文章编号:1673-1549(2018)02-0089-07　　DOI:10.11863/j.suse.2018.02.14

基于 MongoDB 集群的遥感数据存储方法的研究

李宏志[1], 李苋兰[2]

(1. 滁州学院信息学院, 安徽 滁州 239000; 2. 福建师范大学光电与信息工程学院, 福州 350000)

摘　要:随着遥感探测技术的发展,遥感数据的数量和质量都在逐步提高,如何有效地存储和管理这些数据成为当前研究的重点。应用 MongoDB 非关系型数据库,提出了一种基于 MongoDB 数据库集群的遥感数据的储存方法,设计了影像文件与元数据分离存储的策略;并对 MongoDB 集群数据分片的片键机制进行了深入研究,根据遥感数据的特点提出了基于"升序键 + 搜索键"数据分片方案,通过应用哈希一致性算法改善集群间数据分布均匀性,通过分布式的内存缓存提高遥感数据的检索效率,并实现了基于此方法的架构模型。最后针对所提出的存储方法设计了测试方案并进行了对比测试实验,实验结果表明该存储方法能够较好地提高遥感数据存储和检索的性能,适用于遥感领域的数据存储。

关键词:遥感数据;MongoDB 集群;片键策略;哈希一致性;存储模型

中图分类号:TP319　　**文献标志码**:A

引　言

高精度的遥感数据作为一种大容量的信息载体,在国土勘测和军事安全方面有着重要的作用。随着我国的航天遥感技术的发展,遥感数据的数量和质量都在逐步的提高,单个的遥感文件可达到数百 MB,甚至达到数 GB 的大小,如何高效且安全地存储和管理这些遥感数据已成为当前空间信息科学领域的重要研究方向。

由于 MongoDB 集群在存储和管理遥感数据方面具有一定的优势[1-3],使得基于 MongoDB 集群在遥感数据存储方面的研究成为热点。文献[4 - 6]详细介绍了使用 MongoDBGridFS 分布式文件系统存储和管理影像图片文件的方法;田帅[7]提出了一种基于 MongoDB 和 HDFS 的大规模遥感数据存储方案,该方案将遥感元数据和遥感影像文件分开存储和管理,由 MongoDB 存储非结构化的元数据,HDFS 分布式文件系统负责存储大规模的影像文件。这种存储方案需要维护元数据和影像文件的对应关系,同时涉及到的技术较复杂,不便于维护;赖积保等[8]提出了基于云计算的遥感影像存储模型 RSC - DOM,该模型采用改进的遥感数据直接存储方式,实现了对遥感数据的分布式管理,但应用场景要求数据是均匀分布的,因此具有一定的局限性;秦强等[9]提出了基于 MongoDB 的遥感数据存储方案,通过 NoSQL 技术实现了非结构化的遥感元数据的存储;梁海等[10]提出了通过 MongoDB 的分片技术提高数据库的读写性能和效率。本文在上述研究成果基础上提出了基于高速缓存和 MongoDB 数据库的存储模型,底层存储模块采用 MongoDB 及其分布式文件系统 GridFS。在底层存储之

收稿日期:2018-01-12

基金项目:安徽省自然科学基金面上项目(1408085MF126)

作者简介:李宏志(1989-),男,安徽池州人,助理实验师,硕士,主要从事搜索引擎技术与数据挖掘技术方面的研究,(E-mail)1071260932@qq.com

上新建一层基于内存型存储器的缓存层用于快速存储遥感元数据,结合一致性哈希算法,提高遥感数据在分布式集群中分布的均匀性,减少各存储节点之间的数据迁移次数,提高系统的访问效率。

1 MongoDB 的存储架构简介

1.1 MongoDB 的数据存储机制

MongoDB 存储的内容是结构松散的类似于 JSON 结构的 BSON 格式数据,支持 MapReduce 功能,以保证其可以对数据进行复杂的关系和分析;除此之外,MongoDB 支持对于地理信息的索引及检索操作。

GridFS 是构建于 MongoDB 之上的分布式文件系统,利用 MongoDB 的分布式存储机制实现遥感影像文件的存储;GridFS 默认使用 fs. files 和 fs. chunks 集合来存储文件的元数据和文件块。MongoDBGridFS 支持对存储的大文件自动分片,分布式文件存储架构如图 1所示。

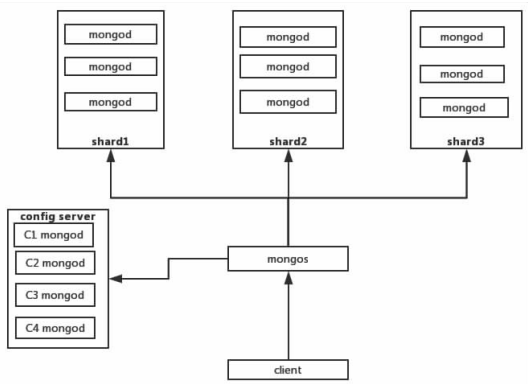

图 1 GridFS 的文件存储分片架构

MongoDB 的存储架构主要由三个部分组成:分片存储服务器、集群路由服务器以及配置服务器。其中存储服务器负责集群数据和文件的存储,路由服务器负责请求的寻址和定位,而配置服务器保存集群服务器所有的配置信息。

MongoDB 采用数据块的概念作为基本的存储单元,当单个数据块的大小达到某个阈值之后,系统会进行数据块分裂,各分片服务器通过数据块存储应用数据,当系统中各分片中的存储块的数量出现失衡时,会启动再平衡算法,在各分片之间进行数据迁移,具体过程如图 2 所示。

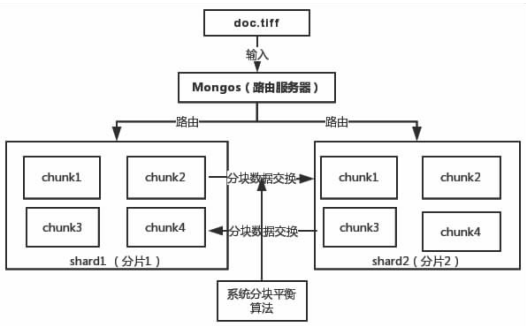

图 2 MongoDB 数据存储机制

1.2 MongoDB 数据分片的片键选择

MongoDB 分片机制能够根据文件的大小将数据划分到不同的分片服务器上,数据分片一般应在数据库建立早期进行,避免由于分片操作造成过多的系统开销。要进行数据分片必须要选择集合中一个或多个索引作为数据分片的基准,称之为片键。片键的选择关乎到集群的性能。MongoDB 根据片键进行数据分块主要包括以下几个步骤:

(1) 开启集合的数据分片。

(2) 创建集合字段索引。

(3) 选定索引字段作为片键。

(4) 系统会依据集合选定的片键对文档数据进行分块,使文档数据按照片键分布在不同的数据块中。

(5) 各分片按照数据块(chunk)的数量调整分片之间的数据分布。

2 基于高速缓存和 MongoDB 的遥感影像文件的存储系统

MongoDB 数据库能够兼容多种数据格式的遥感元数据,对于遥感影像文件的管理也较为方便,遥感数据满足如下模型:

$$ObjectRS = \{Object\ Data, Object\ Image, Object\ Vector\}$$

其中,Object Data 指的是遥感元数据及其对象,Object Image 表示遥感影像数据,Object Vector 代表矢量数据对象。

通常对于遥感数据的访问和操作主要集中于遥感元数据,因此可以使用内存型缓存来加速这部分数据的访问过程。本文研究的遥感元数据见表 1。

表 1　遥感元数据示例

字段	格式	样　例
编号	ItemId	dssc_item_001
名称	ItemName	ah_chuzhou_langya_001
经度	Longitude	40.739037
纬度	Latitude	73.992964
比列尺	Scale	1 - 250000
观测时间	ObserveTime	2017 - 9 - 20
观测方式	ObservePattern	ht - yc - fengyun - 01
地形信息	TerrainInfo	dem_ah_chuzhou_langya_001
文件格式	FileStyle	tiff
季节信息	SeasonInfo	spring
其他描述信息	Desc	安徽省滁州市琅琊区风云系列卫星遥测数据

根据对 MongoDB 分片机制的研究可知,当数据量过大时,各数据分片之间存在数据迁移操作,这会使系统开销较大,影响系统的稳定性。因此可以通过一致性哈希算法建立应用数据与分片节点之间的映射关系,减少系统级的数据平衡操作的执行次数。

2.1　存储系统的架构设计

在设计存储系统时,除了考虑遥感数据的存储和管理之外,还需要保证与其他系统平台兼容,要求系统具备接受和处理来自其他平台的数据并且能够为其他的应用平台提供数据服务的能力。为了保证整个平台的可靠性和扩展性,系统采用层次化设计,可分为交互层、服务层、数据缓存层和数据存储层,系统分层结构如图 3 所示。

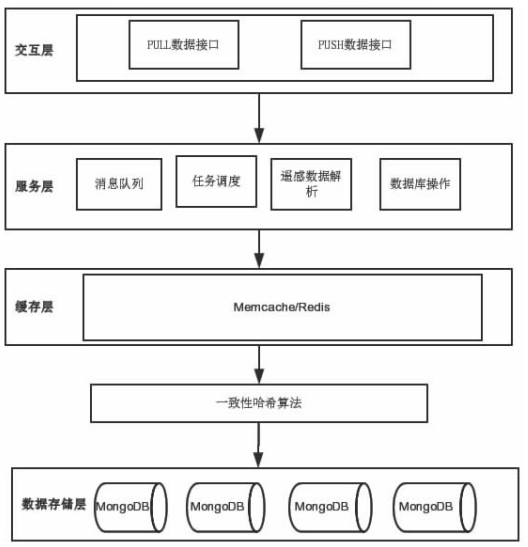

图 3　基于 MongoDB 的存储系统的多层架构模

(1) 交互层位于整个系统的最前端,负责与其他系统或用户进行交互。交互层提供了 PULL 和 PUSH 的两种集成方式,支持多种传输协议和数据格式,一方面用户可以通过 PUSH 方式定义的通用接口向系统提交遥感数据,另一方面用户可以通过 PULL 方式获取系统中的遥感数据,便于与其他系统集成。同时,交互层负责为客户端提供遥感元数据的格式规范,影像文件以及高程文件的上传接口。

(2) 服务层为整个系统架构的核心层,主要包括消息队列模块、任务调度模块、数据库操作服务以及元数据解析模块。系统消息队列负责接收用户的上传的数据,由服务程序实现队列数据的存储,这样能够减少数据库系统的瞬时压力,以提高系统的并发能力。任务调度模块负责调度和管理系统中的各种任务进程的执行,任务调度模块包含了任务执行队列、调度器以及触发器等核心模块[11]。任务调度模块与消息队列配合共同完成请求的分发和系统的负载均衡。

(3)缓存层用于缓存遥感元数据,使用内存型缓存,如 Memcache、Redis 缓存服务器;通过高速缓存使常用数据常驻内存,提高数据的访问效率。通过缓存层访问遥感数据的基本步骤:

S1:查询根据给定的键值查询缓存层是否包含数据。

S2:如果包含了所要的数据,直接从缓存层返回数据或影像文件信息。

S3:如果未命中缓存信息,则进入数据存储层的对应数据库获取数据信息。

S4:系统将获取的信息返回给上层客户端,并回写至缓存中。

(4)数据存储层用于整个系统的数据存储和备份,采用 MongoDB 及 GridFS 集群实现。其中 MongoDB 负责存储和管理遥感元数据,GridFS 集群负责存储和管理遥感影像文件。

架构模型通过引入哈希一致性算法,建立数据与分片节点之间的映射关系,均衡各分片之间的数据数量,减少分片之间的数据迁移次数。

使用哈希一致性算法分配存储节点的过程如下:

```
初始化分片节点的 ip 地址序列 - > shards_queue;
while( shards_queue) {
node_ip = shards_queue. pop( );
```

```
node_key = node_id mod (2^32); //计算对应节点的 key
node_map.push( <node_ip,node_key> );
}
data_key = hash(data.item); //计算待存数据散列值
select_ip = selectNode(data_key,node_map) ;//选择存储的分片 IP 节点
return select_ip;
```

2.2 遥感元数据模型的设计

高效的数据模型能够很好满足应用程序的需求,设计文档数据结构的关键在于考虑是使用嵌入式数据模型还是使用规范化的数据模型。如图 4 所示,Location 和 TerrainInfo 属性字段作为子文档嵌入到父级文档中,这种嵌入式结构是指与给定文档相关联的文档以子文档的形式存放于给定文档内部;规范化数据模型指各文档之间通过使用、引用来表达对象之间的关系,如图 5 所示,该结构类似于关系型数据库的外键关联方式[12]。

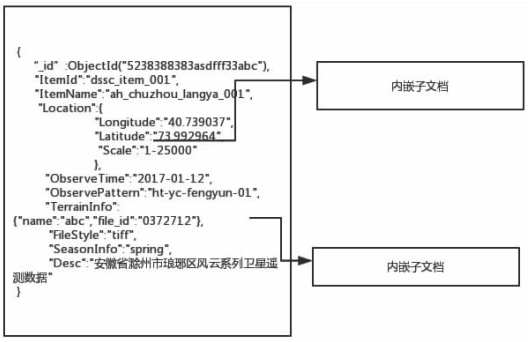

图 4 嵌入式数据模型

图 5 规范化数据模型

存储模型的设计需要权衡存储空间的利用率和系统的查询性能。遥感元数据包含一对多、多对多的实体关系,考虑到遥感元数据的特点,属于读占优类型的数据,数据一旦写入,修改可能性较小,因此选择内嵌式数据模型。另外选择内嵌式数据模型便于对遥感数据进行大规模的聚合和分析操作,提高系统对遥感数据的分析性能。

2.3 遥感元数据 MongoDB 存储片键选择

集合片键决定了数据在集群分片中的分布情况,影响系统的存储性能,因此片键的选择对于存储系统的设计至关重要。良好设计的片键应具备升序片键和随机片键的优点。既要具备良好的数据局部性,也不至于由于数据分布过于局部出现数据热点造成系统的读写瓶颈,要达到这种效果可以采用复合片键[13]。本文在分析遥感元数据的基础上,提出了“升序键 + 搜索键”的片键设计方案。

升序键一般选择文档中的自增_id 或时间戳,在遥感数据文档中符合升序片键的字段有自增_id 和文档记录时间 ObserveTime。字段_id 的自增特性会导致某个节点成为数据热点造成数据过于集中,观测时间字段 ObserveTime 既具有升序的特点又不至于使分片数据过于集中[14];而应用程序对数据的访问主要是集中在系统新数据的访问,按照文档时间进行数据分片能够使新数据尽可能的保存于内存中,提高数据的访问效率[15]。

搜索键的选择是根据应用需求设计的,为了保证查询数据的隔离,搜索键采用常用的搜索字段[16]。对于本文中的遥感数据而言,较为频繁使用的查询字段是经纬度,因此选择经纬度作为分片搜索键能够让经纬度较近的数据分布集中,减少在查询时访问分片的次数。

本文选择“记录时间 + 经度 + 纬度”作为复合片键,记录时间片键能够使文档数据均匀分布在各分片上。选择经纬度作为搜索片键能够提高系统的查询性能和聚合分析能力,如图 6 所示。

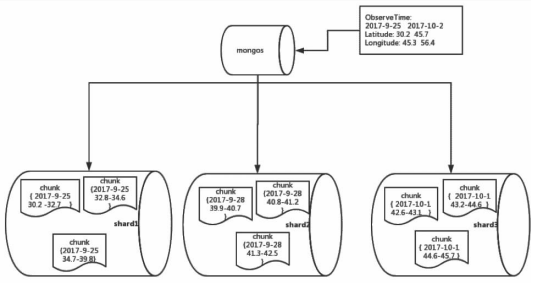

图 6 基于分片标签的复合片键选择策略

3　性能测试与分析

为验证本文提出的基于 MongoDB 的多层分布式存储系统的性能,基于上文提出的数据格式和文件类型进行一系列测试实验。首先针对不同片键选择策略下的数据分布和查询性能的测试;然后验证在引入内存型缓存的情况下,对系统查询性能的提升。

3.1　数据模拟

根据上文提出的遥感元数据模型,基于既有遥感数据信息库模拟出 20 万条遥感数据记录,占用存储空间约 2.3 GB。模拟数据的选取范围:

(1) 数据的观测时间从 2017 年 1 月到 2017 年 8 月份。

(2) 遥感数据的观测方式从“ht - yc - fengyun - 01”和“ht - yc - fengyun - 02”中随机选取。

(3) 观测经度数据在[115.6,119]区间内产生随机数,纬度数据在[28.2,35.7]区间内产生随机数。

对于选取的遥感数据模型中的其他字段,采用模拟填充方式生成字段内容。

3.2　测试环境搭建

测试环境采用八台微型计算机搭建成分布式的存储平台,存储节点基本配置:内存 1 GB,硬盘大小 120 GB,CPU Intel/英特尔 i5 7500,操作系统 CentOS 6.3。其中 6 个节点作为数据存储的分片节点,选取 1 个节点作为运行配置服务器,1 个节点作为路由服务器。测试环境的网络拓扑结构如图 7 所示。

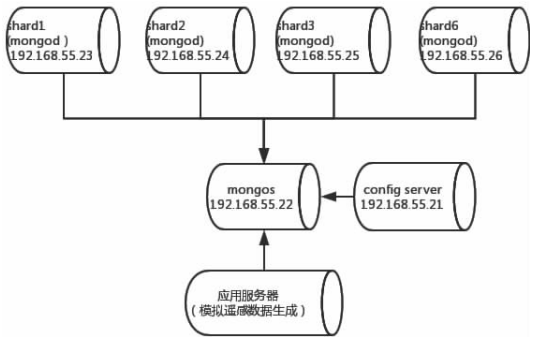

图 7　测试环境拓扑结构图

3.3　数据分片性能测试

本节针对 2.3 节提出的{记录时间,经度,纬度}复合片键方案,通过模拟数据验证该方案对数据分布的均衡性和查询性能的提升。测试数据采用上文中提到的 20 万条模拟数据,通过对比多组片键组合方案,验证该复合片键方案的性能优势。片键策略的实验分组见表 2。

表 2　片键策略的实验分组

序号	类　型	字　段
1	单片键	{ItemId}
2	单片键	{ItemName}
3	复合片键	{ItemId, ItemName}
4	复合片键	{ Longitude, Latitude }
5	复合片键	{ ObserveTime, Longitude, Latitude }

将模拟生成的 20 万条数据按照表 2 的片键策略存入图 5 所示的存储平台中,文档分布结果见表 3。

表 3　文档数量分布测试结果

片　键	文档数量分布测试结果/万条					
	Shard1	Shard2	Shard3	Shard4	Shard5	Shard6
{ItemId}	4.76	3.12	3.25	3.13	3.23	2.51
{ItemName}	2.9	3.75	4.32	3.08	3.16	2.79
{ItemId,ItemName}	4.51	2.32	3.09	3.13	2.43	4.52
{Longitude,Latitude}	4.89	3.23	3.19	3.34	2.45	2.9
{ObserveTime,Longitude, Latitude }	3.69	3.23	3.26	3.47	3.21	3.14

对于表 3 中得到的测试数据,采用统计学中的标准差衡量数据分布是否均衡,计算结果见表 4。

通过表 4 的标准差统计数据,{ObserveTime,Longitude,Latitude}片键方案文档数据在集群中分布的比较均衡,说明{ObserveTime,Longitude,Latitude}片键选择方案能够优化数据在集群中的分布。

表 4　文档数量分布标准差

序号	片　键	标准差
1	{ItemId}	0.7507
2	{ItemName}	0.58725
3	{ItemId,ItemName}	0.97326
4	{Longitude,Latitude}	0.82735
5	{ObserveTime,Longitude, Latitude }	0.20714

图 8 为基于测试数据的查询性能统计情况,从中可以看出,总体上复合片键的查询性能优于单片方案,对于复合片键值{ItemId,ItemName}的查询性能较差的原因,主要是由于该方案下数据分布的均衡性较差。复合片键{Longitude,Latitude}和{ObserveTime,Longitude,Latitude}的查询性能明显好于其他方案;其中{ObserveTime,Longitude,Latitude}方案不仅平均查询时间较短,且查询性能的稳定性表现较好,后期数据量增大后性能波动较小,能较快实现收敛。

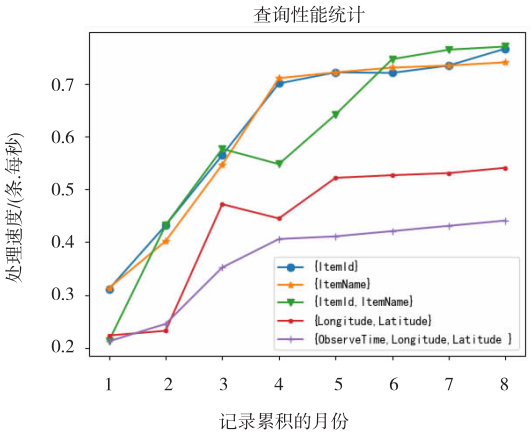

图 8 查询性能统计分析结果图

3.4 缓存模块对查询性能的影响

设置两组对比实验:一组是不使用缓存的分布式架构设计(架构一),另一种是在原来 2.1 节提出的增加了内存型缓存的分布式存储架构(架构二),通过计算文档记录平均查询时间来衡量两种架构不同的性能表现,见表 5。

表 5 两种不同架构模型的平均文档查询时间

组 别	文档记录查询的平均耗时测试(单位:秒)							
	1	2	3	4	5	6	7	8
架构一(不采用缓存)	0.132	0.332	0.561	0.732	0.741	0.756	0.761	0.792
架构二(增加内存型缓存)	0.134	0.325	0.422	0.581	0.582	0.595	0.601	0.612

图 9 为两种架构模型的文档查询的平均耗时对比图。由图 9 可知,在数据量较少时,两种架构模型的平均查询耗时差距不大,随着数据量的增大,架构二(缓存模型)的性能优势逐渐凸显出来,平均查询时间明显低于架构一,且平均查询耗时也趋于稳定;因此可以看出缓存模型能够较好地改善遥感数据的查询性能。

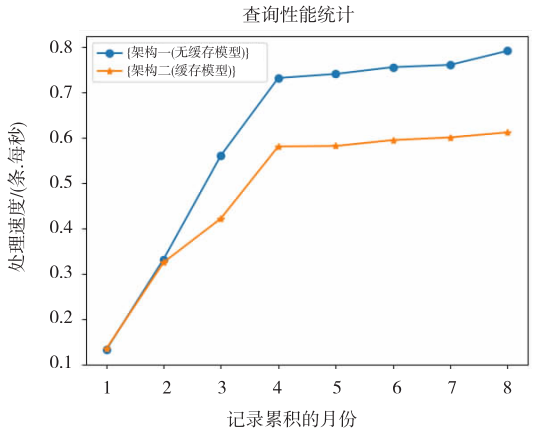

图 9 两种架构模型的平均耗时对比

4 结束语

本文基于非关系型数据库 MongoDB,提出了一种海量遥感数据的分布式存储方案,通过引入内存型缓存提高遥感数据的查询和存储效率,使用一种针对遥感数据特点的分片数据方案;并通过实验验证了该存储方案对海量遥感数据的存储具有一定的优越性。该方案具有一般性意义,能够运用到其他场景中;下一步工作将进一步对这种存储方案进行优化,以提高其整体性能。

参 考 文 献:

[1] 孙敬杰,曾李阳,张尧.基于 MongoDB 的 Web 空间数据存储与管理研究[J].测绘,2017,40(2):72-74,89.

[2] 祁兰.基于 MongoDB 的数据存储与查询优化技术研究[D].南京:南京邮电大学,2016.

[3] 高锐.基于 MongoDB 的黑河流域时空数据云存储关键技术研究[D].兰州:兰州大学,2016.

[4] 徐旭东,郭瑞,文瑞洁.分布式下 MongoDB 对激光点云的存储和处理研究[J].计算机应用与软件,2017,34(2):71-73,168.

[5] 潘永华,闭应洲,符云琴.基于 MongoDB 的医学图像管理技术研究[J].广西师范学院学报:自然科学版,2017,34(2):54-59.

[6] 王震,余洋,张志强,等.基于 MongoDB 的遥感影像存取方案的探讨与实现[J].测绘地理信息,2017,42(3):

39-43.

[7] 田帅.一种基于 MongoDB 和 HDFS 的大规模遥感数据存储系统的设计与实现[D].杭州:浙江大学,2013.

[8] 赖积保,罗晓丽,余涛,等.一种支持云计算的遥感影像数据组织模型研究[J].计算机科学,2013,40(7):80-83,115.

[9] 秦强,王晏民,黄明.基于 MongoDB 的海量遥感影像大数据存储[J].北京建筑大学学报,2015,31(1):62-66.

[10] 梁海.MongoDB 数据库中 Sharding 技术应用研究[J].计算机技术与发展,2014,24(7):60-62,67.

[11] 冯东煜,朱立谷,肖子达,等.一种 MongoDB 集群数据布局优化方法研究[J].计算机工程与应用,2017,53(17):77-84.

[12] 雷德龙,郭殿升,陈崇成,等.基于 MongoDB 的矢量空间数据云存储与处理系统[J].地球信息科学学报,2014,16(4):507-516.

[13] 江颖,邬群勇,唐曙光,等.基于 MongoDB 的闽西客家文化数据存储设计与分析[J].测绘工程,2016,25(3):56-60.

[14] 张路路.基于 MongoDB 的大数据存储方法研究与应用[D].成都:成都理工大学,2015.

[15] 邱儒琼,郑丽娜,李兵.基于 MongoDB 的电子地图瓦片数据存储和服务研究[J].地理空间信息,2014,12(6):155-157,6.

[16] 张尧,甘泉,刘建川.基于 MongoDB 的地理信息共享数据存储模型研究[J].测绘,2014,37(4):147-150,172.

A Method of Storing Remote Sensing Data Based on MongoDB Cluster

LI Hongzhi[1], *LI Xianlan*[2]

(1. College of Computer and Information Engineering, Chuzhou University, Chuzhou 239000, China;
2. College of Photonic and Electronic Engineering, Fujian Normal University, Fuzhou 350000, China)

Abstract: With the development of remote sensing technology, the quantity and quality of remote sensing data are gradually improved. How to effectively store and manage these data has become the focus of current research. Using MongoDB database, a storage scheme of remote sensing data based on the cluster of MongoDB is proposed, and an architecture model based on this scheme is implemented; Finally, The storage architecture is tested, and the results show that the proposed architecture can improve the performance of remote sensing data storage and retrieval, and is suitable for remote sensing data storage.

Key words: remote sensing data; MongoDB cluster; shard key strategy; Hash consistency; storage model