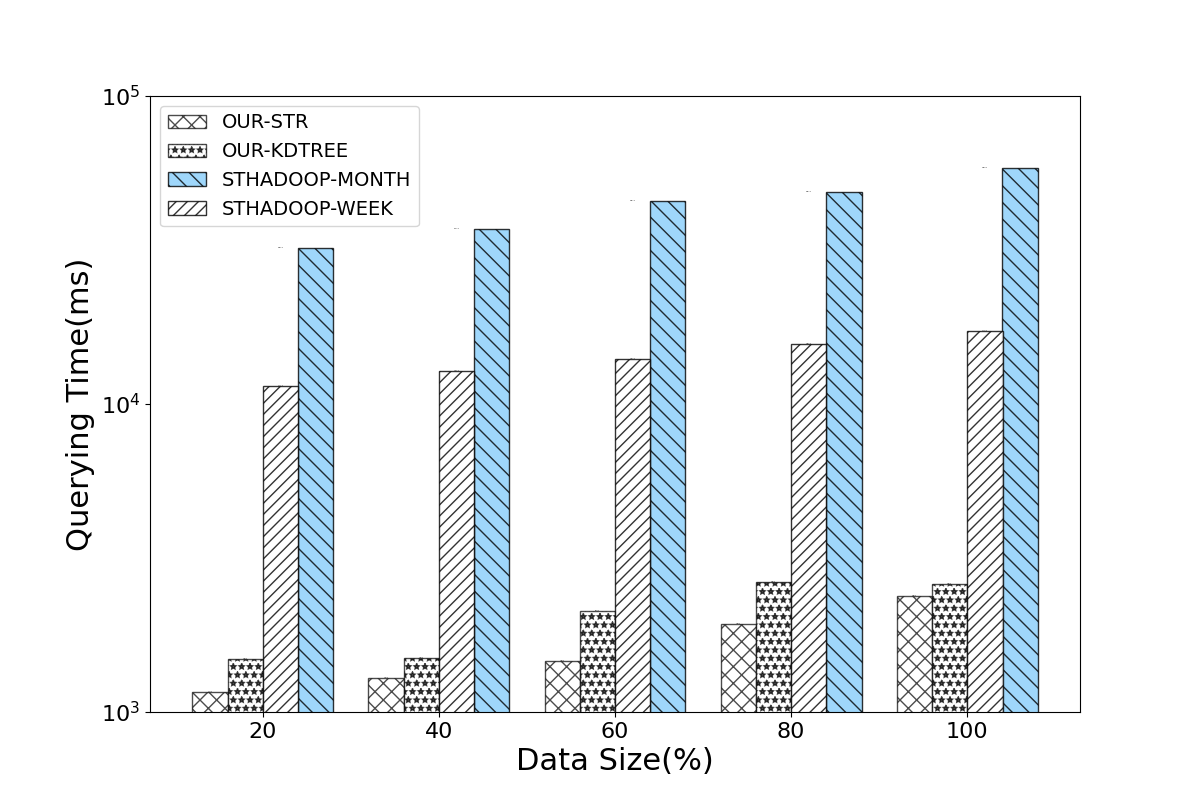
**不同的空间窗口：**随着空间窗口的增大，查询范围内获取到的数据不断增加，每种算法都需要更多的时间来应答时空范围查询。我们的系统的查询效率普遍要高于相STHadoop，除了在以年为粒度的ST-hadoop索引的时空查询。这可能是因为STHadoop的不同时间层级的索引结构占据在HDFS的分区数不同，以年为粒度的索引分区少，从而寻址延迟比较小。而基于de’fen’kau月和周为粒度的时空索引带来的时间延迟比较大。

**不同的时间窗口：**图中比较了航迹点数据不同时间窗口下的时空查询，随着时间跨度的增加,查询范围内获取到的数据不断增加,因为时间跨度增加意味着更多满足条件的记录。从图中可以得出,时间窗口的增加对NHHadoop的性能影响微乎其微。因为建立了有效的空间索引，先进行空间位置上的定位后，再对相关记录进行时间的过滤，因为每次扫描都需要遍历同样时间范围的数据，所以时间窗口的大小对其性能没有什么影响。相反，STHadoop的查询时间大小取决于时间窗口的大小和时间窗口内的数据块大小，所以，在数据密集的时间窗口内，STHadoop的性能表现很差。

1. **时空K-NN算法的性能**

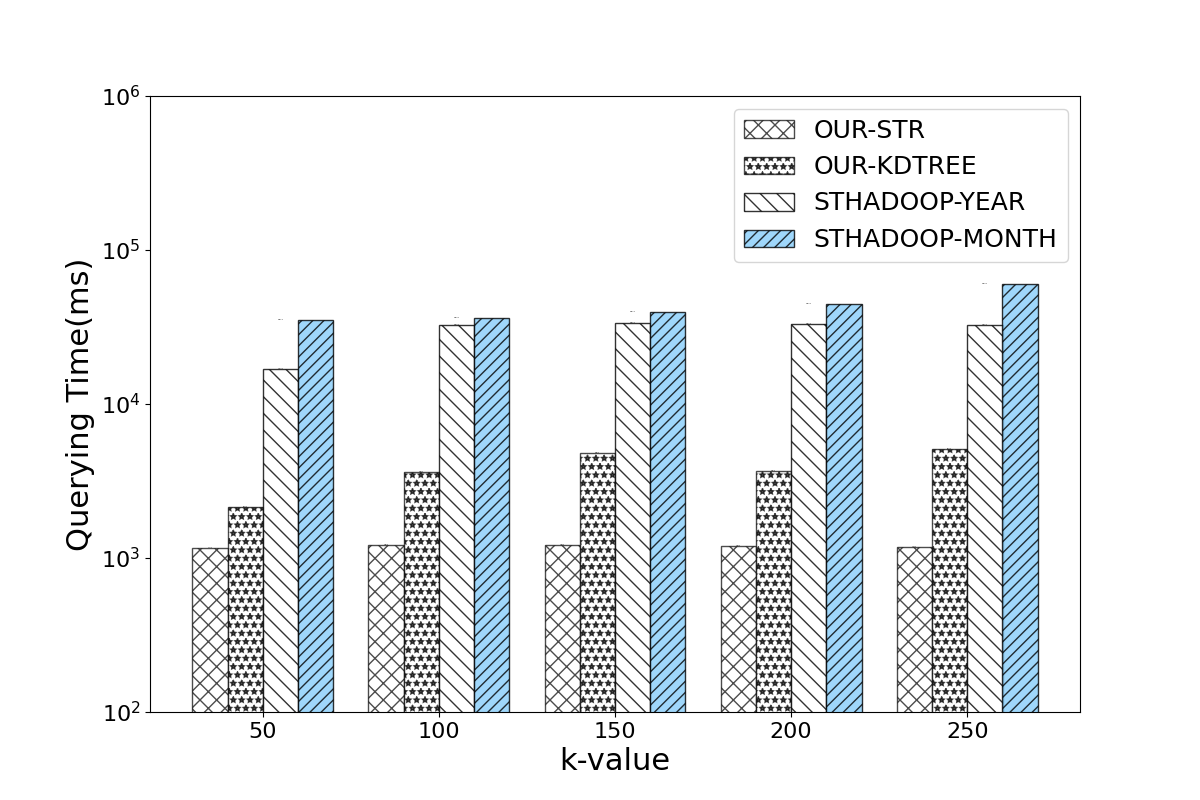
衡量指标：查询时间的大小（ms）

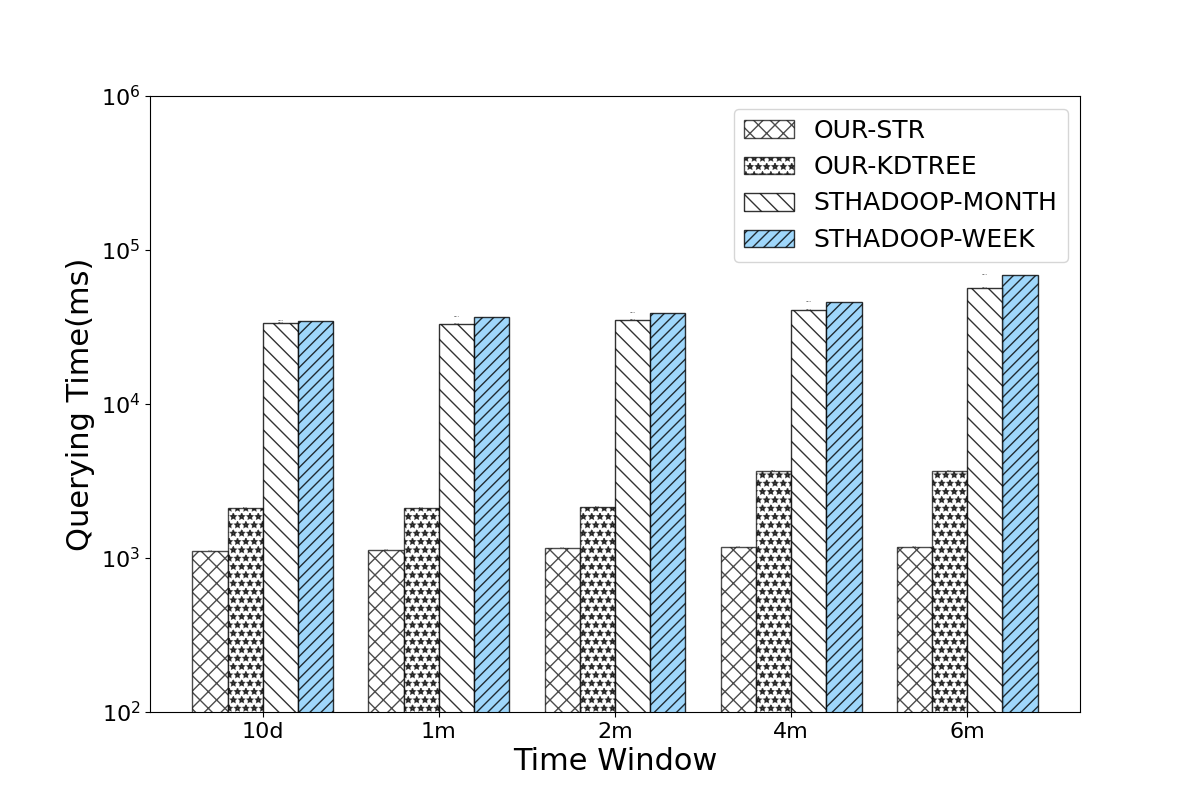
**不同的数据集大小：**随着数据集的增加，KNN查询的时间不断增加。因为在KNNquery中，k-NN查询会触发空间范围查询算法，因此涉及扫描更多的记录。我们的算法普遍比ST-Hadoop要好，因为前面已经证实ST-Hadoop基于年/月的范围查询性能不如我们的算法。



**不同的k值：**如图所示，对于更大的k，所有系统都需要更多的时间，因为我们应该扩展更多的时间并触发更多的空间范围查询，以获得最多的k个最近记录。

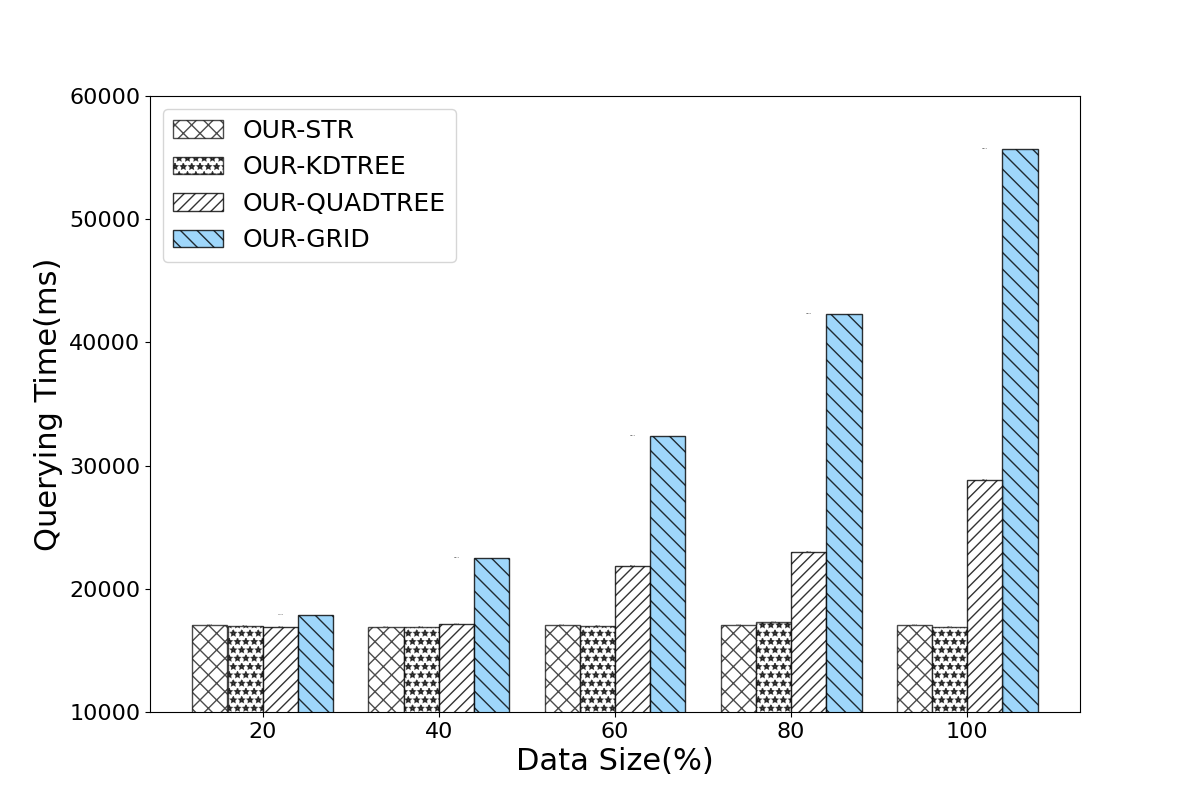
**不同的时间窗口：**随着时间范围的增加，我们发现系统的查询时间没有明显增加.这是因为k-NN查询的结果与时间范围无关，每次范围查询的结果是固定的。

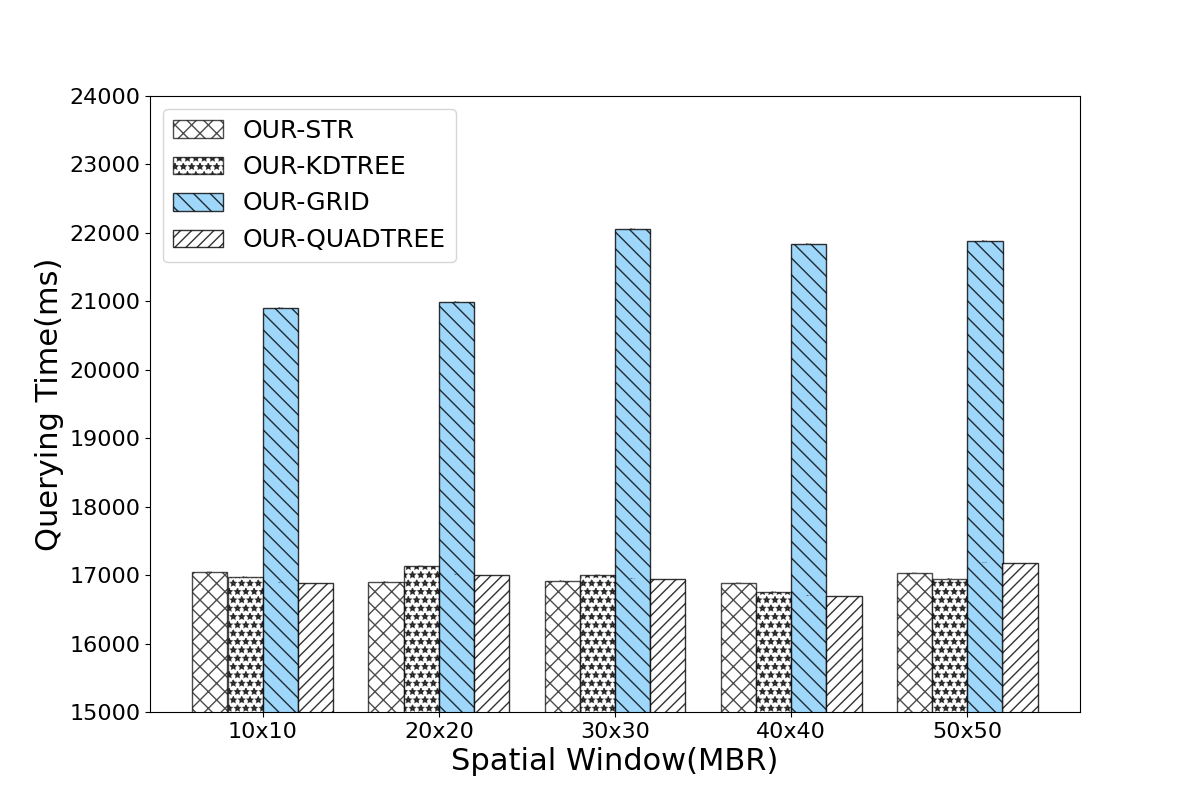


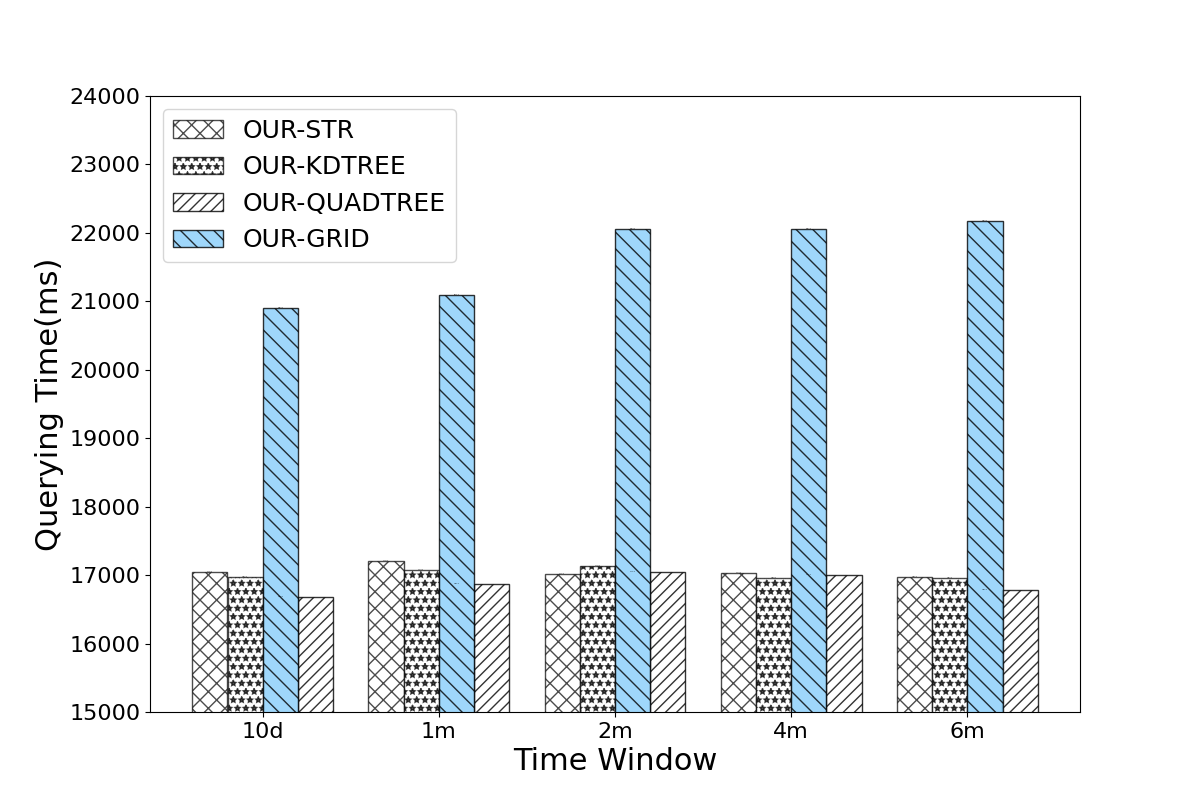


1. **时空多边形范围查询的性能**

时空多边形范围查询的性能与时空范围相似,因为时空多边形范围也是基于多边形的最小外包矩形,找到与MBR相交的数据块,再进行多边形范围的查询.在此不多赘述。我们发现,基于grid索引的时空多边形范围查询的性能明显低于其他索引,这是因为网格索引的本身结构所致,匹配到与范围查询相交的数据块会明显增加.

****



****

**工作总结及展望**

**致谢**

**参考文献**