分　类　号：TP39　　单位代码：10183
研究生学号：2013532135　　密　　级：公开

吉 林 大 学

硕士学位论文

云平台下时空数据存储与索引机制的研究

Spatio-Temporal Data Storage and Index Research on Cloud Platform

作者姓名：杨学毅
专　　业：计算机应用技术
研究方向：数据挖掘与数据存储
指导教师：黄岚 教授
培养单位：计算机科学与技术学院

2016 年 4 月

吉林大学硕士学位论文原创性声明

本人郑重声明：所呈交学位论文，是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：杨学敏

日期：2016 年 5 月 26 日

云平台下时空数据存储与索引机制的研究

Spatio-Temporal Data Storage and Index Research on Cloud Platform

作者姓名：杨学毅

专业名称：计算机应用技术

指导教师：黄岚　教授

学位类别：工学硕士

答辩日期：2016 年 5 月　日

摘 要

云平台下时空数据存储与索引机制的研究

近年来，大量的 GPS 设备产生了大规模的具有时间和空间位置的数据，我们称之为海量时空数据。目前主流的时空数据存储方法大多基于单机空间数据库，由于单机性能有限，扩展性不高，渐渐不能适用于处理大规模时空数据。随着分布式云计算的高速发展，众多高性能的云平台层出不穷，为处理海量时空数据提供了契机，但是这些云平台在机器成本、能耗、实验场地等方面造价高，对普通用户收取的费用也是昂贵的。

当前，时空索引的研究大多是单机下的串行索引，分布式时空索引很少研究。存储在分布式云平台上的海量时空数据，不但数据存放无序，而且需要遍历每条记录，才能检索到用户所需的内容，检索效率低下。所以，时空数据在云平台上的存储策略以及索引构建上面临巨大压力。如何利用云平台的高性能计算能力，有效的存储与检索时空数据，是必须解决的关键技术问题之一。

基于以上问题，通过大量的准备工作，本文提出以下解决方案。首先，采用低成本低能耗的 Cubieboard2 ARM 开发板搭建 Hadoop 分布式云平台，分析云平台性能与能耗，验证其实用性。然后，在 HDFS 层次上，设计了两种全局-局部索引，即 TGrid 索引和 QDtree 索引。TGrid 索引采用改进的网格划分算法将时空数据均匀的划分到 HDFS 数据块中，每个数据块构造一维时间的局部索引；QDtree 索引采用改进的四叉树划分算法划分时空数据，构建多维 R-tree 的局部索引管理数据块中的数据。最后，设计存储优化策略，采用列存储与数据压缩的方法，减小磁盘存储空间，提高数据检索和网络传输效率。实验表明，该云平台虽然单机性能有限，但能够充分发挥 Hadoop 可扩展性，发挥并行计算的优势，弥补单节点的不足，而且成本低廉，有很好的借鉴意义。通过本文提出的两种时空索引机制和存储优化策略，不但合理的存储海量时空数据，节省存储的开销，而且索引对时间和空间属性的有效剪枝，大大提高了数据检索效率。

关键词：

海量时空数据，分布式时空索引，云平台，存储优化

Abstract

Spatio-Temporal Data Storage and Index Research on Cloud Platform

In recent years, a number of GPS devices have produced a large scale data with time and spatial position. We call it massive spatio-temporal data. The prevalent method to store spatio-temporal data is based on spatial databases. It is difficult to deal with massive spatial-temporal data because of the low performance and scalability. With the development of distributed cloud computing, many high performance cloud platforms have been produced, which can be utilized to process the huge spatio-temporal data. But the cloud platforms have high cost in the machines, energy consumption, experimental sites and other aspects. It is also expensive to use the platforms for ordinary users.

Nowadays, most of the spatio-temporal studies mainly focus on the serial index but rarely on the research of distributed spatio-temporal index. The spatio-temporal data should be read one by one to retrieve the desired content with low efficiency in the cloud platform. Therefore, a storage strategy and distributed indexes of spatio-temporal data should be properly addressed to take the advantage of the cloud platform.

Based on the above issues, we propose the following solutions through a lot of preparatory work. First of all, we build a Hadoop cloud platform using ARM development boards called Cubieboard2. We analyze the performance and energy of the cloud platform to verify its practicability. Then, on the level of HDFS, we design two kinds of global-local indexes which are TGrid and QDtree. TGrid divides the data into HDFS data block by improved grid algorithm. One dimensional time index is built in the block. QDtree divides the data into HDFS data block by improved quad tree algorithm. 3DR-tree index is built in the local block to manage the data. Finally, we design the storage optimization strategy. In order to improve the disk utilization and network transmission, we use the column storage and data compression to optimize the storage structure. The experimental results show that our cloud

platform is high scalable and parallel computing. It is a good reference. The two indexes and storage strategy we proposed can store the massive spatio-temporal reasonably and save the storage cost. The indexes improve the query efficiency by pruning time and spatial attributes.

KeyWords:

Massive Spatio-temporal Data, Distributed Spatio-temporal Index, Cloud Platform, Storage Optimization

目 录

第一章 绪论

我们的生活与时间和空间息息相关。例如车辆的流动、流感的传播、土地的沙化、飓风的移动、洪水的泛滥等等，这些都体现时间与空间的属性。随着时间的推移，空间也在不断改变，我们称这种类型的数据对象为时空数据。当前，有效的存储和管理海量时空数据是亟待解决的重要课题，而基于分布式的云平台，高效的时空索引又是其中的关键技术，这不但可以降低时空数据并发控制成本，同时又能充分发挥计算机性能，快速准确的检索时空数据。本文就是基于单机性能低下的 ARM 板，搭建低成本、低能耗的云平台，充分发挥云平台性能，研究分布式时空索引的高效性。

1.1 课题研究背景及意义

近年来，随着技术的更新与发展，移动互联网逐渐进入到人们的生活中，越来越多的用户通过移动设备连入互联网，极大的丰富了人们获取信息的方式，也极大的促进了信息服务水平的提高。有统计显示，截止到 2015 年底，我国手机网民规模达到了 6.20 亿，众多企业积累了大量的个人位置信息。国外调查报告显示，74%的智能手机用户使用手机获取地理相关信息和服务信息[1]，这使得基于位置的服务（Location Based Service，LBS）相关技术与应用模式发展越来越迅猛[2]。LBS 需要获取移动终端或者用户的地理位置信息，一般是通过通讯运营商（如移动、电信）的通讯网络或者其他定位方式如 GPS，并基于 GIS 系统，借助互联网或无线网络，满足用户的需求，提供及时便利的服务。LBS 可以为多种应用提供必需的信息支持，在商业上[3]、公共事务管理[4]中具有极高的应用价值，例如手机端的美团、微信查找周围朋友等。随着 LBS 应用的不断发展，需要处理的时空数据在复杂度和数量方面也呈快速增长趋势，其中数量更是急剧增长。而随着对大量时空数据的分析与挖掘，也带来了传统生活模式的改变，如根据用户的习惯和位置信息推送相应的广告；在旅游业中根据用户的移动位置推荐旅游景点、饮食、住宿；政府通过行为模式的分析及时调整社会管理政策等。

从时空数据的角度分析，不同 LBS 应用的时空数据模式和数据量的规模差异很大，同时这些应用的服务请求量在高峰期和平峰期变化也很大。比如“街旁”和“快的打车”

这两类应用的时空数据量差距巨大，而“快的打车”这款应用在高峰期和平峰期数据的访问量差距也很大。因此，存储时空数据的时空数据库要有很好的弹性和扩展性，但是目前我们常用的时空数据库有很大的劣势，不但处理数据规模较小，不能适用于海量时空数据，而且扩展性差，硬件成本高等。2014 年 12 月 15 日，“快的打车”因用户数量快速增长，订单的数据急剧增加，出现了大面积的服务不可访问的情况，这充分说明了可扩展性的重要性。不同的时空数据库对时空数据的处理有其不同的侧重点，一些应用系统不仅仅对海量时空数据的存储有很高的要求，而且高效的检索性能也是必须的，这对于分布式时空数据库的设计有很大的挑战，也体现了研究的意义，能在海量时空数据中挖掘有价值的信息。

随着云计算技术的发展，各种云平台相继出现，能够为海量时空数据提供大量的存储空间和高效的计算能力。但是相应的需要大量的计算节点，成本和能耗较高，对于企业来说，在不减少服务质量的同时能够降低成本，将大大提高竞争力。采用一种可以代替传统 PC 机的 ARM 开发板作为计算节点是本文研究内容之一。目前大多数时空索引的研究还局限于单机，大量计算节点的分布式索引研究很少，分布式时空索引是研究的关键点。这不但可以优化时空数据的存储，将时间和空间相近的时空对象聚集在一起。而且，检索时可以对时间属性和空间属性进行大量的剪枝，快速定位到查找内容，所以分布式时空索引的研究是非常重要的课题。

1.2 国内外研究现状

随着时空数据规模日益增大，时空数据库相关的各种数据管理技术逐渐吸引了学术界的研究目光。其中以研究具有可扩展性、支持并发性、支持高效检索的分布式时空数据库为热点和难点，这其中包含多个时空数据库关键技术点。如，大规模时空数据的存储结构、基于时空特征的信息检索、高效的分布式算法等，在这些领域已取得了一定的成果。

1.2.1 传统时空数据研究与发展

时空数据库(Spatio-temporal Database, STDB)[5]，指对时空数据进行存储和管理的数据库系统，对于时空对象，可以根据时间属性查找其过去或当前所在的空间位置或者范

围，并且预测其未来的变化[6]。近年来，时空数据库的应用越来越广泛，比如全球定位系统（GPS），LBS，交通控制系统等。但是，传统关系型数据库已经很难承载如此巨大的数据量，在查询效率方面也很低下，为实现对时空数据的高效处理，时空数据库的研究也一直备受关注。一般采用空间数据库来管理空间属性，进而对时间属性进行额外的处理，但是由于时空数据中时间维度的连续性等特点，这种操作很不适用。所以研究学者也在空间数据库的基础上开发了时间数据库管理时间属性。

时空数据由于其时间和空间的连续性，在时空数据库存中，数据会随着时间的延续而不断改变空间位置。随着数据量的增加，在数据管理与数据查询时更加复杂，在不断的更新、检索时空数据时需要消耗更多的资源。为了提高时空数据库的性能，快速访问特定的时空数据，索引机制至关重要。

有一些研究采用传统的分布式数据库作为时空数据库，在解决时空数据库的存储与可扩展性问题上取得了一定的进展，但它们一般是基于关系型数据库，采用不同的划分策略对数据在不同服务器上进行分片。这一方法存在扩展性有限、软件结构复杂、成本较高等不足。也有研究者使用 NoSQL[7]数据库（如 BigTable, HBase, Cassandra）存储时空数据，如 Google Earth 使用 BigTable 存储卫星图像数据[8]，但在数据存储组织上没有考虑地理信息特殊性，同时索引结构也不支持复杂时空查询算法。另一些研究者研究大时空数据存储，希望解决时空数据库的大规模存储问题，如 PIST[9]和 TrajStore[10]这两个基于大时空数据存储研究的时空数据库，他们提出的是在非分布式存储环境下的时空数据分析，并不能适用于海量时空数据的存储与分析。

时空数据相关技术研究中一个非常重要的部分就是数据的索引和查询。虽然对时空数据各类查询问题的研究已经有了较长时间的历史，但是随着各种新型数据、应用的不断涌现，时空数据的不断膨胀，时空数据定义格式和常用的查询算法也不能有效的适用于如此庞大的时空数据。索引的研究非常关键，其中尤以空间索引技术的研究更为广泛。如经典的空间索引 R-tree[11]，它由 B-tree[12]扩展而来，是一个高度平衡的树结构，能够有效的对空间数据进行处理。在 R-tree 基础上，又提出了众多改进，如 R*-tree[13]，Beckman 等人对 R 树的插入算法和删除算法进行改进，使 R 树的效果更好；Singh S[14]提出了基于倒排索引和基于 R 树的混合索引方法；Walid G 等人[15]总结了近几年获取时空数据的方法，其中介绍了 R^{PPF} 树[16]，R^{PPF} 树可以有效的获取过去、当前、和预测未来

位置的时空数据。以上的研究只是在单机上建立索引和查询，并没有考虑如何在分布式环境中使用，受限于单机性能的瓶颈与成本的增加，如何改进传统时空索引以适应新的技术需求是亟待解决的问题。

1.2.2 分布式并行索引的发展

云计算技术的快速发展为处理海量数据提供了很好的技术支持。云计算技术的高可用性、高可扩展、低成本、按需配置等良好特性，符合高标准时空数据库相关技术要求。因此，研究一种以云计算技术为基础的时空数据库，是本项目所要研究的内容，也是一项十分有意义和前景的工作。Hadoop[17]是一个以 Google 公司的系统为蓝图的开源项目，是一个卓越的云平台，包括 Yahoo、百度、阿里巴巴、华为等众多公司都在其上构建自己的云平台。Hadoop 系统中的计算节点可以从几个到几万个，具有极好的可扩展性，可使用廉价的 PC 机为计算节点，部署成本较低。这些特性更好的适用于构建分布式时空数据库，但在云平台上如何高效地存储和管理海量数据，如何快速有效的处理各种查询，都需要深入的研究。基于 Map/Reduce 的云计算，有着廉价、高容错性和高可扩展性等优势，在企业中被广泛的应用[18]。Map/Reduce 计算框架的基本思想是将问题进行合理的划分，使其能分解成为可以并行处理的问题。在时空数据的处理上，CloST[19]提出一种基于 Hadoop 的分布式时空数据存储系统，该系统根据时空数据，分为三层目录，ID 属性，时间属性，空间属性，依次构建索引目录，但是在基于空间范围查询时并没有采用高效的 R 树索引，必须依次扫描文件记录。所以，这激发我们研究分布式下的时空索引机制，采用 R 树及其变种，构建一种分布式的 R 树索引，设计出高效的，可扩展的，支持大时空数据存储与检索的存储系统。

同时，有研究者就某些特定的时空数据，在查询操作中如何使用 Map/Reduce 范式进行加速的方法进行了一些研究。Ariel Cary[20]等人在 MapReduce 框架下创建了 R-tree 索引并行化的一般方法，即将空间数据划分成不同的分区，并在每个分区中创建单独的 R-tree 索引，最后将所有索引合并成一个 R-tree。Yunqin Zhong[21]等人研究了一个分布式组合时空索引方案。Zhong Y 等人[22]研究了在 Hadoop 下进行时空查询的模式和方法，对地理数据进行聚合，同时采用了两层索引结构，可以有效的进行数据查询。但这些研究并没有在 HDFS 层次进行索引的构建，具有很大的限制。exHDFS [23]系统中，基于 HDFS，设计了时空数据的划分方法，将时空数据均匀分布到集群，时空数据在磁盘上

进行重组，采用分布式缓存机制，提高检索效率，但是该系统的检索方式单一，并且数据压缩和网络传输方面可以进一步优化。

Ahmed Eldawy[24]等人研究出 SpatialHadoop，一个分布式空间存储与高效检索系统，该系统在 HDFS 层次上进行修改，对空间数据构建 Grid，R-tree，R+-tree 索引，具有开创性意义。但是系统只是针对空间数据，不能有效的用于处理时空数据。同时，Ahmed Eldawy 等人又在此基础上提出 SHAHED[25]，一个用于 NASA 存储和处理卫星数据的分布式时空系统，该系统按照年、月、日对卫星数据的时间属性进行划分，再以空间属性划分数据，最后在每个数据块中构建 R-tree 索引等，检索时先对时间索引进行剪枝，然后通过检索空间索引结构缩小空间范围，降低查询规模，但是该系统对数据的选用有很大的限制性，不具有通用性，同时在对象随着时间变化移动方面不能进行处理。

1.3 本文研究内容及贡献

本文研究目标是构建分布式时空数据库的关键技术，分为分布式存储技术、分布式索引技术、以及基于 Map/Reduce 范式的分布式计算算法。为在云计算架构下的高可用性、高扩展性的分布式时空数据库的实现提供基础性的理论和方法。

对目前时空数据库相关技术的研究现状以及应用的驱动，研究高效的时空数据库已经成为当前科学与技术领域的热点研究问题，能否研究出相关技术来解决具有可扩展性、高并发性、支持高效查询与检索的时空数据库是一项十分有意义的研究工作。因此，本项目拟构建一个以 HDFS(Hadoop Distributed File System)为存储载体、多维 R-tree、Map/Reduce 为时空查询算法范式的时空数据库及其相关技术，促进时空数据库处理与分析领域的研究与发展。

1.3.1 低成本、低能耗、高扩展性的 Hadoop 云平台

随着云计算的发展，为海量时空数据的存储和研究提供了技术上的支持，但是不论是对于企业还是学术研究者而言，云平台巨大成本都是不可忽视的问题。无论是在单机硬件价格，能耗，或者巨大的占地面积，消耗都是巨大的，对普通用户的收费也是昂贵的。对于学生而言，已经不能满足单机伪分布式的实验研究，而分布式集群的成本又是学生不能负担的。

所以，针对这一问题，通过大量的准备工作，我们决定采用一款 A20 处理器的 ARM 开发版，Cubieboard2，搭建 Hadoop 分布式云平台。通过实验进行分析，虽然 Cubieboard2 单机性能低下，但是该云平台主要有以下优势：（1）云平台充分发挥了可扩展性的优势，随着节点的增加，提高了整体性能。（2）对于大规模时空数据，该云平台具有不错容错性，支持多个客户端并发访问，能处理较复杂的查询处理操作。（3）云平台的成本低，并且能耗少，体现了绿云[26]的概念。可以为回收二手手机搭建分布式云平台提供借鉴。

1.3.2 两种分布式时空索引机制

SpatialHadoop[23]在 HDFS 层次上，构建了全局和局部索引，用于高效的处理空间数据，但是对于时空数据有很大的限制，无论是数据的划分策略还是全局-局部索引机制，都不直接用于时空数据。所以本文针对传统时空索引的研究和分布式集群的特性，也设计了全局-局部索引机制，主要有以下三种贡献：（1）为适用于不同分布的时空数据设计了两种新时空划分算法。（2）在局部索引阶段，通过构建一维时间索引和多维 R-tree 索引，进一步提高查询效率。（3）优化存储结构，减少磁盘 IO，提高网速传输效率。

对于海量时空数据，数据划分策略至关重要，因为这样可以将时间与空间上相邻的时空对象聚集在一起存储，有利于提高检索效率。本文设计的两种划分策略，一种针对分布相对均匀的时空数据，采用时空网格划分策略，在时间与空间上将时空对象进行划分。另外一种是基于时间的改进四叉树划分策略，用于处理分布相对不均的时空数据。这样做的优势非常明显，不但将相邻的时空对象聚集存储，而且在检索时，可以先进行时间和空间上的粗略剪枝，迅速定位到要查找的区域。而在局部索引阶段，为了进一步缩小查询范围，本文分别设计了一维时间索引和 3DR-tree[27]索引，前者能快速的定位到查询的时间区间，然后再进行空间上的范围查询。而后者则是在 R-tree 基础上，添加了时间维度，将 R 树的最小外包矩形提升为最小外包长方体。由于本文搭建的云平台硬件上的限制，单机性能低，网络传输速度慢，所以，为了更适应该平台，我们采用列存储和压缩算法，将时空数据压缩处理，大大减小了数据量，提高磁盘利用率和网络传输效率。

1.4 本文组织结构

本文共有六章，每章的主要内容如下所示：

第一章 主要讲述了本文的课题背景和研究意义，国内外研究现状和主要研究内容及贡献。

第二章 介绍本文中用到的 Hadoop 相关知识，并主要讲解了当前分布式时空索引相关内容。

第三章 在成本、能耗、性能上对本文搭建的云平台进行分析，为后文时空数据索引的研究打下铺垫。

第四章 主要介绍改进的两种索引机制，将时空数据合理的划分到集群节点中，防止数据倾斜，高效的管理时空数据。

第五章 优化时空数据存储结构，减少磁盘 IO，提高网络传输效率，并通过实验分析本文设计的索引性能。

第六章 总结与展望。

第二章 相关知识介绍

在本章中，主要介绍本文研究中用到的相关技术与理论知识。传统的关系型数据库不能适用于存储海量时空数据，本文采用 Hadoop 搭建的云平台，研究分布式时空索引，实现时空数据的有效存储与管理。首先介绍云平台相关知识，详细讲解 Hadoop 中的 HDFS 分布式文件系统和 MapReduce 并行编程框架；然后针对当前流行的空间和时空的分布式并行索引，介绍 SpatialHadoop 空间系统模型；最后介绍本文的研究工作。

2.1 云平台的相关知识

云计算是在传统单机处理模式上的一个质的飞跃，在分布式计算机集群上并行计算，利用网络，合理分配计算资源和服务。目前，云平台不断涌现，比较有名的云平台有亚马逊 EC2，google 的 App Engine, Abiquo 公司的 AbiCloud 等等。本文采用的 Hadoop，一个模仿 Google 体系架构的开源项目，能在分布式环境下高效的处理大规模数据。Hadoop 主要包含两个核心的内容：HDFS 分布式文件系统和 MapReduce 分布式计算模型。基于 Hadoop 搭建的云平台可以部署到廉价的机器上，所以，本文中，采用了 Cubieboard2 的开发板，充分体现云平台的可扩展性、容错性、低成本、低能耗等特性。

2.1.1 Hadoop 发展

近年来，诸多企业积累了大量的数据，在存储和处理这些数据上面临巨大的挑战，如何挖掘这些隐藏的财富是提升企业竞争力的有效手段。但是单机 CPU 和内存的硬件发展速度不能适用于海量数据。Hadoop，这个开源的分布式计算平台，冲破了海量数据的限制。Hadoop 主要有以下两个组件：HDFS（Hadoop Distributed File System）分布式文件系统和 MapReduce 计算模型。可以非常高效的处理大规模数据，用户不用知道底层分布式系统是如何运作的，进行程序开发。从 2003 年到 2004，Google 发表 GFS[28]，MapReduce[29] 著名的论文，揭开了分布式处理的篇章。2005 年，Nutch 开发出 MapReduce 应用，为后来的发展奠定基础。2007 年，百度开始使用 Hadoop 用于处理离线数据。2008 年，Apache 将 Hadoop 作为一项顶级的项目进行研究，并在这一年在 TB 级数据排序记录中创造历史。2009 年，由 Cloudera 推出了 CDH，将 Hadoop 推向用于商业化。并继

续刷新 TB 级排序记录。目前，Hadoop 已经发展成了多个子项目，HDFS，MapReduce，HBase，Hive，Pig，Zookeeper 等，而最新的 Hadoop 版本中引入 Yarn 框架，将 Hadoop 推向新的高度，更加适用于当前的发展趋势。

2.1.2 HDFS 原理研究

Hadoop 分布式文件系统（HDFS），被设计成可以运行在大量硬件设备上的分布式文件系统，HDFS 具有高度容错性。HDFS 采用主从架构模型。主要包括三类节点，NameNode（主节点）, DataNode（从节点）和 SecondNameNode。如图 2.1 所示。

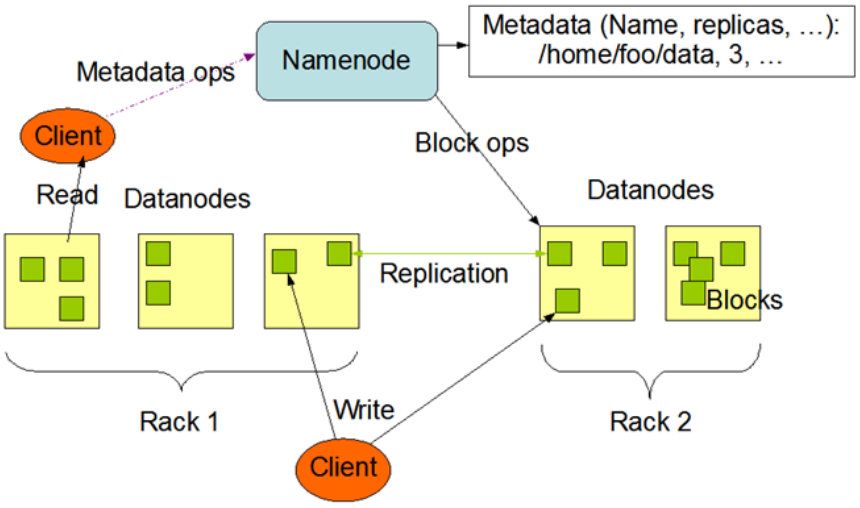


图 2.1 HDFS 结构图

（1）NameNode: 这是主节点，也是整个集群的管理中心，用于管理和维护分布式文件系统的命名空间、集群配置、元数据信息等。管理文件和数据块（Block，默认大小 64M）的对应关系，数据块与 DataNode 的信息等。

（2）DataNode：是文件存储的的基本单位，是 HDFS 的从节点，也称数据节点，通常 HDFS 包含多个 DataNode。数据块就存储在 DataNode 中，并且数据块通过复制（默认 3 份）分布到不同的 DataNode 中。在集群中，可以动态的添加节点，增强集群的性能。

（3）SecondeNameNode: 该节点主要作为 NameNode 的备用节点，周期性地更新 NameNode 节点的数据信息，当分布式集群 NameNode 出现故障时，该节点用于数据的恢复，保障系统的稳定性。

2.1.3 MapReduce 编程模型

Hadoop 实现了 MapReduce 分布式编程模型，可以在大规模集群上进行复杂的并行计算。与 HDFS 相同，MapReduce 也设计为主从模型，即 Master/Slave，通过运行 Map-Reduce 实现系统来管理多个大规模计算过程，并且同时能够保障对硬件故障的容错性。Map-Reduce 分布式计算框架主要采用 Map 和 Reduce 方法，以 Key-Value 键值对的形式进行数据传输。作业由一个 JobTracker 和多个 TaskTracker 执行，其中 JobTracker 运行在主节点，是作业的管理中心，负责调度每个任务（task），使 TaskTracker 执行相应的任务。TaskTracker 则是运行在从节点上，与 JobTracker 通过心跳机制联系。

如图 2.2，JobTracker 创建多个 Map 任务和 Reduce 任务，数量可以在程序中手动设定，这些任务分配给 TaskTracker，为减少网络的数据传输量，任务会就近分配到距离输入分片近的节点。首先执行 Map 任务，对于输入分片的每条记录，执行 map 方法后，以 Key-Value 键值对的形式，先输出到内存的缓存中，数据一般按照 Key 值排序，分成不同的组，隔一定时间段写入本地磁盘。而 Reduce 任务将处理这些中间结果，通过网络读取数据，通过 reduce 方法进一步处理之后，输出到 HDFS。

对于输入文件，hadoop 首先由 FileSpliter 将文件分割成 n 份，每份文件有一个 map 任务执行，数量由 mapreduce 程序指定，由 salves nodes 限制。然后每一个分割文件通过 RecordReader 将文件每条记录转换成 K-V 键值对，将这些记录传送给 Map 方法。

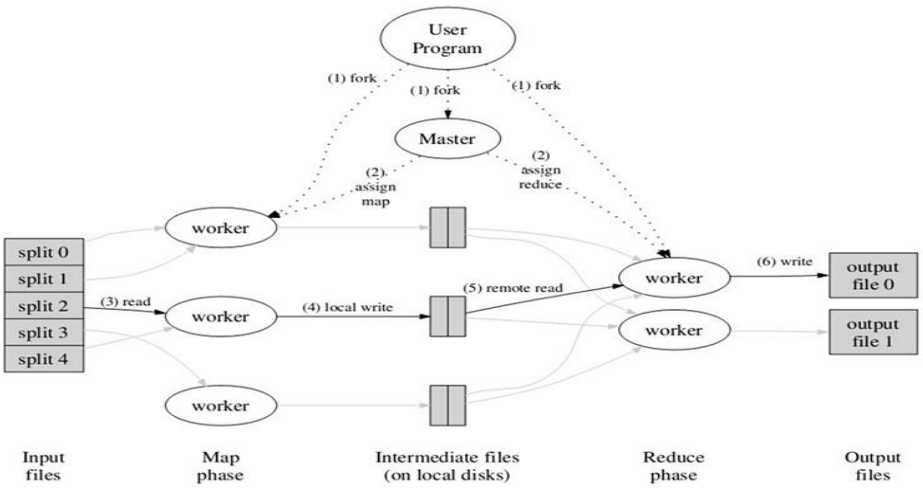


图 2.2 MapReduce 分布式编程模型

2.1.4 Yarn 模型

从 2004 年 Google 发表关于 MapReduce 的论文，经历长达十几年的发展和研究，Hadoop 已经渐趋成熟，Hadoop 也从 MRV1 发展到 MRV2。而 MRV2 中 Yarn 是将 JobTracker 的集群资源管理和作业管理进行分离，不再是由 JobTracker 单独管理，尤其是对于节点性能低的机器，将大大提高集群管理的性能。Yarn 中，资源管理器（ResourceManager）运行在主节点。节点管理器（NodeMagager）运行在从节点。ResourceManager 中分成两个组件，应用程序管理器（ApplicationManager）负责集群中资源的管理和调度，调度器（Scheduler）则根据各个应用的资源需求和集群各个节点 Container（资源的分配单位，将内存、CPU、磁盘等资源抽象封装的概念）进行调度。NodeMagager 负责启动应用程序，监控容器 Container 的使用情况，并与调度器进行通信。

2.2 分布式时空索引理论

2.2.1 时空对象

我们所在的生存空间，每个对象都包括空间属性和时间属性，空间属性表示物体存在的位置，而时间属性则是表示物体存在的周期，从开始诞生，到结束，或者是物体运动的轨迹。每个对象都可以用坐标来表示，在地学中，一般采用经度和纬度来表示空间属性，用时间表示时间维度，即（x, y, z）,其中，x 表示经度，y 表示维度，z 表示时间，物体的位置会伴随着时间的变化而不断变化，称之为时空数据。天空中飘动的云彩，航行在天空的飞机，翩翩飞舞的蝴蝶，水中游动的鱼儿，狂风暴雨下的台风，流感病毒的扩散，疾驰的汽车等等。

时空对象的运动轨迹可以分成离散型和连续型。前者的变化特征是，空间随着时间点而变动，往往时间点之间间隔大；连续形式的变化是空间随着时间变化而不断发生改变，往往形成运动轨迹。随着大数据信息的剧增，无论是离散还是连续形式的时空数据，都在不断累积，形成海量时空数据，传统的时空数据存储与处理办法受到很大的限制，所以在研究时空数据时，索引的建立是一种很好的方式，而分布式时空索引的研究目前还很少。

2.2.2 时空索引

传统的 B-tree 索引、 R-tree 索引等被广泛用于关系型数据库。对于处理空间数据，网格索引，四叉树索引，R-tree，R+-tree 等索引显示出强大的优势，在 R-tree 基础上发展而来的时空索引，如 3DR-tree，MR-trees[30]，HR-trees[31]，TPR-tree[32]等，优缺点如下：

（1）3DR-tree 基于 R-tree 创建的时空索引，是将时间属性当做空间的另一个维度，用于处理过去时的时空对象，非常适合时间段的查询，索引结构相对简单。但是对于生命周期较长或者空间位置变化大的时空对象，出现大量交叉区域，查询效率不高。

（2）HR-tree 将时间单独处理，每一个时间段内构建一个单独的 R-tree，对于连续的两个 R-tree，如果有节点相同，则保留其中一个。该索引适合时间点的查询，但是占用大量的存储空间。

（3）MR-tree 与 HR-tree 相似，也是重叠的方式。为了减少每个时间段内 R-tree 的存储空间，每个 R-tree 的节点可以指向其他 R-tree 中保持不变的节点来节省存储空间，对时间片查询有效，但是可能会有许多重复的节点。

（4）TPR-tree 能够很好的处理将来查询的时空索引。定义基于边界矩形的时间函数，很好的支持时间点、现在和将来的时空对象查询，但是对于过去时的数据查询有限。

在添加时间维度之后，索引的结构也就更加复杂，但他们的基础都是 R-tree。R-tree 索引是 B 树的一个多维空间扩展，在 20 世纪 90 年代由 Guttman 提出的，广泛应用于空间数据库。R-tree 很好的解决了高维空间搜索问题，是一颗平衡树结构，采用了 B 树的分割思想，在添加、删除操作时采用合并、分解节点的方法，保证树的平衡性。每个 R-tree 的非叶节点结构（MBR, Child-Pointer）, MBR 表示包含所有子空间区域的最小外包矩形，Child-Pointer 为指针，指向孩子节点，叶子节点（MBR，空间对象），表示在最小外包矩形范围内的空间对象。图 2.3 是 R-tree 索引结构的示意图，叶结点如 R8，R9，R10 表示 MBR，包括在该空间范围内的所有空间对象，R3 则将 R8，R9, R10 全部包围，形成一个大的矩形范围，R3 与 R4 则由更大的矩形范围 R1 包围，在查询时如果 R1 与查找范围有重合，则检索所有 R1 内的子空间区域，直到到查找到查询范围相交的叶节点，加载叶节点内的所有空间对象，返回符合要求的数据。

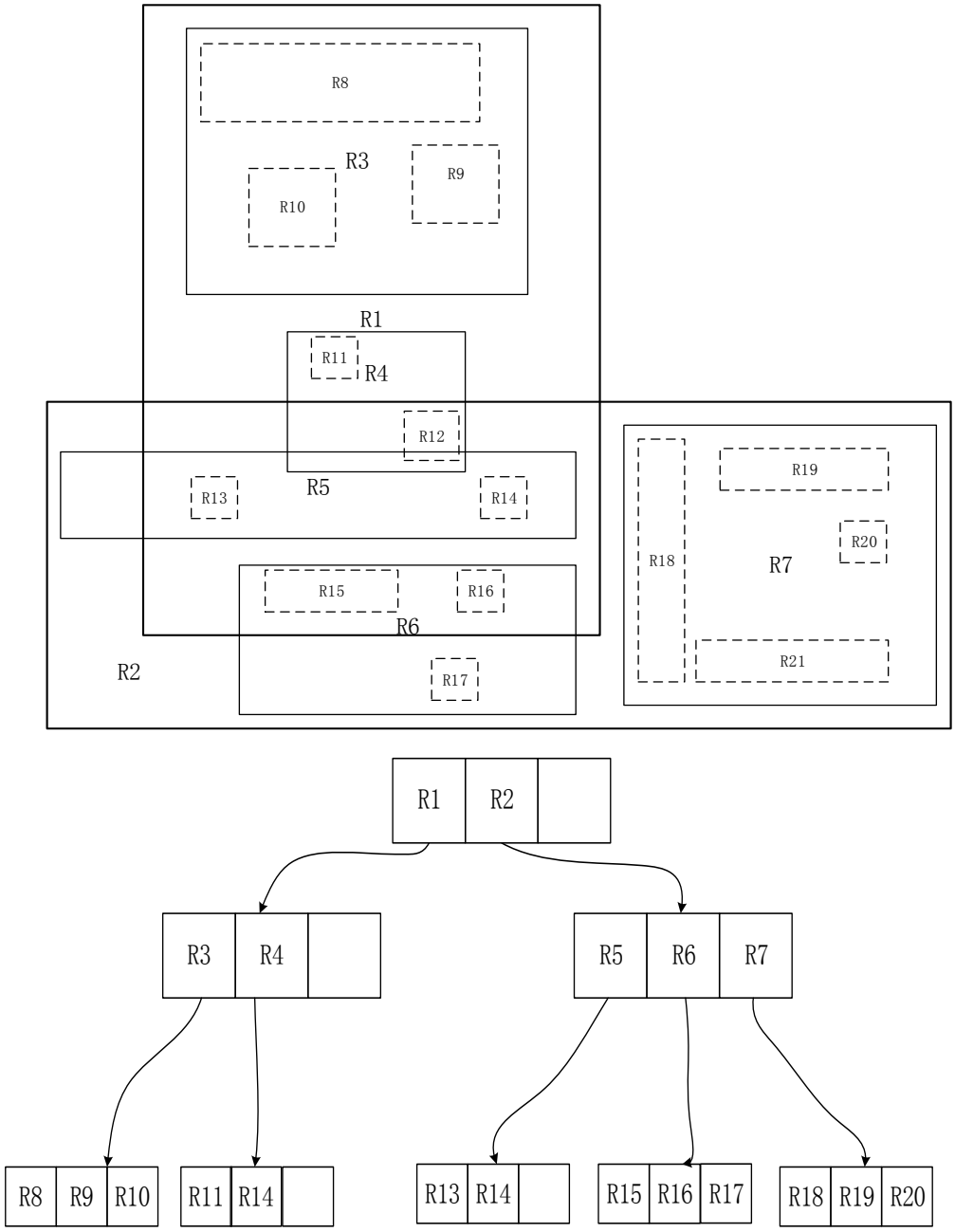


图 2.3 R-tree 索引图

2.2.3 传统 Hadoop 存取时空数据

而在 Hadoop 中，输入的原始文件是没有索引的，他们根据 Hadoop 预定的每个数据块大小（默认 64M），将数据分散存储到数据块中，所以如果查找某个时间和空间范围的记录，就需要扫描所有时空数据的所有数据块内容。如图 2.4，在每个 map 任务中查找到符合查找范围的数据，最终汇总到 reduce 任务，输出查询内容，效率是极其低下的。

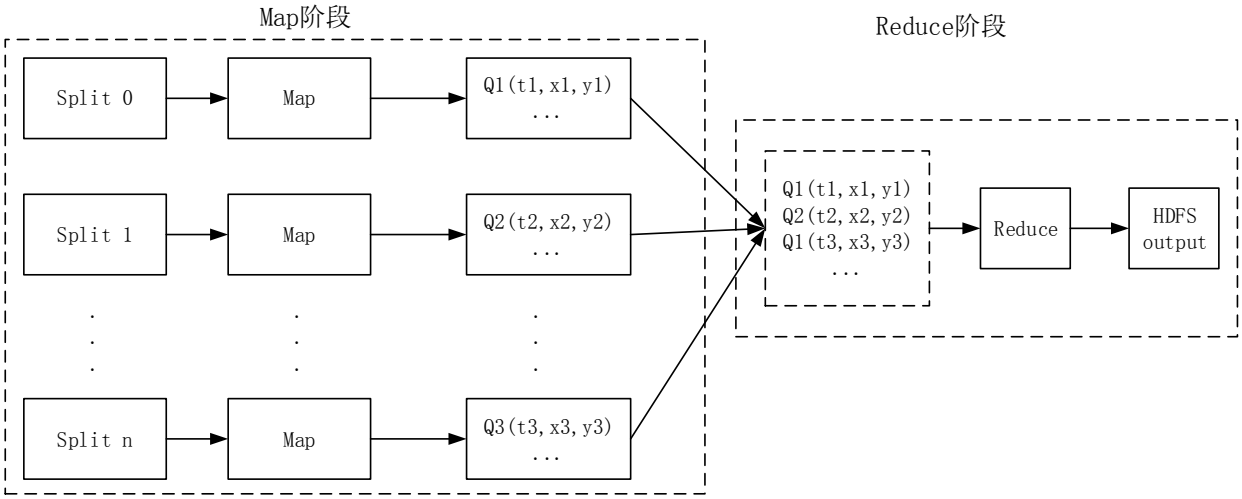


图 2.4 Hadoop 时空数据检索

所以，为了充分利用 Hadoop 的并行计算能力，以及发挥时空索引的优势，我们需要将二者结合起来，构建分布式时空索引。

2.2.4 HDFS 两层索引

许多分布式索引都是基于两层索引构建的，典型的如 SpatialHadoop,在处理空间数据时，按照不同的划分策略，将空间数据划分到不同的数据块中，并在数据块中构建局部索引，如 R-tree、R+-tree，然后将所有局部索引加载到主节点形成全局索引模式。

分布式时空数据的难点在于索引的构建，为了能够高效存储和检索时空数据的记录，克服 hadoop 分布式对索引的限制，本文采用全局-局部索引结构，如图 2.5 所示。该时空索引包括两层，一层称为全局索引，存储在主节点，而数据存储在从节点中，并在每个从节点上构建一个局部索引。这样做有三个优势：

（1）批量加载索引文件并不适用于大数据集，因为这需要消耗大量的内存空间，尤其是该平台节点内存有很大的限制，每台 ARM 机器只有 1G 内存，不能一次性读入所有索引。而我们采用全局-局部索引结构，每个从节点有选择批量加载索引文件，从而加速检索效率。

（2）传统的时空索引，例如 3DR-Tree[33], HR-tree[34], STR-tree[35]索引等等，不能直接用于 Hadoop 中，索引结构适用于单机串行编程，而采用局部索引结构，每个从节点在执行 MapReduce 作业时，局部索引结构互不影响，可以并行执行 map 和 reduce 任务。

（3）全局索引先按照一定的数据分割方式将数据简单划分，所以在每次任务开始时，先读取全局索引，快速定位到数据对象所在的数据块，从而提高效率。

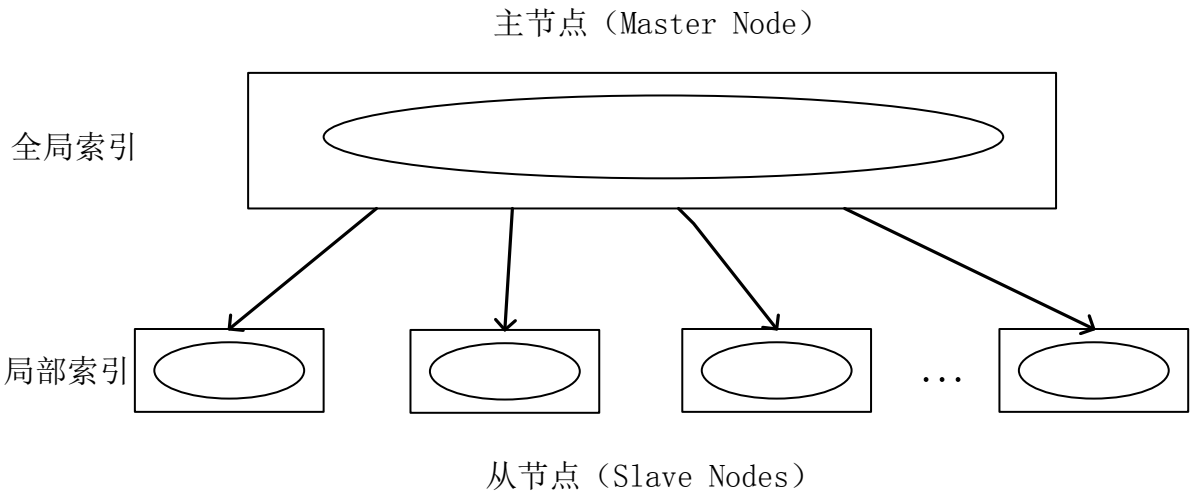


图 2.5 全局-局部索引结构图

SpatialHadoop 中，介绍了两种组件 SpatialFileSplitter 和 SpatialRecordReader。用于处理全局索引和局部索引，对空间范围进行剪枝，并且支持多种查询类型。如图 2.6 所示。在传统的 hadoop 处理基础上对 FileSplitter 和 RecordReader 进行改进。SpatialFileSplitter 读取主节点的全局索引，根据查找的内容，定位到相应的数据块，然后 SpatialRecordReader 提取数据块中的局部索引，转化成（Key, Value）键值对形式，提交给 map 方法。这样通过全局索引首先对空间范围进行检索，缩小查询的数据块，然后通过局部索引在数据块中查找到相应的数据，充分利用了分布式空间索引。

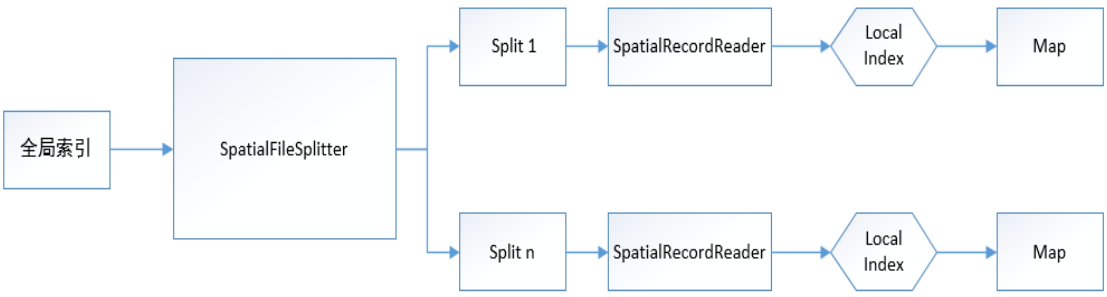


图 2.6 SpatialHadoop 的 Map 阶段

构建两层索引的关键在于数据的均匀划分，如在 MapReduce 框架下构建空间 R-tree 索引[36]的过程中，已经有很多方法，如基于 Hilbert 曲线的空间划分算法，计算空间对象最小外包矩形的 Hilbert 值，然后按照 Hibert 值计算的分区函数，将空间对象划分到不同的分区，并构建局部的 R-tree 索引。而四叉树的划分策略也很常用，但是由于时空属性中时间属性的特殊性，时空对象往往分布不均匀，这就会对时空对象的划分造成很大的影响，空间的数据划分策略有一定的局限性。

2.3本章小结

本章介绍 Hadoop 中的 HDFS 和 MapReduce 两个组件，详细讲解了 MapReduce 读取数据的一般处理过程，为后文中的时空索引机制的构建提供技术支持。并重点讲解了分布式索引结构，传统的时空数据索引并不适用于分布式集群，所以，需要在 HDFS 层次上进行改进，构建新的时空索引，并利用 MapReduce 并发计算模型通过索引有效的处理海量时空数据。

第三章 基于 ARM 的云平台搭建以及性能分析

当前，数据规模日益扩大，大数据已经成为时代的潮流，传统的 DBMS 存储机制已经渐渐无法满足我们的需求，TB 级数据的处理已经非常常见。为了满足海量数据的存储，高并发的读写，以及具备高扩展性、高性能、高容错性和稳定性，分布式文件系统策略已经得到认可。各大厂商的云计算竞争也日趋激烈，如果在不减少服务质量的同时能够最大化的节省成本和能源消耗，将具有非常大的优势。

在本章中，采用一种基于 ARM 的 Cubieboards2 开发板搭建 Hadoop 云平台，通过对单个 Cubieboard2 和整个集群的性能进行实验分析，为后文时空数据的存储及分布式索引的构建做准备。本章将重点介绍 Cubieboard2 的性能，云平台的搭建，云平台性能的测试。该云平台以其低廉的成本、较低的能耗，可以作为在校学习研究之用，同时对于企业也有很好的借鉴意义。

3.1 CubieBoard2 介绍

尽管一些大学已经拥有分布式云平台，但并不是每个学生都有机会接触到这些设备，更别说是在云平台上进行学习和实验。对于学生来说，许多收费的云平台，又是很难负担得起的。所以我们很难确切的理解云平台在处理海量数据时的优势。

同时，在移动互联网高速发展的今天，手机已经成为日常必备工具，我们手机更新换代如此之快，根据统计显示，截止 2015 年底，我国手机用户已达 15.3 亿，每年产生的废弃手机大约有 2 亿部。面对如此巨量的手机垃圾，如何回收和利用将是个难题。鉴于手机的低成本，低能耗，如果我们能够利用二手手机，搭建一个云计算 Hadoop 平台，是一件很有意义的事情。通过查找资料，本文选择了 CubieBoard2 A20 处理器的 Android ARM 板，搭建一个低成本、低能耗、高可扩展性的 Hadoop 云平台，这样，每个学生可以搭建自己的云平台，用于学习和科研。

Cubieboard2 由一个珠海的 Cubietech 团队开发并推出，采用 A20 处理器的开发板，可以运行 Android ，Ubuntu 和其他 Linux 系统。Cubieboard2 的配置信息[37]见表 3.1，硬件结构如图 3.1，Cubieboard2 采用 CortexTM-A7 的双核处理器，且内存 1G，功能更为强大，网卡最大速度为 100Mbps。

表 3.1 CubieBoard2 硬件规格

CPU	AllWinnerTech SoC A20 @1.0 GHz, ARM CortexTM-A7 Dual-Core
GPU	ARM® Mali400MP2, Complies with OpenGL ES 2.0/1.1
Memory	1GB DDR3 @960M
Storage	4GB internal NAND flash, up to 64GB on uSD slot, up to 2T on 2.5 SATA
Power	5VDC input 2A or USB otg input
Networking	10/100 ethernet, optional wifi
USB	Two USB 2.0 HOST, one USB 2.0 OTG

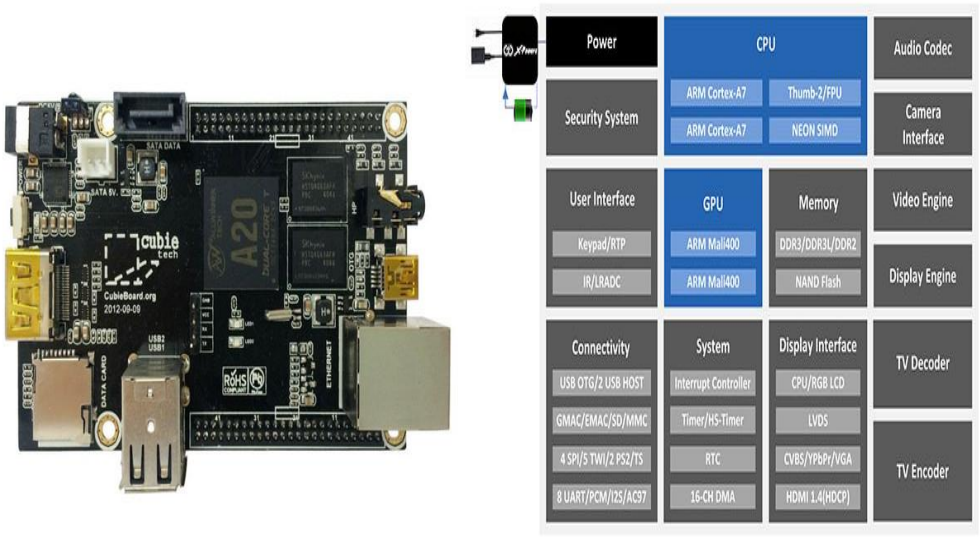


图 3.1 CubieBoard2 硬件图

3.2 云平台搭建

3.2.1 云平台硬件设备

往常我们搭建一个完全分布式 Hadoop 系统，需要多台 PC 机，对于学生而言，硬件资源相对紧缺，我们不得不搭建一个本地单机版的 Hadoop，或者采用虚拟机在本地实现多节点，但是性能较低，运行不流畅。本文采用的 Cubieboard2 搭建的云平台不但成本低廉，而且在性能上非常可观。在硬件设备上，我们主要采购如表 3.2，Cubieboard2

单个价格在 310 元左右，硬盘购买的是希捷 ST500LM021，500G 容量，转速 7200RPM，32MB 缓存，受限于 Cubieboard2 的网卡，采用百兆带宽的交换机。该集群的总成本在两万元左右，仅仅是两台 PC 机的价格，所以在成本的优势非常明显。

表 3.2 硬件采购成本

硬件	数量（个）	成本（元）
Cubieboard2	30	9300
希捷 ST500LM021	30	9400
交换机、路由器	4	700
其他（插排、网线、数据线等）	若干	960

3.2.2 搭建过程

对于每一个 Cubieboard2，我们安装 lubuntu 系统，外接 500G 硬盘，搭建结构如图 3.2 所示，采用 30 台 Cubieboard2 ARM 板，五个一组，每三组一个机架，构建基本的云平台的架构。

图 3.3 介绍了云平台架构图，每台机器的主机名分别为 cloud001~cloud030，cloud001 为主节点。cloud001~cloud015 为一个机架，cloud016~cloud030 作为另一个机架，网络采用百兆带宽的交换机。

云平台系统搭建的主要步骤简单介绍如下，搭建过程均通过 shell 脚本编程执行。

（1）查看路由器表，记录 DHCP 协议为集群动态分配的 ip 地址，每一组进行编号。

（2）ssh 远程登陆协议登陆到每个 cubieboard 的 lubuntu 系统，配置静态 ip 地址，范围：192.168.1.11~192.168.1.40。

（3）配置 hostname 和 hosts。分别设置各个节点名 hostname 为：cloud001~cloud030。hosts 文件用于存储集群中各个节点信息，负责将节点名映射到相应的 ip 地址。

（4）配置 ssh 无密码登陆，安装 pdsh，pdcp，用于管理云平台，其中 pdsh 用于批量执行命令，pdcp 批量在集群中上传文件。

（5）安装 jdk1.7（ARM），hadoop1.2.1，hive，zookeeper 等。

图 3.2 云平台搭建实体图

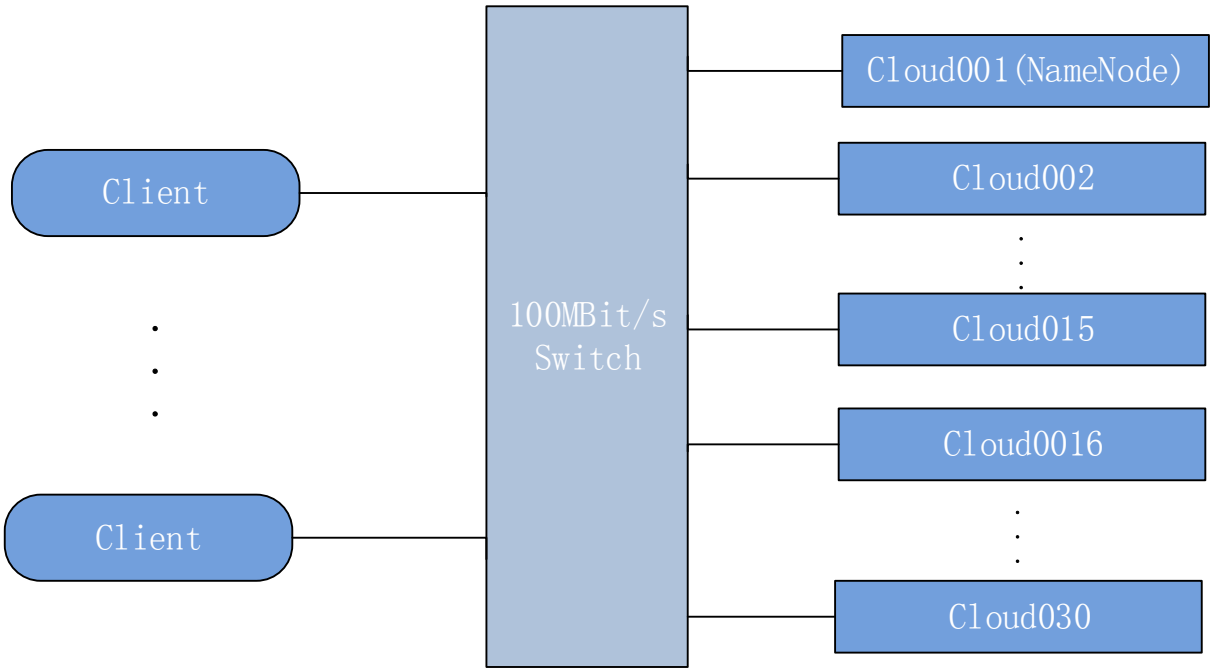


图 3.3 云平台架构图

3.3 云平台性能测试方法

在处理大数据方面，云平台显示出了巨大优势，但是我们不得不承认现有的云平台在成本和能耗上还不具备优势，所以我们采用低成本的 CubieBoard2 搭建云平台。由于 Cubieboard2 性能较低，与手机类似，甚至性能低于手机性能，所以将来我们同样可以大量回收二手手机，搭建一个云平台，为科研服务。

3.3.1 Cubieboard2 性能分析工具

在本小节中，对单个 Cubieboard2 性能分析对比，采用的性能分析工具：UnixBenchmark（Version 5.1.3），UnixBench 是一种测试 Unix 系统机器性能最为经典的工具之一，它提供了一套专门为 Unix 系统检测的指标，对每个检测的项目进行分析，计算出一个分数，最终将所有分数汇总得出一个综合评分，通过这个评分，可以非常清晰的比较出该系统的性能优劣。UnixBench 的检测项目主要集中在以下几个方面：包括系统调用、读写、进程、2D、3D、管道吞吐量、CPU 浮点运算等系统性能，由于本集群不用图形图像处理，所以 2D 和 3D 测试项目省略。

Dhrystone 测试，测试侧重在对字符串的处理，不包括浮点算数运算的操作。这个测试用于测试链接器编译、代码优化、内存缓存、等待状态、整数数据类型等，硬件和软件设计都会对测试结果有很大影响。

Whetstone 测试，该测试主要是对浮点运算进行处理，测试其性能。通过几个典型的测试项目，包含浮点、整数等数学计算。

File Copy 测试，主要是对文件的读写性能进行测试，包含文件传输，并使用大量的缓存。用于检测的标准是在规定时间（默认 10s）内写入、读取文件的字符数量。

Pipe Throughput（管道吞吐）测试，Pipe 用于进程之间的简单通讯。管道吞吐测试的指标是每秒钟单个进程写入一个管道 512bit 数据，并记录读取的次数。

Process Creation(进程创建)测试，该项测试中，操作系统将创建大量的进程，通过单个进程产生相应子线程，并且能立即退出的次数来衡量其性能。

该工具能对操作系统的检测项目进行评分，最后得出一个综合评分，评分越高，则其性能也就越好。

3.3.3 云平台性能分析方法

在本文中，我们采用了两种典型的 benchmark 测试方式来检测云平台的性能，并测试云平台的能耗。

（1）采用 TestDFSIO 测试 HDFS 分布式文件的读写性能，创建一个 MapReduce 作业，并发的进行读操作或者写操作，其中，MapReduce 作业分为 Map 任务和 reduce 任务，map 任务用于读或者写每个文件，该任务的输出汇总到 reduce，进行数据的整合与处理，最终统计出结果。实验中我们设置每次读写的文件大小都为 1024MB，测试并发文件数即 map 数分别为 3,6,12。

（2）TeraSort，对输入的数据，通过 map 和 reduce 函数，按照 key 值进行排序，并输出到 HDFS 文件系统。首先我们采用 TeraGen 生成随机数据，然后运行 TeraSort 指令，对生成的随机数据进行排序并输出，最后 TeraValidate 指令对输出结果进行验证。例如 Teragen 每行数据的大小是 100B。要产生 1T 的数据，需要的行数 =102410241024*1024/100=10995116277 行，在本次实验中，我们生成的随机数据为 30G 的文件数据，作为 TeraSort 的输入。测试过程中，本文采用 IBM 推出的 nmon[38]软件监测整个集群，nmon 能在性能测试的过程中实时的记录使用的系统资源，在本文中设置每隔 30 秒捕捉一次系统资源的数据，最终输出到指定的文件中， 采用 nmonanalyser 工具分析这些文件。nmon 不会占用过多的系统资源，CPU 使用率低于 2%。很适合该云平台。

（3）对于云平台，能耗是一项非常大的支出，所以，对于能耗的检测非常重要，并想办法在软件和硬件角度降低能耗。在本文云平台中，耗电量主要集中在单个节点，而每个节点能耗包括两部分：Cubieboard 和硬盘。通过购买计量插排插座，每个接口测量一组 Cubieboard，一次性测量三组，15 个 CubieBoard 和硬盘的功率。

3.4 实验及分析

在该实验中，采用上文中介绍的云平台进行测试，节点为 30 个 Cubieboard2 的 ARM 板，分为 28 个数据节点，主节点和辅助节点。Hadoop 版本为 1.2.1，HDFS 数据块大小采用默认 64M，每个数据块的副本参数为 3。同时，设置 3 个额外的客户端节点，连接该集群。

3.4.1 单节点性能测试与集群能耗测试

图 3.4 为 UnixBench 测试工具对三种不同机器的实验结果对比图。其中，Vmware Ubuntu 是 Intel(R) Core(TM) i3-2120 CPU @ 3.30GHzx 下 Vmware 虚拟机下的测试结果，综合评分大约 1000 分。 Aliyun 是在测试的 Intel(R) Xeon(R) 512 内存的阿里云服务器，综合性能也远远好于 Cubieboard2。Cubieboard2 在管道吞吐量、进程创建、文件复制读写等测试项目上评分都很低，而且最后一项综合评分表明其性能也远低于正常 PC 机。

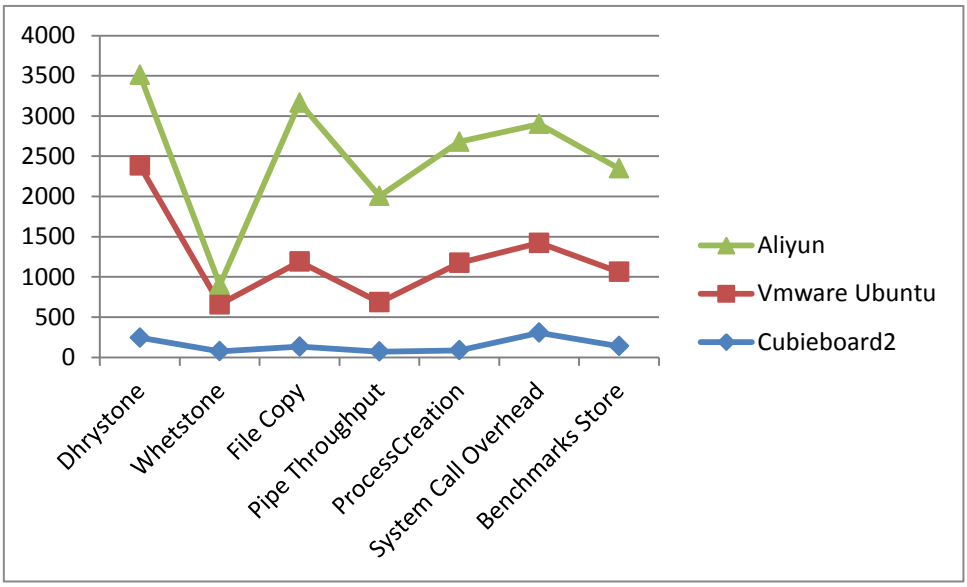


图 3.4 Unixbench 性能检测

3.4.2 读写测试

在上面的测试中，我们发现 Cubieboard2 的单机性能非常低。当我们搭建成云平台之后，不但充分发挥其低成本、低能耗、高扩展性的优势，并能大大提高该云平台的性能。在这一节中，我们采用 Hadoop 自带的两个典型的 benchmark 测试方法来检测该云平台的性能。根据节点的不同，对比分析实验结果。

采用 Hadoop 自带的测试方法:TestDFSIO，分别执行三次读写操作，读写文件数量分别为 3,6,8，文件大小均为 1024MB。 为了使测试更加准确，减少误差的影响，每一个实验测试三次，然后取平均值，结果如下表所示。单个文件吞吐量 MB/s 表示每个文件读写速率的平均值。

表 3.3 3 个文件读写测试结果

测试类型	单个文件吞吐量 MB/s	测试执行时间（s）
写操作	4.88	256.13
读操作	12.21	135.19

表 3.4 6 个文件读写测试结果

测试类型	单个文件吞吐量 MB/s	测试执行时间（s）
写操作	3.43	346.64
读操作	8.89	168.56

表 3.5 10 个文件读写测试结果

测试类型	单个文件吞吐量 MB/s	测试执行时间（s）
写操作	2.57	456.595
读操作	7.59	210.82

由上表我们可以看出，该集群的读写速率慢。hadoop 在存储时采用的是副本策略，即每一个块都会进行复制，默认是三份，所以写一份文件，就会产生两份的网络传输，而读操作，只是读取其中一份的数据即可，所以在读文件时效率较高。Cubieboard2 的读写性能随着文件数量的增加，数据量的增大，读写时间也增加，但是时间并没有成倍增大，因为随着文件数量增加，map 的并发执行增大，所以文件的读写效率增大。

另外我们采用上节介绍的 Vmware 虚拟机搭建 Hadoop 集群，机器数量 5，DataNode 4 个，网络为百兆带宽，在写入 10 个 1G 文件时，单个文件吞吐量为 2.63，大约用时 401 秒。这两个集群基本达到网络带宽的瓶颈。

3.4.3 TeraSort 测试

本文我们采用 Hadoop 自带的测试方法: TeraSort，产生的随机数据大小为 30G，主要验证该平台系统的综合性能。由于 30G 的数据量，TeraSort 可以在排序过程中体现 MapReduce 的运行时性能，同时排序结果写入 HDFS，可以检测磁盘 IO 性能。

我们生成 30G 随机数据，大约耗时 63 分钟，而运行排序数据则耗费时间更长，在 reduce 阶段任务进行到一半之后失败，这是因为默认的 reduce 任务是 1，在 reduce 阶段所有数据规约到一个 reduce 方法，单节点处理能力远远不够。所以我们更改 reduce 任务数量为 20 个。如图 3.5 所示，主节点启动 TeraSort 的 MapReduce 作业，从 21:50 开始，到 22:39 结束，大约用时 49 分钟。主节点进行任务的分配，之后的排序操作主要集中在从节点中，所以在 CPU 和磁盘 IO 方面占用都很小。在图 3.6 和 3.7 中，节点 cloud008 启动 13 个 map 任务，在 22:10 结束所有 map 任务，这个阶段进行数据采样，将数据分为 R（本文 R 的值为 20）个分区，并把所有数据划分到对应的分区。 reduce 阶段包括 shuffle 和 sort ，并把排序好的数据写入 HDFS。这两个阶段的 CPU 利用率几乎达到 100%，并且磁盘 IO 读写频繁，该集群处理能力主要受限于 CPU 性能，所以对于该云平台需要增加 map 和 reduce 任务的数量，提高整个集群的 CPU 利用率，而不是单个节点负载过大。在图 3.8 和 3.9 中，第一阶段的数据传输主要是读取 cloud008 本地数据和通过网络传输接收其它节点的数据，而第二阶段 reduce 任务输出的结果再写入 HDFS, 所以从 22:24~22:27，将排序好的数据通过网络传输到其它节点，而 22:27 开始到任务结束，读取排序好的数据写入 HDFS。我们发现从作业的开始到结束，网络传输占据很大时间，数据传输时间比较长。

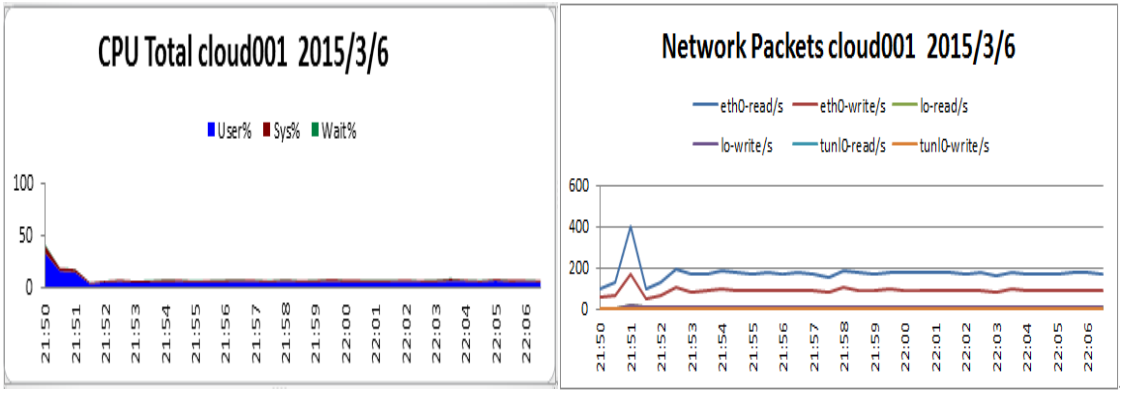


图 3.5 NameNode CPU 和网络数据传输

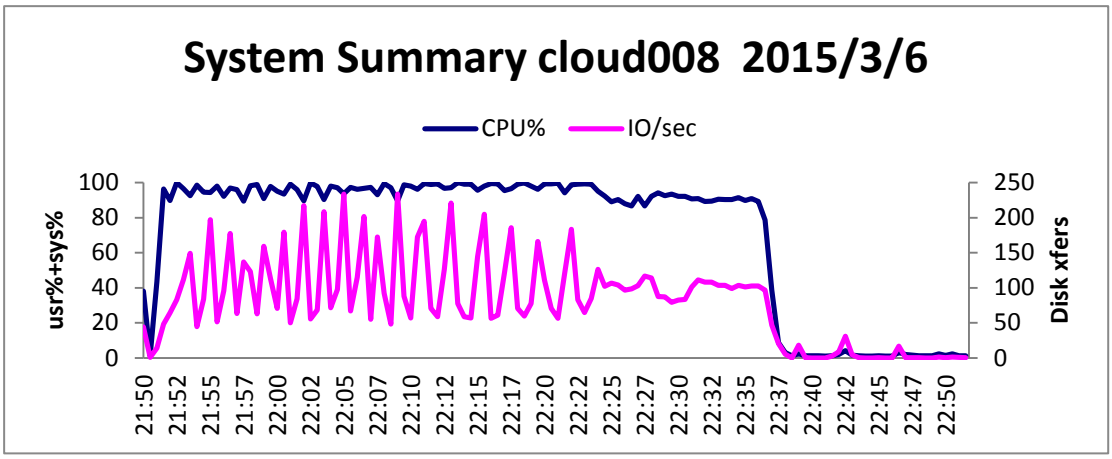


图 3.6 节点 cloud008 CPU 和 IO 使用率

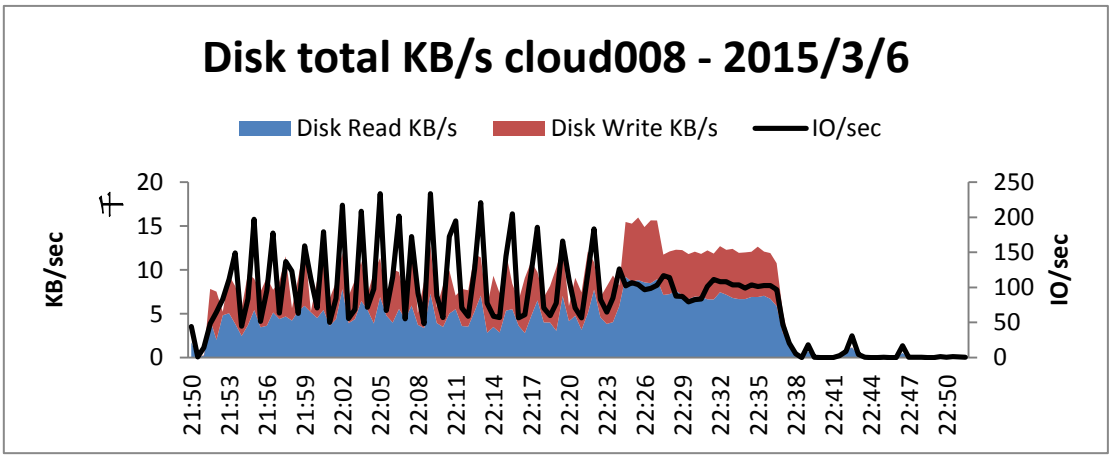


图 3.7 节点 cloud008 磁盘读写速度

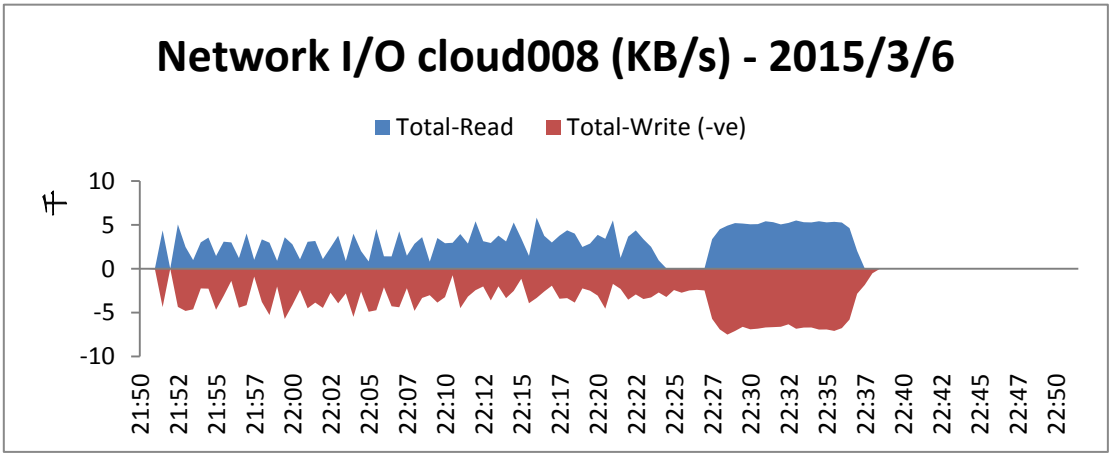


图 3.8 节点 cloud008 整体网络数据传输

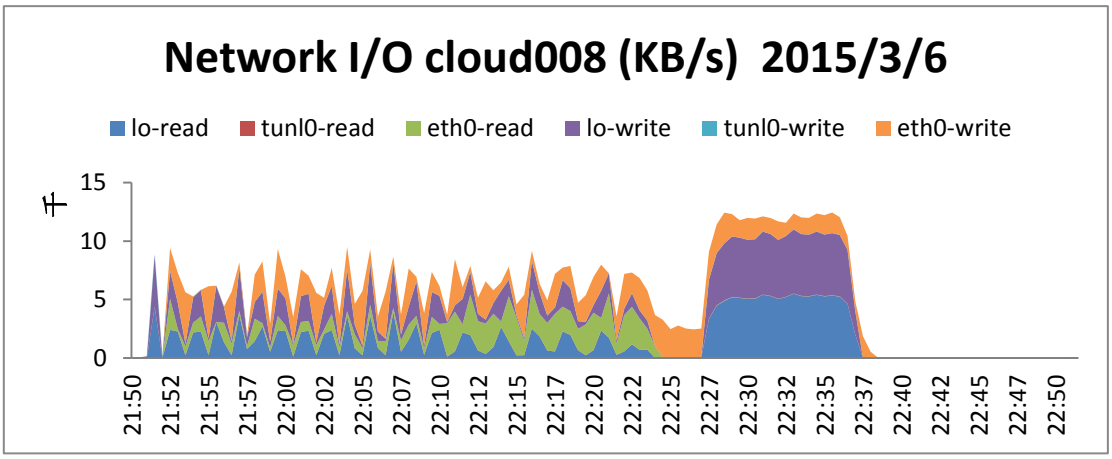


图 3.9 节点 cloud008 详细网络数据传输

通过对 Hadoop 集群中的 TeraSort 性能测试，对于提高作业的运行效率，除了优化 map 和 reduce 作业外，需要考虑到网络传输性能，作业的配置参数等。对于该平台，由于单机性能较低，所以在处理数据时有很大的劣势，但是当我们成功搭建成云平台之后，在性能上大大提高，影响数据处理效率的因素主要在 CPU 性能和网络传输。对云平台性能进行优化时，更需要我们综合考虑，逐项进行改进。

3.4.4 能耗测试

本文采用 15 个节点测试能耗，接入计量插排插座，统计云平台运行时的功率。当 3 个客户端同时并发访问该云平台，每个客户端进行 2G 数据的 TeraSort 排序。如下表 3.6 所示，每个用户分别执行 job1,和 job2，然后计算所有客户端每个作业的平均值，reduce 任务设置为 10 时，job2 的 reduce 阶段执行时间是 job1 的 16.7%。

表 3.6 作业执行时间对比

作业	Map 数量	时间(秒)	Reduce 数量	时间（秒）
Job1	30	62	1	957
Job2	30	59	10	160

在表 3.7 测试了上述实验的 15 个 Cubieboard2 和 15 个硬盘的功率，集群在不启动任何任务，没有任何负载的情况下，功率大约 61.2W。而在高负载，运行密集型作业时，最大功率达到 93.1W，相差 30W 左右，在能耗方面的优势非常明显。所以，搭建成云

平台时，不但使其具有低成本、低能耗的特性，其处理海量数据能力也将大大提升。

表 3.7 15 个 Cubieboards 和硬盘耗电量

测试项目	平均值（W）	最小值(W)	最大值(W)
15 组普通运行	约 61.2	约 60.1	约 62.3
15 组负载运行	约 90.1	约 87.1	约 93.1

3.5 本章小结

本章主要研究分析了当前分布式存储的知识，对于企业研发的各种云平台进行了解，由于成本、能耗的考虑，我们决定采用了 Cubieboard2 这种低廉的 ARM 机器搭建 hadoop 云平台，该云平台具有 Hadoop 的一系列优势，同时具备在处理大数据方面的可行性。随着时空数据的日益增长，采用 MapReduce 架构处理大规模时空数据，已成为当前的一个研究热点。而本文中采用 Cubieboard2 搭建的云平台，不但成本低廉，耗能较少，而且其处理能力也很可观，为后文中时空数据的处理打下基础，同时也可作为高校科研的借鉴。

第四章 分布式时空索引机制

面对大规模的时空数据时，我们可以采用 Hadoop 的 HDFS 进行数据存储，由 MapReduce 实现的开源框架，执行时空数据的分析。然而，仅仅采用这种原有的 Hadoop 框架是不能有效的适用于时空数据处理的，因为在数据检索时需要扫描整个文件系统，效率极其低下，构建分布式时空索引是解决这个问题的关键。构建索引一般分为两个层次，分布式全局索引和局部索引。上一章中，我们已经介绍了搭建的 Hadoop 云平台，并验证其处理大规模数据的能力，下面，我们介绍本文设计的分布式时空索引。重点介绍两种改进的全局-局部索引的结构和相关算法：时空网格索引(简称 TGrid)和四叉树-3DRTree 索引（简称 QDTree）：

（1）TGrid 索引：全局索引按照时间和空间网格算法划分时空数据，局部索引采一维时间索引。

（2）QDTree 索引：全局索引采用改进的四叉树算法划分时空数据，局部索引采用 3DR-tree 索引管理块内的数据。

4.1 分布式时空索引构建方法

4.1.1 时空数据模型

为了支持高效的分布式时空索引，我们定义新的时空数据存储模型，这个数据格式可以很方便的在时间维度和空间维度进行比较，减少操作量。定义一个时空对象 Q(Timestamp, Lon, Lat, Attr1, Attr2, … , AttrN)，表示在特定时间，特定位置的一条记录，其中，Timestamp 表示该记录的时间戳, 后文中用 T 表示，（Lon，Lat）表示空间位置，一般是经纬度，Attr1，Attr2…AttrN 表示该时空对象的其他属性，如 ID，前三个属性为时空对象的核心属性。如图 4.1 所示，长方体 C(A, B)，其中 A(t1, lon1, lat1)，B(t2, lon2,lat2)，表示一个在时间范围 t1~t2，空间范围 (lon1, lat1) ~ (lon2, lat2)，我们称之为长方体（Cubiod）。与传统空间数据表示形式最小外包矩形（MBR）相似，我们称之为最小外包长方体（MBC）。MBC 的定义非常方便后文中的索引的构建，以及时空数据的检索。

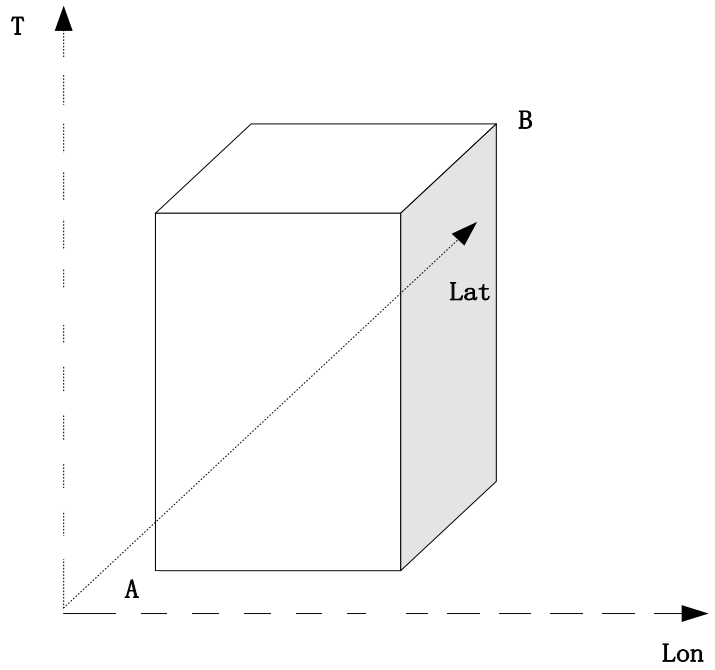


图 4.1 时空坐标下最小外包长方体

由于时空对象复杂多变，数据格式也是复杂不一，所以，本文将统一成上述数据模型，不同的时空数据可能需要先进行数据的预处理操作。由于时空数据的核心属性放置在前三项，其他属性依次排列在后面，在时空索引构建和时空数据检索时方便快捷，而且这种存储方式非常灵活，根据不同的时空对象，在每条记录后面的属性中定义不同的数据格式，甚至，同一种时空对象的每条记录的属性也可以不一样。

4.1.2 两层索引构建一般流程

第二章中介绍了时空数据的两层索引模式，下面我们同样采用两层索引模式，但是我们介绍两种改进的的索引构造方式：

（1）时空网格索引（TGrid）：在全局索引阶段，首先改进传统的网格划分算法，将数据按照时间和空间属性划分到不同分区中，然后在每个分区中按照时间属性进行精细的划分，构建一维时间的局部索引，最终存储到 HDFS 的数据块中，这种方式能有效处理均匀分布的时空数据。

（2）四叉树-3DRTree 索引（QDtree）：另外针对分布不均匀的时空数据，采用改进的四叉树划分算法，将数据划分到不同分区中，在局部索引阶段，我们在 R 树索引基

础上添加时间维度，构建 3DR-tree 索引。

根据两种索引构造方法，在构建时所采取的流程基本一样。如图 4.2 所示，第一步，将所有时空数据按照相应的划分算法，划分到不同的分区中，目标是在同一个最小外包长方体（MBC）内的时空数据划分到一起。第二步，在每一个分区中单独构建局部索引，将索引和时空数据写入 HDFS 的数据块中。第三步，将所有节点的局部索引加载到主节点中，形成目录结构，构成全局索引。

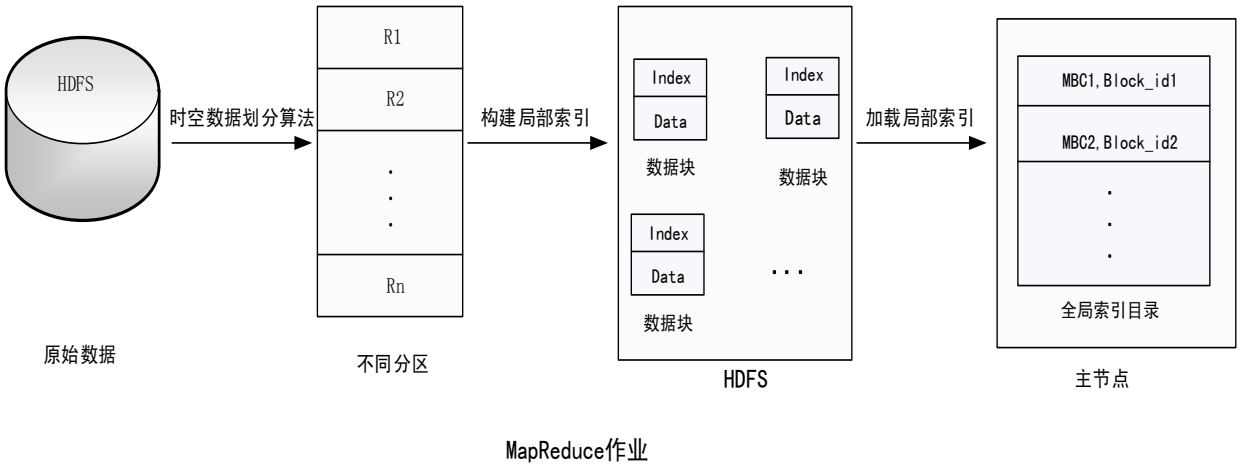


图 4.2 两层索引构建流程

详细描述如下：

（1）加载原始时空数据，存储在 HDFS 上，此时数据存放杂乱无章。

（2）数据划分：

a) 启动 MapReduce 作业，在 Map 阶段，将遍历数据块中所有的时空数据记录，计算出包含这些所有时空数据的最小外包长方体（MBC），记录时空数据总大小。

b) Map 输出的中间键值对：（Key，Value）,Key 表示最小外包长方体,Value 表示该范围内的数据大小，本地可以进行 Combine 操作。

c) Reduce 阶段，对 Map 阶段的输出数据规约操作，计算出包含所有时空数据的最小外包长方体和数据总大小，根据不同的数据划分方法，求出时空数据的逻辑划分标准。

（3）启动 MapReduce 作业，在 Map 任务中，按照不同数据划分算法将数据划分到不同分区中，Reduce 任务，根据阈值判断分区中的数据是否划分到同一个数据块中，若满足，构建局部索引，将索引和时空数据写入 HDFS 中。

（4）最后将所有数据块的局部索引写入一个文件，汇总到主节点，形成全局索引的目录文件。

主节点建立一个基于内存的全局索引，这个全局索引由批量加载构造，并且一直存储在主节点的内存中。一旦主节点失效或者重启，当我们需要使用全局索引时，全局索引就会被重新创建。

本文设计的两种划分算法各有优劣，其中第一种时空网格划分法，更加适用于数据分布相对均匀的时空数据，并且索引构造简单，耗时短，不过当时空数据分布不均匀时，在划分阶段耗时比较长，某些数据块存储的数据量小，出现数据倾斜的问题。第二种改进的四叉树划分算法耗费的时间比较长，但是每个数据块的数据大小基本一致，负载相对均衡，在数据查询时也有很多便利之处。下面，我们详细介绍这两种分布式时空索引。

4.2 时空网格索引(TGrid)

传统的网格索引就是按照空间的长度和宽度划分成不同的子空间单元，不同的空间数据划分到对应的单元中，在 SpatialHadoop[22]中就是采用的这种方法。但是对于时空数据，如果划分标准依然采用这种方式，会因为不同的时间段内数据的聚集程度不一样，造成集群的数据倾斜，单节点负载过重。尤其本文中的云平台单节点性能较低，所以我们改进分割算法：

（1）在数据划分阶段，将时间粗略划分成不同时间区间，每一时间段内进行空间网格计算。

（2）局部索引构建时，将时间划分成更加精细的区间，构建一维时间索引。

4.2.1 改进的网格划分算法

在不同时间段内，时空数据的密集程度不一样，比如早上和下午的上下班时间车流量比较密集，所以，根据不同的时间段划分时空数据有很大的意义。我们目标是把所有时空数据均匀的划分到数据块中，需要知道划分的标准。首先我们计算划分的块数量 N,

满足每个块的实际存储的数据大小都不大于 HDFS 数据块的默认大小（64M），因为这样可以满足：（1）空间相邻性，空间相邻的对象可以被分到相似的分区中（2）负载均衡，所有的分区应该具有相似的大小。（3）在局部索引构建时，有足够的空间存储局部索引。

我们采用如下算法来实现上述目的：

（1）划分时间区间，将时空数据划分到不同的时间段内。默认区间数量为 10。

（2）统计时间区间的最小外包长方体（MBC），采用公式(4.1)计算分区的数量N。

$$N = \lceil \frac{S(1+\alpha)}{B} \rceil \tag{4.1}$$

其中 S 是输入 MBC 的时空数据总大小，B 是阈值，默认为 HDFS 块的设置大小（64M), α 负载因子，一般默认是 0.2 即可，N为分区的数量，这样每个块的实际存储数据容量小于块的默认大小。

（3）计算分区边界。根据分区的数量 N，我们将 MBC 在空间范围进行均匀划分，$\lceil\sqrt{N}\rceil \times \lceil\sqrt{N}\rceil$，每一个单元计算出左下角和右上角的空间坐标。

（4）如果还有时间区间内的空间没有划分，返回步骤（2），否则进入下一步（5）。

（5）根据时间区间和空间网格计算的分区标准，将时空数据划分到相应的分区中。

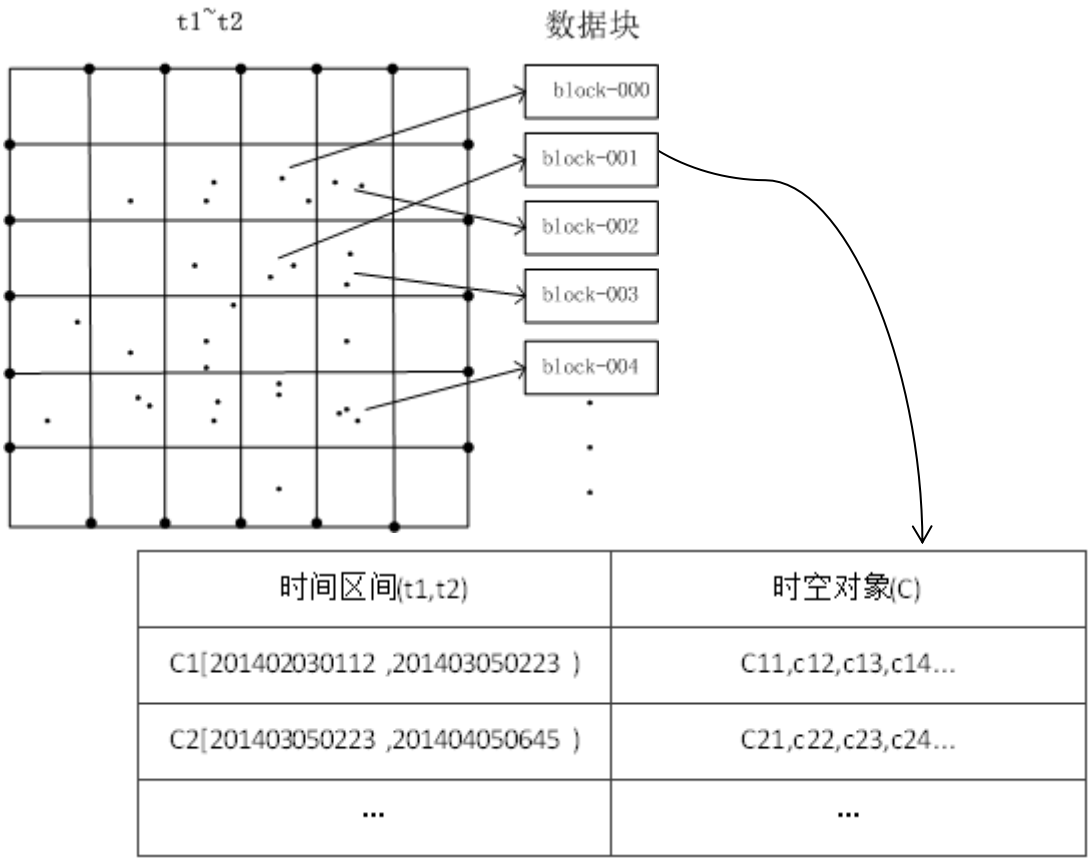
在划分过程中，有些网格单元的数据量大于数据块的大小，所以，在实际划分时，我们需要设定一个阈值（略小于 64M），当数据达到阈值上限时，则将数据输出到新的数据块中，划分最后，每个时间区间的数据块数量可能会大于 N。

4.2.2 局部一维时间索引

上一小节中计算出了时空数据的划分标准，并将数据划分到不同的分区中，在每个分区中，将在 reduce 任务阶段创建出一维时间索引，将索引和数据输出到 HDFS。最后把所有局部索引汇总到主节点中，构成全局索引。

网格索引是一个平面索引，在这个索引中网格单元的内容被无序的存储，为了更快定位到查找的时空对象，本文将采用一维时间索引管理数据块中的数据，实质上就是将包含整个数据块的 MBC，按照时间维度进一步划分成更小的 MBC。存储格式如图 4.3 所示。 局部索引是时间索引目录，相同时间区间内的时空数据存放在一起，在查询时先通过时间进行剪枝，加载所有与查询的时间范围相交的时空数据，再通过具体的时间

和空间范围查找到所需的时空对象。这种索引的优势主要有以下几个方面：（1）索引构建简单，耗时短，需要节点的处理能力低，非常适合本文的云平台。（2）在时空数据检索时，避免加载数据块内的所有数据，大大提高了检索效率。（3）对时间范围的检索非常有效。对于时空网格划分算法，数据结构简单，但是更能适用于 MapReduce 的并行计算，索引构建的时间也更短。



时间区间(t1,t2)	时空对象(C)
C1[201402030112 ,201403050223)	C11,c12,c13,c14...
C2[201403050223 ,201404050645)	C21,c22,c23,c24...
...	...

图 4.3 时空网格划分示意图

4.3 QDTree 索引

上节中介绍的 TGrid 索引，在数据进行划分时有一个弊端。当数据分布不均匀时，不同的网格数据密集程度不一样，少数的网格分布了大量的时空数据，而大部分的网格内时空对象较少，造成大量冗余的数据块。所以，为了解决这个问题，当时空数据分布不均匀时，本文采用了四叉树划分策略。

4.3.1 改进的四叉树划分算法

为了使空间数据更均匀分布，保证节点的负载均衡，对四叉树算法进行改进，设计更好的时空数据划分策略。如图 4.4，在时间段 t1~t2，四叉树的空间区域的划分图，每个空间区域均匀划分成四个子区域。图 4.5 中，四叉树的非叶节点包含四个子孩子，代表子空间区域，每一个子空间区域再被均匀划分成四份，依次类推直到划分结束，叶节点指向局部索引树。为了在大规模时空数据中高效的划分，运行 MapReduce 作业，并且按照如下算法进行四叉树的划分。

改进的四叉树划分算法如下：

（1）划分不同的时间区间，默认划分成 10 个时间段 T，并记录每个时间区间内的所有时空数据大小。

（2)在 T 时间区间与空间范围组成一个大的 MBC，按照空间范围划分成小的 MBC。

（3）如果 MBC 的时空数据总大小低于阈值，则停止划分，将该 MBC 的所有数据输出到同一分区中。相反，需要继续对 MBC 的空间区域递归的进行划分，划分成均匀的四个子 MBC。

（4）对划分的每个子 MBC，调用第（3）步，直到该 MBC 的所有时空数据的大小满足阈值。

（5）重复第（2）步，直到所有时间区间内的时空数据划分完成。

阈值默认设置为（60M），这与数据块的大小有关，数据块中的数据大小不能超过 HDFS 的默认大小（64M），这样可以满足在数据块中有足够的空间存放时空数据和局部索引文件。但阈值也不能过小，会造成四叉树的深度更大，划分时空数据时间更长，数据块的数量增大，检索效率下降。通过以上时空划分步骤，将同一时间区间内，空间相邻的时空数据划分到 HDFS 同一个数据块中。如果时空数据分布不均匀，则分布相对密集的区域四叉树的深度更大，叶节点的空间区间更小，而分布稀疏的区域叶节点空间范围较大，有效的克服了网格划分的不足。

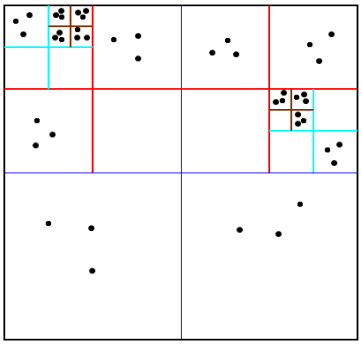


图 4.4 空间区域的四叉划分法

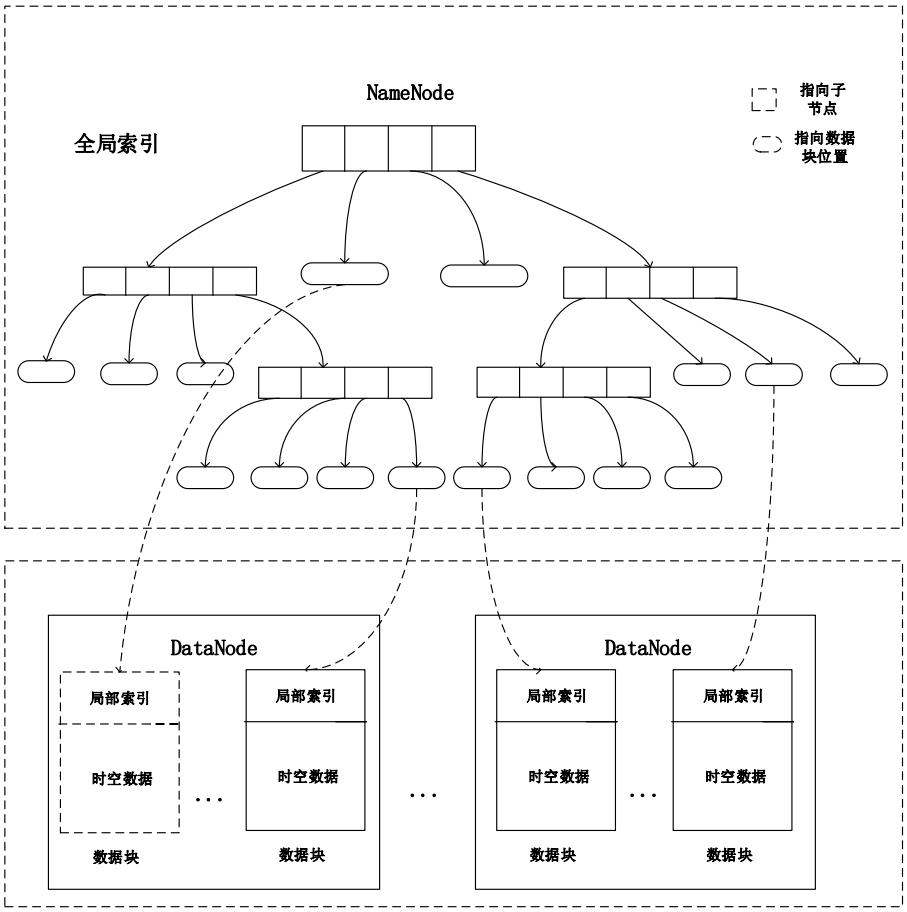


图 4.5 时间段 QDTree 树索引结构图

4.3.2 局部 3DR-tree 索引

为了提高磁盘吞吐量，减少网络带宽带来的限制，将相邻的时空数据聚集存储，提高检索效率，本文采用 3DR-tree 索引。3DR-tree 索引在传统的 R-tree 基础上扩展而来，构建索引的方式是将时间看作空间的另一个维度，利用 R-tree 进行空间索引，可以有效

的查询历史数据，很适合本文一次性写入大量时空数据并进行读取分析，在处理长时间段的查询时优于其他索引结构。在 MapReduce 任务中，时空数据划分到不同的分区之后，在 Reduce 任务阶段，我们便开始在每一个分区中构建局部的 3DR-tree 索引，基本操作与 R-tree 类似。如图 4.6，,非叶节点数据结构为（node_id,MBC）,node_id 标示该节点，MBC 表示最小外包长方体，用（t1,lon1,lat1,t2,lon2,lat2）表示。叶节点数据结构为（object_id,MBC），object_id 表示时空对象，MBC 表示时空对象在该时间段内所处的空间位置不变。该索引结构简单，非常适用于本文的云平台。

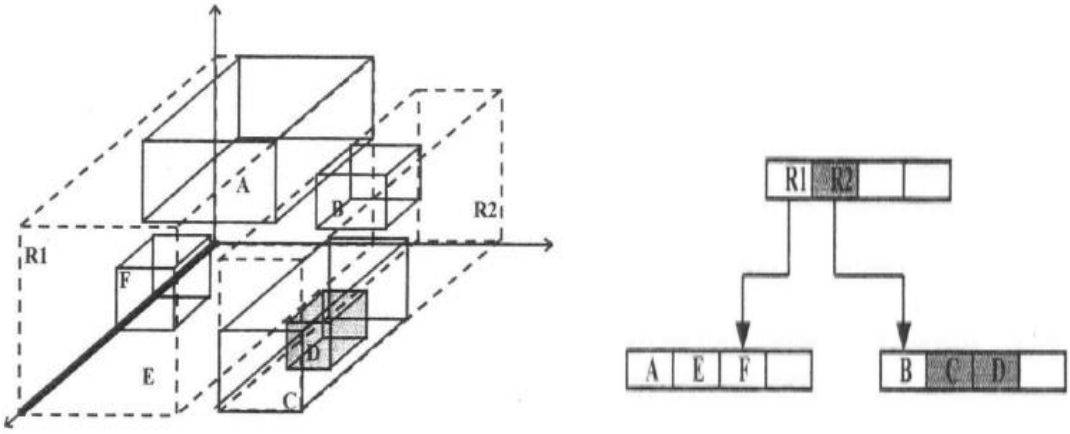


图 4.6 3DR-tree 图

4.4 本章小结

本章主要介绍了设计的两种全局-局部索引：TGrid 索引和 QDTree 索引。通过对传统网格索引和四叉树索引的改进，在 MapReduce 作业中，将时空数据并行的划分到 HDFS 数据块中；而在每一个数据块中构建的一维时间索引和 3DR-tree 索引，将相邻的时空数据对象存放在一起，提高检索效率；最后将所有局部索引整合到主节点，形成目录索引文件。

第五章 存储优化与实验分析

为了减小磁盘的存储空间，提高磁盘 IO 和网络传输，本章将介绍时空数据的存储优化策略，并进行实验分析。本章内容如下：

（1）数据存储优化。根据不同的索引结构，设计基于列存储的存储模式，减少冗余数据的分析，提高检索效率，并采用高效的压缩算法，减少磁盘存储空间与网络传输时间。

（2）实验结果与分析。测试 TGrid 和 QDTree 两种索引的构建时间，时空数据的检索效率和时空数据的存储大小等。

5.1 存储优化

由于本文所采用的云平台，单机处理能力和读写性能较低，受限于网络带宽，网络传输速度慢，处理时空数据耗时较长。当时空数据大小超过 50G，在构建索引时，往往会出现 Map 任务和 Reduce 任务失败的情况。所以，我们需要采取存储优化策略。

5.1.1 时空数据的列存储

前一章中，介绍了时空数据划分成不同的分区，并在 reduce 阶段，构造局部索引，将索引和每一个分区的时空数据写入 HDFS 数据块中。但是最后我们发现由于集群中网络传输数据较慢，磁盘读写性能较低，耗费了大量的时间，所以本节我们采用数据压缩和列存储模式。

数据库的列存储与传统的关系型数据库按行存储不同，在典型的关系型数据表中，每一行包含一个记录的字段值，那么每一条时空数据顺序存放在一起。而列存储，将相同属性的一列进行存储。如图 5.1，每一列具有相似的属性，如果查找时空对象的时间属性,则只需扫描时间这一列即可，大大降低了磁盘 IO，在进行网络传输时更有优势，所以采用列存储有以下几点优势：（1）本文的时空数据是对历史数据的查询，所以数据的修改不多，虽然列存储在写的过程中把每一行的记录分拆写，耗时较长，但是检索的效率更加高效，减少冗余数据。（2）由于列存储的每一列数据属性相同，所以数据的解析非常方便，相比于行存储，需要对每一条记录的多个属性进行分析，对处理器要

求较高。（3）列存储的每一列属性相同，压缩算法更加高效。

时间	空间	Id
20140102123	（32.23, 56.23）	3306
20140102124	（33.23, 57.23）	4307
20140102125	（34.23, 58.23）	5430

行存储	… | 20140102124 | 33.23 | 57.23 | 4307 | 20140102125 | …

列存储	… | 20140102124 | 20140102125 | (32.23, 56.23) | …

图 5.1 时空数据行列存储模式

受 RCFile[39]启发，我们将时空数据划分为多个行组，同一行组内的数据按照列存储方式进行存储，不同行组按照行存储。TGrid 索引中，局部索引为一维时间索引，时空数据按照时间区间被划分为多个 MBC，每个 MBC 被成为一个行组，行组中的时空对象按照时间顺序存放在一起；QDTree 索引中，局部索引采用 3DR-tree 索引，每个叶节点为 MBC，看做一个行组。每一个行组内的时空数据按照列存储方式写入 HDFS 数据块中。如图 5.2 所示，每一个最小外包长方体（MBC）内的数据按照列存储模式写入文件，每个 MBC 按照正常顺序存储。每个行组中包括行组头部，用于分隔 HDFS 数据块中的两个连续行组，每个行组内的数据段，同一列的所有数据顺序存储，在图 5.2 中的示例图中可以看到首先存储时间列的数据，然后存储空间列的所有数据，并依次存储其他列的数据。在数据检索时，由全局-局部索引对时间和空间进行剪枝，最终定位到 HDFS 数据块中的行组，然后根据要查找的属性，加载行组中相应的列数据，最终过滤掉不符合查询条件的时空对象。

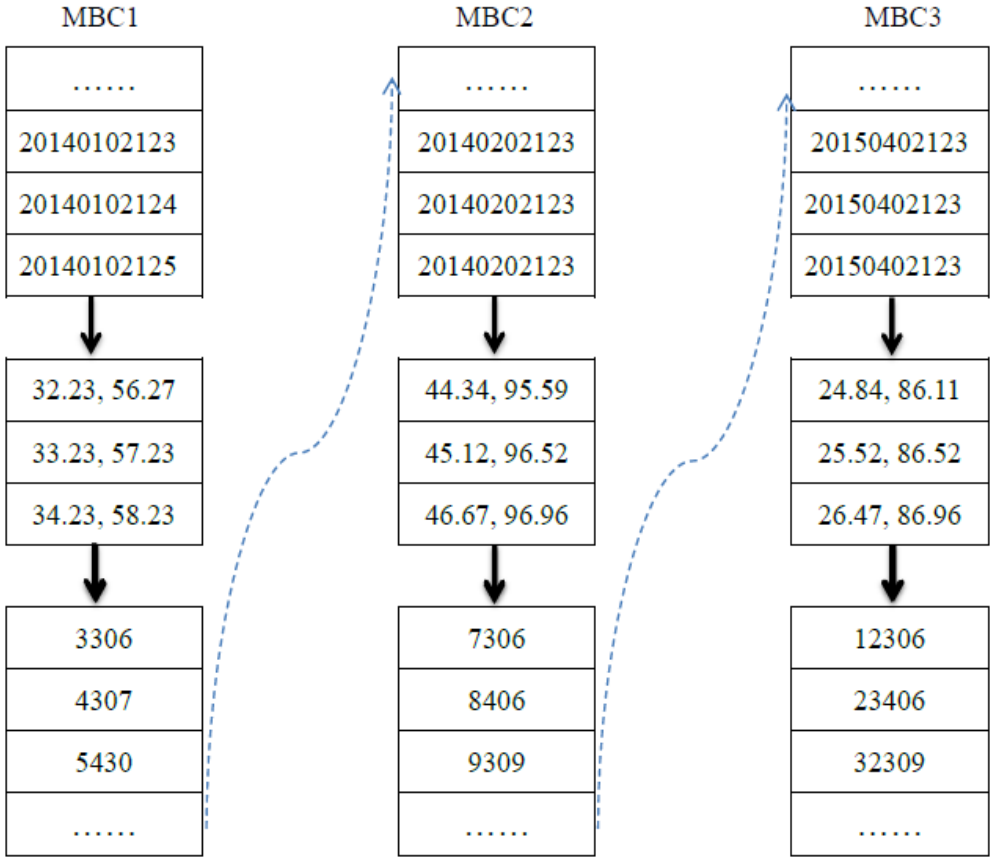


图 5.2 时空数据列存储示例图

5.1.2 时空数据压缩存储

Hadoop 集群间的通讯依赖于网络，为了提高磁盘 IO 和网络传输效率，减少索引构建时间。在 MapReduce 作业中，对于 Map 任务输出的中间结果进行压缩，可以通过更改配置文件进行设定。当时空数据构造完局部索引，写入 HDFS 数据块中时，根据不同的局部索引设定划分的行组，调用 gzip 库将每个行组中的列进行独立压缩。在时空数据查询时通过 Lazy 解压缩算法避免解压不需要的列。由于采用压缩算法，最终数据块的大小远小于 64M，所以在时空数据划分阶段，我们将划分的阈值设为 128M，这样每个数据块的实际存储大小小于 40M。

为了有效的加载全局-局部索引文件，查询所需数据，需要在 MapReduce 框架进行修改。与 SpatialHadoop[22]类似，我们也设计两个组件，tsFileSplitter 和 tsRecordReader。tsFileSplitter 读取全局索引文件，增加对时间的比对方法，在时间与空间范围上对查询的数据块进行剪枝，定位到要查找的数据块。再通过 tsRecordReader，提取数据块中的

局部索引，以<最小外包长方体，索引指针>键值对形式，将值传送给 Map 方法，在 Map 方法中不用遍历所有的时空数据，根据局部索引查找相应的行组，将所需的列读入内存进行解压，最终在 reduce 阶段，过滤掉重复和不符合查询范围的时空数据，将结果输出到 HDFS。

5.2 实验结果与分析

本文在第三章介绍的云平上，对大规模时空数据进行存储，创建分布式时空索引机制，并对时空数据进行检索测试，设计相关的性能评估对比的实验。首先，我们与传统数据库，例如 PostgreSql 相对比，测试磁盘存储大小。然后采用不同大小的时空数据对两种索引性能进行对比。

5.2.1 实验环境与数据集

在第三章中充分介绍了所设计的云平台，通过 Hadoop 自带的相关检测方法，证明了该云平台在处理海量数据的可行性与可靠性，并且为了充分发挥集群的性能，需要对 Map 和 Reduce 任务数量合理的分配。另外，本文采用 PostGIS 做对比，PostGIS 是关系型数据库 PostgreSQL 的一个扩展，可以提供空间对象存储、空间索引等服务。在本文的实验中，这种数据库不能成功加载大量的时空数据，因此在云平台上进行大规模时空数据的实验，而采用少量的时空数据与这个数据库进行实验对比。

数据集根据某地区的 5GB GPS 出租车原始数据的数据格式，设计时空数据生成程序，随机生成 100GB 模拟 GPS 时空数据，并根据本文定义的时空数据模型进行数据的预处理。数据的格式为（T,Lon,Lat,Id,A1,A2），T 表示时间，(Lon，Lat)表示空间位置，Id 表示时空对象，A1，A2 为其他属性，为方便计算，T, Lon, Lat 为 double 类型，Id，A1，A2 为整型数据。

5.2.2 时空数据存储性能分析

图 5.3 显示对于 2.19G 的 GPS 时空数据，不同存储方法存储之后的大小，在本文的系统中，是数据大小和索引大小之和。对于 2.19G 的数据，TGrid 索引构建方法用了 0.641GB 的存储空间，采取列存储方案，并且用压缩算法进行压缩，大大减少了存储空间，由于 QDtree 索引比网格索引大，所以存储空间也相对大一些，但是二者相差不多。

相对于 SequenceFile 存储方法，大约是 5~6 倍。而传统 PostGIS 数据库，存储 2.29G 时空数据花费的空间是本文系统的 10~12 倍，所以存储优势非常明显。存储 100G 的时空数据，TGrid 索引构建方法存储空间大约是 37.3GB，压缩率为 37.3%，所以对于海量时空数据，在存储空间上具有很大的优势。

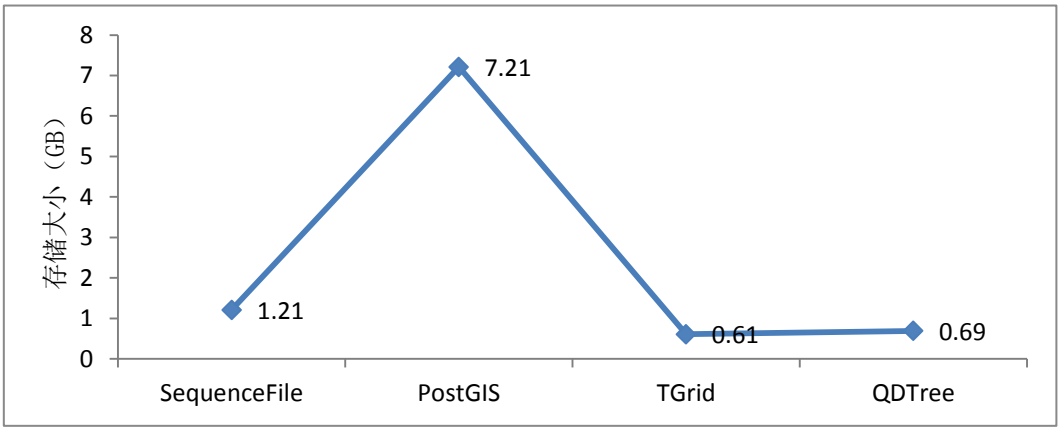


图 5.3 不同存储结构数据大小对比

5.2.3 时空数据索引构建时间分析

图 5.4 中，分析两个索引构造的时间。数据预先生成，索引的构建是读取 HDFS 上的数据，然后经过两个 MapReduce 任务一次性构建，支持多次读取，检索时空数据的内容。数据从 1G 增长到 20G，索引的构建时间呈线性增长，由于本文在网络传输时数据采用压缩机制，所以在索引构建时，磁盘 IO 和网络传输方面大大减少了时间。TGrid 索引的构造方式比较简单，花费的时间也比 QDTree 索引少，而随着数据量的增加，在 MapReduce 作业中并发执行的 map 任务和 reduce 任务的数量也会增加，提高了索引构建效率。在图 5.5 中，采用 TGrid 索引划分的数据块数量明显多于 QDtree 索引，TGrid 索引在数据划分时空间区域的数据密集程度差别很大，需要划分成更小的空间区域才能满足划分的阈值，而 QDtree 索引采用的改进四叉树算法，时空数据密集的空间区域被划分的更小，而数据稀疏的区域则停止划分，所以这种划分策略减少了数据块的数量。

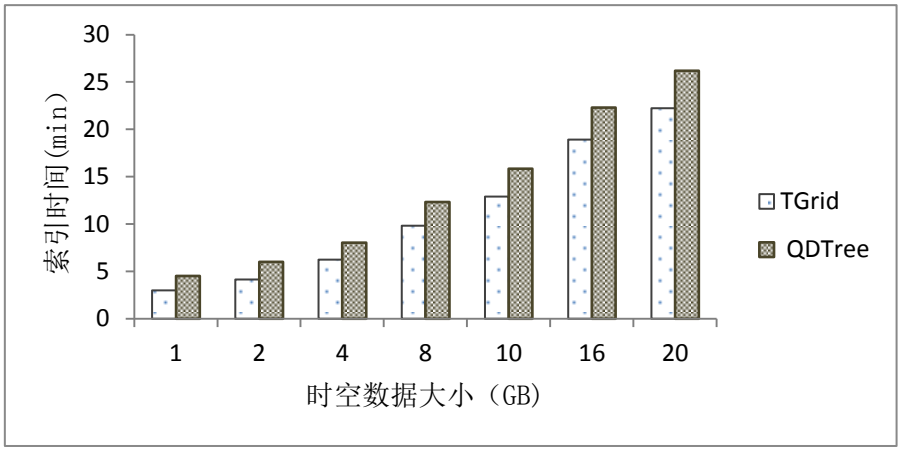


图 5.4 索引构建时间

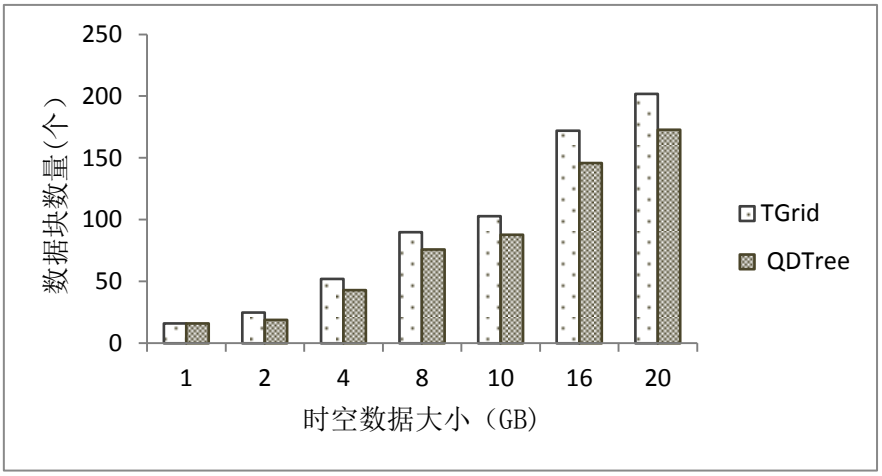


图 5.5 数据块的数量

5.2.4 时空数据检索性能

为了测试索引的检索性能，我们采用 100G 的模拟出租车时空数据。选择多个空间范围和时间范围进行检索实验。在图 5.5 中，在固定的时间段（t1~t2）内，检索不同的空间范围，横坐标表示空间面积从总面积的 0.001%增长到 1%。在空间检索方面，QDTree 效果比 TGrid 索引好，但随着检索空间的变大，检索过程中需要加载的数据块数量增加，在 TGrid 的局部一维索引中，虽然可以很快定位到查找的时间段，但是在空间查询时略显不足，需要遍历行组内的所有时空对象，而且由于时空数据在划分时会出现大量数据块内容少的情况，数据块的数量较多。而 QDTree 索引，采用改进的四叉树划分时空数据，每个数据块大小基本一致，时空数据分布均匀，在检索时空数据时能够很好的定位

到查找的 MBC，查询效率相对较高。在图 5.6 中，横坐标表示时空数据总时间段的倍数，从 0.01%增长到 0.2%，当时间间隔小的时候，TGrid 索引检索性能略高于 QDTree 索引，这是因为 TGrid 索引对时间的剪枝比较高效，但是随着时间跨度增大，检索的时空对象的增加，TGrid 索引显现出对空间属性检索低下的缺点。相比较之下，QDTree 更适用于大范围的时空检索。

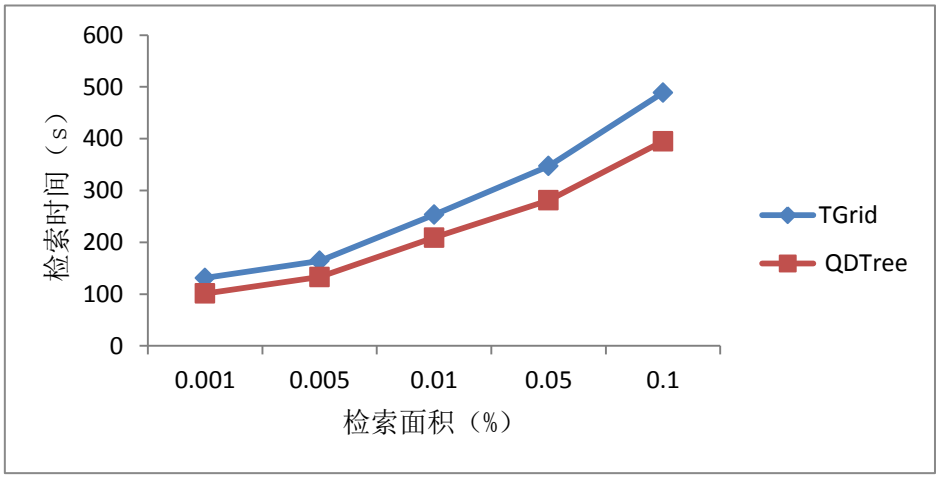


图 5.5 固定时间间隔内检索空间区域

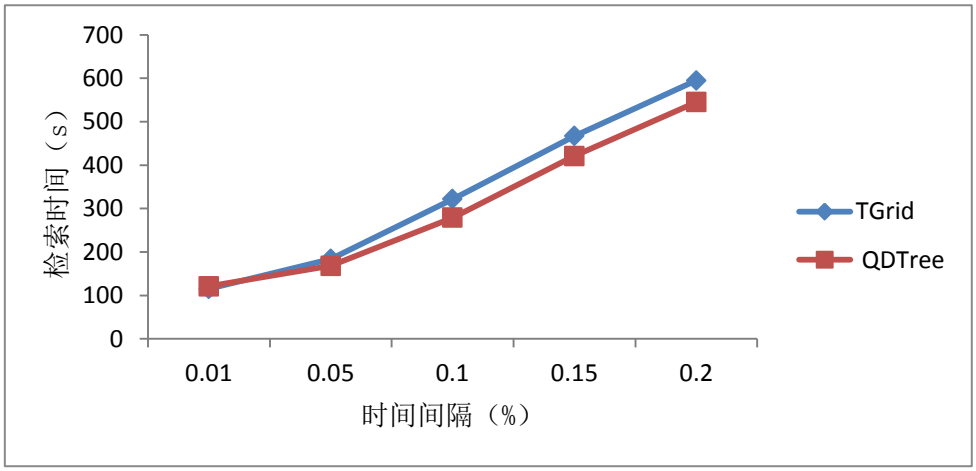


图 5.6 固定空间范围检索不同时间段

图 5.7 展现了该云平台的可扩展性强和索引的稳定性。实验采用 20G 的模拟时空数据，该集群节点从 15 增加到 30，检索的范围：总时间的 1%，总空间范围的 1%。随着节点的增加，系统的性能有了明显提升，尤其是本文的云平台，由于单节点性能过低，处理能力有限，需要将 MapReduce 作业中的任务分配到多个节点。

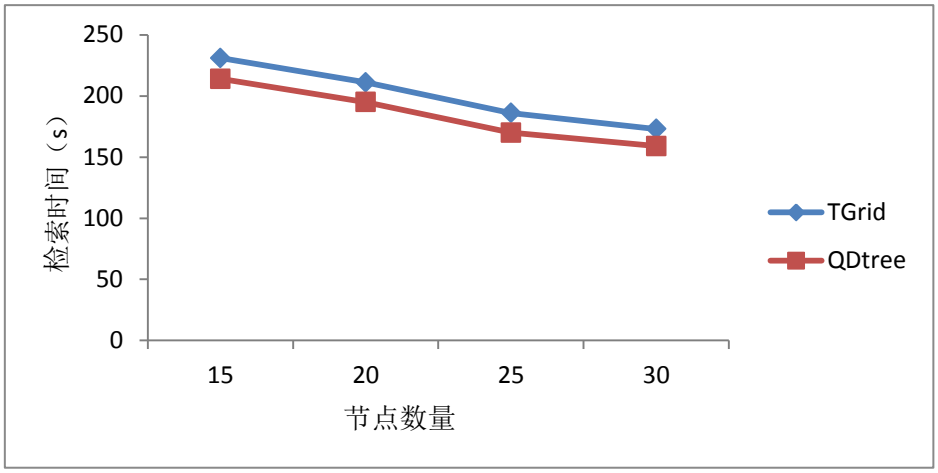


图 5.7 云平台扩展性能

5.3 本章小结

本章主要介绍对时空数据的存储优化策略，通过列存储方式和压缩算法，大大减小了磁盘存储空间，提高了网络传输和数据检索效率。通过实验，对本文中构建的两层索引在创建时间和检索效率上进行对比， TGrid 索引的构造时间比 QDtree 索引短，TGrid 索引在数据划分时会产生更多的数据块，部分数据块的数据较少，并且局部索引对空间属性的检索效率较低，所以该索引适合均匀分布的时空数据。而 QDtree 索引在数据划分时更加合理，每个数据块大小基本一致，耗时也相应较长，不过索引的剪枝效果更好，检索效率较高。

第六章 总结与展望

6.1 论文总结

随着时空数据的快速增长，我们迫切需要部署分布式时空数据库对海量时空数据进行处理。但是目前分布式时空索引的研究不成熟，而用于大数据处理的云平台在成本、能耗、场地等方面消耗巨大。本文针对这些问题进行研究，采用价格低廉的 Cubieboard2 搭建了低成本、低能耗的云平台，设计了分布式时空索引机制，在时空数据存储方面进一步优化，实现对时空数据的高效存储与检索。

第一章介绍了课题研究的背景与其意义，根据国内外研究的现状，确定分布式时空索引的研究目标。第二章介绍了本文中用到的相关技术和理论，对当前比较流行的分布式空间索引和时空索引技术进行分析，确定本文的研究方案。第三章介绍了采用低廉的 ARM 开发板搭建基于 Hadoop 的分布式云平台，充分发挥其高扩展性、高可靠性、高容错性等优势，不但在成本和能耗上体现出了巨大的前景，而且在处理大数据方面也显示出了较强的处理能力。第四章介绍设计的 TGrid 和 QDtree 索引，通过改进的网格和四叉树算法，均匀的划分时空数据，并设计一维时间索引和多维 R-tree 索引，构造出两种全局-局部索引，适用于分布类型不同的时空数据。第五章对时空数据的存储进行优化，采用列存储方式和压缩算法，减小磁盘存储空间，提高了网络传输效率。最后，设计不同的实验，在索引构造时间和检索效率上进行对比，分析其不同的适用情况。

本文设计搭建的云平台非常适合普通用户学习和研究，通过设计分布式时空索引和优化存储结构，使得该云平台能够高效的处理海量时空数据，有很大的借鉴意义。

6.2 工作展望

本文基于 Hadoop 搭建的云平台，设计分布式时空索引和存储策略，能有效的存储和检索大规模时空数据。但是还有许多改进之处：

（1）对于时空数据更多的检索功能还需要进一步完善，也将对分布式时空索引提出更大的挑战。

（2）时空索引的学习需要进一步加强，另外，对于引领潮流的最新技术要不断的

学习，如 Storm，Spark，这些知识对于海量时空数据的研究有很大意义。

（3）最后，我们可以研究时空数据可视化的功能模块，直观的分析时空数据分布。

随着不断学习和研究，我们将不断完善分布式时空索引，更能适用于多样化的查询算法，挖掘出有价值的信息。

参考文献

[1] Zickuhr K. Three-quarters of smartphone owners use location-based services[J].Pew Internet & American Life Project, 2012.

[2] Dhar S, Varshney U. Challenges and business models for mobile location based services and advertising[J]. Communications of the ACM, 2011, 54(5):121-128.

[3] Dingler J R, Rudd J R, TREVATHAN M B. Location based services with Multiple transmission methods: U.S. Patent 8,515,445[P]. 2013-8-20.

[4] Aloudat A,Michael K, Chen X, et al. Social acceptance of location-based mobile government services for emergency management[J]. Telematics and Informatics, 2014, 31(1):153-171.

[5] 郑祖芳. 分布式并行时空索引技术研究[D]. 中国地质大学, 2014.

[6] 张林,汤大权,时空索引的演变与发展[J].计算机科学,2010,37(4):15-26.

[7] Cattell R. Scalable SQL and NoSQL data stores[J]. ACM SIGMOD Record, 2011, 39(4): 12-27.

[8] Chang F, Dean J, Ghemawat S, et al. Bigtable: a distributed storage system for structured data[J]. Acm Transactions on Computer Systems, 2008, 26(2):205--218.

[9] Botea V, Mallett D, Nascimento M A, et al. PIST: an efficient and practical indexing technique for historical spatio-temporal point data[J]. GeoInformatica, 2008, 12(2): 143-168.

[10] Cudre-Mauroux P, Wu E, Madden S. Trajstore: An adaptive storage system for very large trajectory data sets[C]. Data Engineering (ICDE), 2010 IEEE 26th International Conference on. IEEE, 2010: 109-120.

[11] Guttman A. R-trees: a dynamic index structure for spatial searching[J]. ACM SIGMOD Record, 1984, 14(2):47-57.

[12] Comer D. Ubiquitous B-tree[J]. ACM Computing Surveys (CSUR), 1979, 11(2): 121-137.

[13] Schneider R, Seeger B, Beckmann N, et al. The R*-tree: an efficient and robust access method for points and rectangles[C]. Proc. ACM SIGMOD Symposium on Principles of Database Systems. 1990: 322-331.

[14] Singh S, Mayfield C, Prabhakar S, et al. Indexing uncertain categorical data[C]. Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. IEEE, 2007: 616-625.

[15] Khalefa M E, Mokbel M F, Levandoski J J. Skyline query processing for incomplete data[C]. Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. IEEE, 2008: 556-565.

[16] Pelanis M, Šaltenis S, Jensen C S. Indexing the past, present, and anticipated future positions of moving objects[J]. ACM Transactions on Database Systems (TODS), 2006, 31(1): 255-298.

[17] Apache Hadoop. http://wiki.apache.org/hadoop/

[18]黄山. 基于 Map-Reduce 框架云环境时空查询技术研究与实现[D]. 东北大学, 2011.

[19]Tan H, Luo W, Ni L M. CloST: a hadoop-based storage system for big spatio-temporal data analytics[C]. Proceedings of the 21st ACM international conference on Information and knowledge management. ACM, 2012: 2139-2143.

[20]Cary A, Sun Z, Hristidis V, et al. Experiences on processing spatial data with mapreduce[C]//Scientific and statistical database management. Springer Berlin Heidelberg, 2009: 302-319.

[21]Yunqin Zhong,Jinyun Fang,Xiaofang Zhao. VegaIndexer: A Distributed index scheme for big spatio-temporal sensor data on cloud[C].Geoscience and Remote Sensing Symposium (IGARSS), 2013 IEEE InternatIOnal Digital Object, 2013: 1713 – 1716.

[22] Zhong Y, Zhu X, Fang J. Elastic and effective spatio-temporal query processing scheme on hadoop[C]. Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data. ACM, 2012: 33-42.

[23]钟运琴, 方金云, 赵晓芳. 大规模时空数据分布式存储方法研究[J]. 高技术通讯, 2013, 23(12): 1219-1229.

[24] Eldawy A, Mokbel M F. SpatialHadoop: A MapReduce framework for spatial data[C]//Proceedings of the IEEE International Conference on Data Engineering (ICDE'15). IEEE. 2015.

[25] Eldawy, A, Mokbel, M.F, Alharthi, S, et al. SHAHED: A MapReduce-based system for querying and visualizing spatio-temporal satellite data[C]// Data Engineering (ICDE), 2015 IEEE 31st International Conference on. IEEE, 2015:1444-1447.

[26]孙大为, 常桂然, 陈东, 等. 云计算环境中绿色服务级目标的分析, 量化, 建模及评价[J]. 计算机学报, 2013, 36(7): 1509-1525.

[27] Gong J, Zhu Q, Zhong R, et al. An efficient point cloud management method based on a 3D R-tree[J]. Photogrammetric Engineering & Remote Sensing, 2012, 78(4): 373-381.

[28] Ghemawat S, Gobioff H, Leung S T. The Google file system[C]//ACM SIGOPS operating systems review. ACM, 2003, 37(5): 29-43.

[29] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[30] Kim K C, Yun S W. MR-Tree: a cache-conscious main memory spatial index structure for mobile GIS[M]//Web and Wireless Geographical Information Systems. Springer Berlin Heidelberg, 2004: 167-180.

[31] Nascimento M A, Silva J R O. Towards historical R-trees[C]//Proceedings of the 1998 ACM symposium on Applied Computing. ACM, 1998: 235-240.

[32] Choi Y J, Min J K, Chung C W. A cost model for spatio-temporal queries using the TPR-tree[J]. Journal of Systems and Software, 2004, 73(1): 101-112.

[33] Theoderidis Y, Vazirgiannis M, Sellis T. Spatio-temporal indexing for large multimedia applications[C]// Multimedia Computing and Systems, International Conference on. IEEE Computer Society, 1996:441-448.

[34] Nascimento M A, Silva J R O. Towards historical R-trees[C]// Proceedings of the 1998 ACM symposium on Applied Computing. ACM, 1998:235-240.

[35] Pfoser D, Jensen C S, Theodoridis Y. Novel Approaches in Query Processing for Moving Object Trajectories.[C]// VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt. 2000:395-406.

[36]吴丽鑫, 闫思宇. MapReduce 框架下基于 R-树的 K-近邻连接算法设计[J]. 数字技

术与应用, 2015 (7): 135-135.

[37]http://docs.cubieboard.org/products/start from 30.3.2015.

[38]Griffiths N. nmon performance: A free tool to analyze AIX and Linux performance[J]. 2003.

[39]He Y, Lee R, Huai Y, et al. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems[C]//Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011: 1199-1208.

作者简介及在学期间所取得的科研成果

作者简介：

姓名：杨学毅

性别：男

民族：汉族

出生日期：1990 年 4 月 2 日

出生地：山东省济南市

学习经历：2013 年 9 月份就读吉林大学计算机科学与技术学院计算机应用技术专业硕士研究生。

参加项目：

项目名称：吉林省村镇规划与建设信息系统

合同编号：2013220101001858

项目时间：2013.11-2014.12

负责任务：负责数据分析与 GIS 地图展现

发表论文：

[1] Yang X Y, Huang L, Wang K P. Detecting Link Communities Based on Hadoop[J]. Applied Mechanics & Materials, 2015, 727-728:955-958.

致 谢

研究生的三年时光飞速而过，这三年的学习生活充实而有意义。不但学到了很多知识，而且在项目实践中更是体会到团队合作的重要性和面对各种难题锲而不舍的精神。为我的人生上了宝贵的一课，让我有自信、有能力面对将来的挑战。在完成毕业设计的课题过程中，得到了老师的悉心指导和同学的大力帮助。

首先，我要感谢我的导师黄岚教授。我们整个实验室梯队在黄老师的带领下蒸蒸日上，学术科研氛围浓厚，技术交流频繁，老师组织的讨论班中我学到了很多知识。黄老师以严谨的学术态度要求我，并尽最大努力帮助我解决遇到的难题。在我做毕业设计阶段 ，黄老师更是对我认真耐心的指导，在许多问题上给予针对性的意见，让我受益匪浅，黄老师还在百忙之中组织了众多老师和同学为我们的课题进行分析和指导。在论文的撰写时也对我提出了宝贵的建议。在此，向黄老师表达最真挚的感谢！

其次，我要感谢王康平老师。王老师待人随和，拥有一流的计算机技术，在科研方面同样非常优秀，是我学习的榜样。研一上学期跟随王老师做项目，在面对各种困境时，王老师一直支持我，指导我，提高了我的各项能力。无论是我在技术学习、课题研究，还是生活中碰到的问题，王老师都耐心的给我解答。在此，真诚的感谢王老师的帮助。还有实验室的师兄弟们，师兄师姐为我传授了很多经验。在我课题研究阶段，也得到了师弟们大力帮助，我们一起学习，讨论，使我不断的进步。

最后，我要感谢我的家人。我的家人是我精神上的支柱，在遇到困难时我才能坚持不懈，感谢我家人的陪伴。