

MongoDB 负载均衡算法优化研究

陈敬静¹, 马明栋², 王得玉²

(1. 南京邮电大学 通信与信息工程学院, 江苏 南京 210003;
2. 南京邮电大学 地理与生物信息学院, 江苏 南京 210003)

摘要:随着 Web2.0 网络应用的兴起和大数据技术的发展,传统的关系型数据库(ORDBMS)已经难以满足海量数据的存储需求。非关系型数据库(NosQL)因其高扩展性、高伸缩性、高可用性和容错性等特点,得到了越来越多的应用。作为一种新兴的 NosQL 数据库, MongoDB 数据库因具有模式自由、易于扩展、故障自动恢复、支持自动分片等特点,被广泛应用于大数据处理与分析中。文中首先介绍了 MongoDB 自动分片架构原理和实现机制,然后分析了 MongoDB 自带的负载均衡算法,其虽能使各个节点数据量达到平衡,但没有考虑各个节点的负载均衡。为了解决节点的负载平均问题,在原算法基础上提出了一种基于节点实时负载的负载均衡改进算法,改进算法的主要思想是引入节点负载指数作为 chunk 块迁移的一个判断条件。通过搭建测试环境并进行实验,验证了改进的负载均衡算法可以有效地均衡分片中的数据,提高集群的并发读写性能,从而证明了算法的有效性。

关键词:非关系型数据库; MongoDB; 自动分片; 负载均衡; 节点负载

中图分类号: TP393.0

文献标识码: A

文章编号: 1673-629X(2020)03-0088-05

doi: 10.3969/j.issn.1673-629X.2020.03.017

Research on Optimization of MongoDB Load Balancing Algorithm

CHEN Jing-jing¹, MA Ming-dong², WANG De-yu²

(1. School of Telecommunications & Information Engineering, Nanjing University of
Posts and Telecommunications, Nanjing 210003, China;

2. School of Geographical and Biological Information, Nanjing University of Posts and Telecommunications,
Nanjing 210003, China)

Abstract: With the rise of Web2.0 network applications and the development of big data technology, the traditional relational database (ORDBMS) has been difficult to meet the storage requirements of massive data. Non-relational database (NosQL) has been applied more and more because of its high expansibility, high scalability, high availability and fault tolerance. As a new NosQL database, MongoDB is widely used in big data processing and analysis because of its free mode, easy expansion, automatic fault recovery and automatic fragmentation. We firstly introduce the principle and implementation mechanism of MongoDB automatic fragmentation architecture, and then analyze the load balancing algorithm of MongoDB. Although it can balance the data volume of each node, the load balancing of each node is not considered. In order to solve the load averaging problem of nodes, an improved load balancing algorithm based on real-time load of nodes is proposed. Its main idea is to introduce the load index of node as a judgment condition for chunk block migration. By setting up the test environment and conducting experiments, it is verified that the improved load balancing algorithm can effectively balance the data in the fragment and improve the concurrent read-write performance of the cluster, which proves its effectiveness.

Key words: NosQL; MongoDB; auto-sharding; load balancing; node balancing

0 引言

随着通信技术、互联网、云计算的飞速发展,互联

网浪潮已经到达,接入到互联网中的设备呈指数增长,越来越多的数据以不同内容和形式涌现出来,大数据

收稿日期: 2019-04-08

修回日期: 2019-08-09

网络出版时间: 2019-12-05

基金项目: 江苏省自然科学基金-青年基金项目(BK20140868)

作者简介: 陈敬静(1996-),女,硕士研究生,研究方向为图像处理与图像通信;马明栋,博士,教授,研究方向为地理信息系统平台软件设计与开发等;王得玉,博士,副教授,研究方向为水环境遥感、GIS软件设计与开发。

网络出版地址: <http://kns.cnki.net/kcms/detail/61.1450.TP.20191205.1113.034.html>

时代正式到来。大数据时代最主要的特征就是数据种类与数量的繁多,除了结构化数据还有半结构化数据以及非结构化数据,这要求数据存储系统必须能够处理高并发问题,同时还要易于扩展^[1]。尽管在主流应用场景中仍然使用传统关系型数据库,但面对海量的数据,它很难满足海量数据的存储需求,已经无法满足人们日益增长的需求。为了解决如何存储和处理海量数据的问题,非关系型数据库(NoSQL)应运而生^[2]。MongoDB 也是 NoSQL 的一种,因其非常适合处理海量数据和高并发而得到大量应用。文中通过研究 MongoDB 的自动分片原理,提出一种改进的基于节点实时负载的负载均衡算法^[3],以有效解决其自身算法存在的部分问题。

1 MongoDB 的自动分片

1.1 MongoDB 简介

MongoDB 是 10gen 公司使用 C++编写开发的基于分布式文件存储的开源 NoSQL 数据库系统,由于其性能高效、功能丰富,在生产中得到了广泛应用。MongoDB 除了具有 NoSQL 数据库的相关特性外,还具有自动分片、集群扩展、单点故障自动恢复、复杂查询等优点,非常符合存储海量的半结构化或非结构化数据^[4]。MongoDB 将数据存储为一个文档,数据结构由键值对组成,其文档的数据结构非常松散,是类似于 JSON 的 BSON 格式^[5],存储效率高。它是一个面向集合的,模式自由的文档型数据库,相较于传统的关系型数据库,在面对海量数据的挑战时更加有优势,其主要功能特性有:面向集合存储、模式自由、容易扩展、支持复制和数据恢复、支持动态查询、支持完全索引以及自动处理分片等^[6]。

1.2 分片介绍

分片(Sharding)是指将内存中的数据拆分成不同的块,分别存储到不同机器上的过程。通过分割数据到不同的服务器,让数据集的不同部分分别由不同的服务器负责,使得单个机器上的请求数得到减少,系统总负载得到提高,总存储空间也得到提高^[7]。分片是数据库系统扩展的必然产物,而不是某个特定数据库软件附属的功能,分片能在一定程度上决定系统性能的优劣。MongoDB 采用自动分片(Auto-Sharding)机制,如图 1 所示。

自动分片技术一般用于自动配置、监控和数据转移,当数据量大到服务器的磁盘、内存难以负担时,自动分片技术可以自动平衡负载和数据分布的变化,提升系统的扩展性能。此外,它还提供无单点故障自动恢复、自动故障转移以及动态添加额外服务器等技术,为提升系统性能提供了很大帮助。

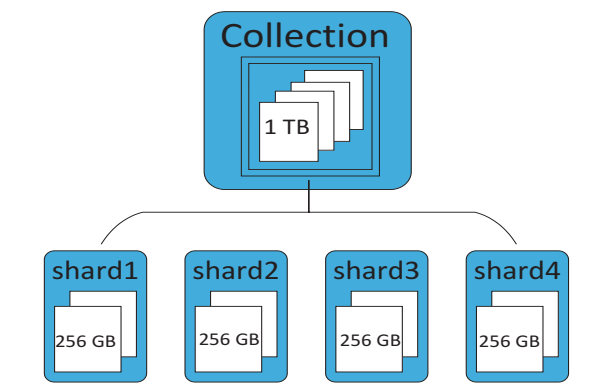


图 1 分片原理

1.3 自动分片集群架构

MongoDB 的自动分片集群架构如图 2 所示,主要包括分片服务器、路由服务器以及配置服务器,这三种服务器负责的功能如下:

(1)Mongos:路由服务器,负责将读取和写入的请求从应用程序路由到分片。集群通过 Mongos 连接客户端和服务器,当客户端向数据库发送更新或查询操作请求,Mongos 接收请求并聚合,然后发送给分片服务器,它并不存储数据,只传递请求^[8]。

(2)Config Server:配置服务器,存储集群的配置信息以及分片与数据的对应关系,运行集群时向路由服务器提供配置信息和对应关系。一般 MongoDB 自动分片集群中配有多个配置服务器,每个配置服务器中都保存了所有信息,防止信息丢失。

(3)Shard:分片节点,用于存储数据。在架构中,一个片内可以有多个 Mongos 服务器,每个服务器中存放的数据都相同,主服务器只有一个,其他均为从服务器。存储数据的部件是分片节点,为了获得高扩展性和数据一致性,分片常与副本集(Replica Set)同时使用,防止该数据片单点故障^[9]。

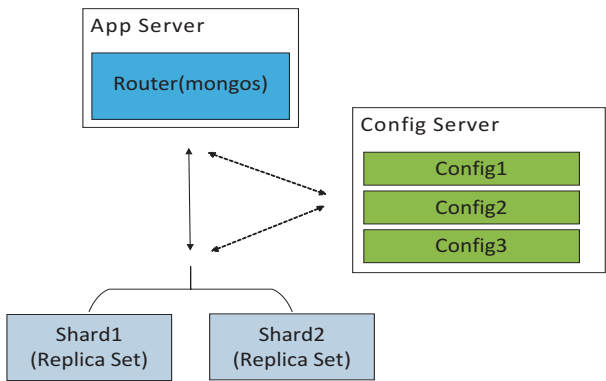


图 2 自动分片集群架构

一般情况下,当用户向数据库发送操作请求时,Mongos 会解析数据库的分片 shard key(片键)规则,在存储元数据的配置节点配置服务器中查找相关信息,找到对应的分片后将请求转发到正确的片上,对客户端发送来的请求进行响应,最后 Mongos 将获取到

的结果发送给应用程序^[10]。将数据片段与应用程序分离是 MongoDB 分片技术中最独特的地方,使用这种分片机制,用户可以在不更改程序的前提下,实现对数据库系统的扩展。

2 MongoDB 负载均衡算法

2.1 Chunk 块拆分

MongoDB 将分片服务器内部数据分为 chunks,不同 chunk 块代表这个分片服务器内部的部分数据,由指定片键的某一连续范围内的文档组成。当 chunk 块过大时,MongoDB 后台进程会计算每个 chunk 块的大小并选择拆分点,根据拆分点将该 chunk 块切分成更小的 chunk 块,避免 chunk 块过大的情况^[11]。在 MongoDB 中,负责数据迁移的工具就是均衡器 (balancer),balancer 是一个后台进程,负责 chunk 块的迁移,从而均衡各个 shard server 的负载。拆分 chunk 块最重要的两点是选择合适的拆分点和不同 chunk 块

所占用的空间基本相等。

2.2 负载均衡算法分析

随着 MongoDB 中的数据越来越多,分片中 chunk 块的数量也越来越多,每个分片服务器上 chunk 块的个数也不相同,且差异越来越大。在 MongoDB 中,默认的负载均衡算法认为 chunk 块数量相当即负载均衡。负载均衡的实现主要来自于其内部负载均衡器 (balancer) 进程的运行,均衡器周期性地检查各分片,当分片间 chunk 块的数量差到达迁移阈值 (默认为 8),均衡器启动自动数据迁移,将数据从包含最多 chunk 块的片上迁移到 chunk 块最少的片上,直到 chunk 块的数量差不大于 2 为止^[12]。数据迁移以 chunk 块为单位进行迁移,最初默认大小为 64 M,但随着数据量的逐渐增大,最终每个块的数据量会达到 200 M^[13]。

通过阅读源码并对源码进行分析,MongoDB 负载均衡算法的流程如图 3 所示。

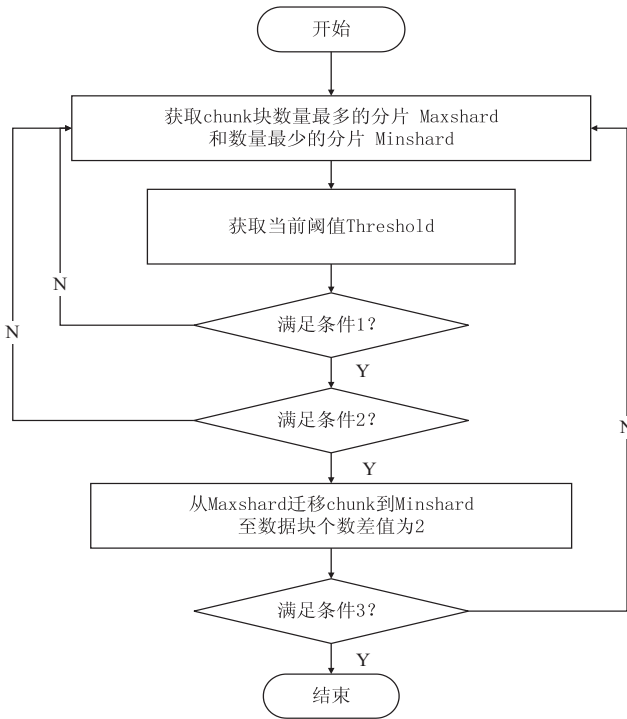


图 3 负载均衡算法流程

- 条件 1: Minshard 未达存储上限且不存在写回队列;
- 条件 2: $\text{Maxshard} - \text{Minshard} \geq \text{Threshold}$;
- 条件 3: 负载均衡器仍然开启。

2.3 负载均衡算法的争议

一直以来,MongoDB 的负载均衡算法在实际应用中仍然存在许多问题。这个均衡器只考虑了数据的存储平衡,而没有考虑负载均衡^[14]。节点的负载取决于其配置及接受任务的轻重,尽管分片节点上的 chunk 块数量相同,但在选择目标分片时,没有考虑分片节点

上的负载不相同这一问题,因此得到的结果也可能不是最好的,所以很有必要对 MongoDB 的负载均衡算法进行改进。

3 基于节点实时负载的负载均衡算法

3.1 基于节点的均衡算法思想

通过前面的分析了解到 MongoDB 的负载均衡算法并不是十分完善,因此文中在原始算法的基础上提出一种基于节点实时负载的负载均衡算法。改进的算法将节点的实时负载情况作为判断条件,在节点上增

加负载代理,在负载均衡器上增加负载监视器,通过负载代理监测各节点的负载情况并将数据发送给负载监视器,均衡器将节点负载指数作为确定源分片和目的分片的一个指标。

将节点 i 的负载代理检测到的 CPU 占有率、内存使用率以及网络带宽占有率分别记为 C_i 、 M_i 以及 N_i ,CPU、内存和网络带宽的权值分别设为 i_1 、 i_2 和 i_3 ,则节点 i 的负载指数 I_{load} 可表示为:

$$I_{load} = i_1 \times C_i + i_2 \times M_i + i_3 \times N_i \tag{1}$$

$$i_1 + i_2 + i_3 = 1 \tag{2}$$

假设自动分片集群架构中有 n 个分片,则平均负载 Ave_{load} 为:

$$Ave_{load} = \frac{\sum_{i=1}^n I_{load}}{n} \tag{3}$$

最大的节点负载为 Max_{load} ,设定阈值 γ (一般默认为 8),若

$$Max_{load} - Ave_{load} \geq \gamma \tag{4}$$

则此节点过载。改进的基于节点的均衡算法在均

衡 chunk 块数量的同时,还均衡了分片节点上的负载,可以有效提升原算法的性能,解决数据分布不均的问题^[15]。

3.2 基于节点的均衡算法设计

基于节点实时负载的负载均衡算法具体设计如下:

(1)负载代理周期性地遍历各分片,获取节点负载信息并发送给负载监视器。

(2)负载监视器计算出平均负载,确定负载最大的分片、chunk 块数最多的分片和存储即将超过上限的待移除分片,将既不是待移除分片且 chunk 块数也不是最多的分片列入最小分片的候选列表。

(3)若分片块数差超过设定阈值,则确定源分片为 chunk 块数最多的分片;若分片块数差满足条件,但存在待移除分片,则确定源分片为待移除分片;若前两个条件都不满足但根据式(4)计算得出存在过载分片,则源分片为负载最大的分片;若以上条件都不满足,则不需要迁移。

算法流程如图 4 所示。

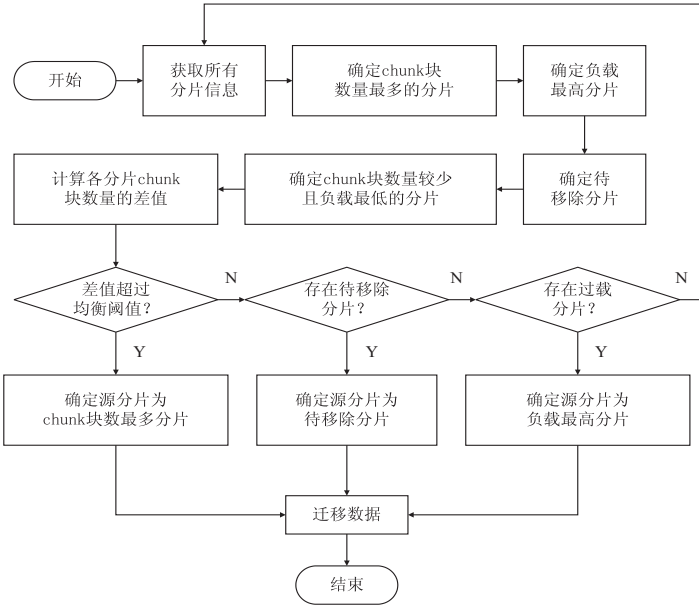


图 4 基于节点实时负载的负载均衡算法流程

4 算法性能评估

4.1 测试环境

测试环境基于 MongoDB 的自动分片集群,由 3 台机器构成,每台机器内存都为 4 GB,操作系统为 Linux Redhat,MongoDB 版本为 3.4.0。集群包含 3 个分片,每个分片由一个副本集组成,每个副本集包含 1 个 primary 节点,2 个 secondary 节点和 1 个 arbiter 节点。结合实际运行情况,设置 CPU 占有率和内存使用率权重占比较大,网络带宽占有率权重较小, i_1 、 i_2 和 i_3 分别为 0.2、0.7 和 0.1,阈值 γ 设置为 13%。

4.2 平台搭建

- (1)配置副本集。

```
> cfg = { _id: "shard1", members: [ { _id: 0, host: "192.168.169.128:40007" }, { _id: 0, host: "192.168.169.128:40007" }, { _id: 0, host: "192.168.169.128:40007" } ] }
> rs.initiate( cfg );
```
- (2)配置 Config Server。

```
mongod -- configsvr -- dbpath =/home/mongod/data/
mongo.conf
```
- (3)配置 Mongos。


```
mongos -f /home/mongod/mongos/mongo.conf
```

(4) 配置 Shard。

```
>db. runCommand ( { addshard “ shard1/192. 168. 148. 61; 40007,192. 168. 148. 63;40007,192. 168. 148. 65 ;40007” } )
```

4.3 算法测试结果

首先,测试集群的并发写入功能。为了保证数据总量相等,均插入一百万条数据,在不同并发数下,新旧算法每秒可写入的数据记录如表 1 所示。

表 1 新旧算法并发写入性能数据统计

算法	1	10	25	50	100
原算法	11 700	8 700	5 900	4 200	2 800
新算法	11 800	8 850	6 100	4 570	3 300

在并发数和记录不变的情况下,测试集群的并发读取性能,新旧算法每秒读取的数据记录如表 2 所示。

表 2 新旧算法并发读取性能数据统计

算法	1	10	25	50	100
原算法	9 300	13 400	19 000	21 800	23 000
新算法	8 900	14 000	21 000	23 000	25 500

从实验数据结果可以看出,在并发数较小的时候,新算法的读写性能并没有明显优于原算法,甚至可能会低于原算法,但随着并发数的增加,新算法明显优于原算法。这是因为改进的基于节点实时负载的负载均衡算法将节点负载作为一个考虑条件,当数据量不够大时,计算节点的负载情况资源利用率低,影响了系统性能,而在大数据及大并发的情况下,应用新算法之后的读写性能明显优于原算法,提高了集群的并发读写能力。

5 结束语

首先介绍了 MongoDB 自动分片的原理,然后分析了其负载均衡算法的缺点,针对分片间分配数据不均匀的问题,提出了一种基于节点实时负载的负载均衡算法。接着搭建了测试环境,针对数据的并发读写性能与原算法做一个对比实验,通过实验得出,该算法在数据的读写均衡上得到了明显优化,提高了集群的并发读写性能,证明了算法的有效性。

参考文献:

[1] 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.

[2] 马 骏,张飞龙. 基于 MongoDB 的遥感规格化数据云平台的设计与实现[J]. 计算机时代,2016(2):49-52.

[3] 赵立斌. 分布式 MongoDB 集群高可用性的研究和性能优化[D]. 成都:电子科技大学,2016.

[4] CBODOROW K. MongoDB 权威指南[M]. 邓 强,王明辉,译. 北京:人民邮电出版社,2016.

[5] KANG Y,PARK I,RHEE J,et al. MongoDB-based repository design for IoT-generated RFID/sensor big data[J]. IEEE Sensors Journal,2016,16(2):485-497.

[6] 伍业雄. 基于 MongoDB 和 Spark 的计量大数据处理技术方案及应用[J]. 信息与电脑,2018(7):141-144.

[7] 李朝奎,严雯英,杨 武,等. 三维城市模型数据划分及分布式存储方法[J]. 地球信息科学学报,2015,17(12):1442-1449.

[8] CURINO C,JONES E,ZHANG Y,et al. Schism:a workload-driven approach to database replication and partitioning[J]. Proceedings of the VLDB Endowment,2015,3(1):48-57.

[9] 郭远威. 大数据存储 MongoDB 实战指南[M]. 北京:人民邮电出版社,2015:38-41.

[10] ABBES H,GARGOURI F. MongoDB-based modular ontology building for big data integration[J]. Journal on Data Semantics,2017,7(2):1-27.

[11] 何杭锋. 基于 FODO 算法 MongoDB 自动分片的改进[J]. 计算机技术与发展,2013,23(7):127-130.

[12] GILLEY C,RINGDAHL J E. The effects of preference and token reinforcement on sharding behavior exhibites by children with autism spectrum disorder[J]. Research in Autism Spectrum Disorders,2014,8(11):1425-1433.

[13] FORNES O, GHEORGHE M, RICHMOND P A, et al. MANTA2,update of the Mongo database for the analysis of transcription factor binding site alterations[J]. Scientific Data,2018,5:180141.

[14] 邓志飞,应良佳,王军威. 基于 IODA 算法 MongoDB 负载均衡的改进[J]. 现代电信科技,2013(7):9-13.

[15] 卢至彤,李 翀,柯 勇,等. 一种 MongoDB 应用优化策略[J]. 计算机系统应用,2017,26(5):55-61.