

引用格式:冯国军,贺占庄,吕瑛. 基于 DBAF 算法的 MongoDB 负载均衡策略[J]. 微电子学与计算机, 2021, 38(2): 52-55.
[FENG G J, HE Z Z, LV Y. Research on data sharding and load balancing algorithms based on mongoDB[J]. Microelectronics & Computer, 2021, 38(2): 52-55.]

基于 DBAF 算法的 MongoDB 负载均衡策略

冯国军, 贺占庄, 吕 瑛

(西安微电子技术研究所, 陕西 西安 710054)

摘 要: 面对高速发展的物联网产业所带来的数据爆发式增长的问题, NoSQL 能比较好的处理海量数据. 本文选取 NoSQL 家族中比较典型的 MongoDB 开展了研究. 分析了 MongoDB 的工作机制, 针对当前其负载均衡算法的未考虑数据块访问频率的问题, 提出了一种基于数据块访问频率 (DBAF) 算法的负载均衡策略. 实验结果表明该新算法可以有效提高系统的并发读写性能.

关键词: MongoDB; 集装箱数据; 自动分片; 负载均衡

中图分类号: TP311

文献标识码: A

文章编号: 1000-7180(2021)02-0052-04

Research on data sharding and load balancing algorithms based on mongoDB

FENG Guo-jun, HE Zhan-zhuang, LV Ying

(Xi'an Microelectronics Technology Institute, Xi'an 710054, Shaanxi, China)

Abstract: Faced with the problem of explosive growth of data brought by the rapid development of the Internet of things industry, NoSQL can better handle massive data. In this paper, the typical MongoDB database in NoSQL is selected for research. The storage architecture of MongoDB is analyzed, and the auto-sharding mechanism of MongoDB is studied. Aiming at the shortage of current MongoDB load balancing algorithm, a load balancing strategy based on data block access frequency (DBAF) algorithm is proposed. The experiments show that the new load balancing strategy can effectively improve the concurrent read-write performance of the system.

Key words: MongoDB; container data; auto-sharding mechanism; Load balancing

1 引言

随着物联网技术在全球范围内的推广, 海量非结构化的物联网数据也随之而来. 传统数据库在面对这些数据时查询效率低, 并且对一些稀疏松散的数据进行建表非常困难, 会极大浪费硬件存储资源; 不仅如此, 面对当今带宽逐渐增大的网络系统, 对用户的访问带来的就是高并发的操作^[1]. 这使得对传统关系型数据库的存储结构设计更为困难.

近些年来, 国内外众多学者主要研究方向主要集中在利用非关系型数据库 NoSQL 管理大型数据

集^[2]. 本文以 NoSQL 家族中典型的 MongoDB 数据库负载均衡策略为切入点, 提出一种基于数据块访问频率 (DBAF) 算法的负载均衡策略, 最终验证了新算法可以有效均衡数据分布从而提高整个集群系统的并发读写性能.

2 MongoDB

2.1 MongoDB 分布式集群架构

MongoDB 分布式集群架构由众多副本集、路由服务器以及配置服务器构成^[3].

其结构如图 1 所示. 副本集中主节点服务器参

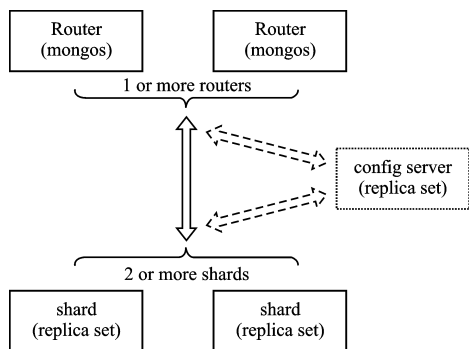


图 1

与用户的访问,次节点作为主节点的备用节点增加整个系统的鲁棒性,配置节点储存着副本集以及路由服务器的对应关系。

2.2 MongoDB 自动分片机制

MongoDB 分片机制是指按照预设定的分片键将数据分割到不同的分片集群中。

分片机制是为了应对越来越大的数据量对单台服务器造成压力过大而提出的。面对现在信息数据的爆炸,对服务器的硬件要求也越来越高,当单台服务器的性能到达极限时,就需要引进其他“兄弟”帮帮忙分担数据存储的压力,也就是分片。这样的水平扩展不仅可以增强整个系统的扩展性和可靠性,也可以解决高并访问的问题。

3 MongoDB 负载均衡策略

在分片键确定之后,MongoDB 会通过 mongos 路由器对数据进行分片操作。在每个分片集中数据实际是以逻辑块(chunk)进行存储的,在达到某一条件时,MongoDB 会通过 mongos 里面集成的 Balance 服务对整个存储系统进行负载均衡的操作。Balance 进程会定期扫描各个分片中存储块的数量,当包含两个分片中块数量的差值达到某一阈值时,就会触发负载均衡机制,对数据块进行迁移操作达到数量均衡的效果。但是问题在于系统仅仅只考虑了数据块的不均衡却没有考虑数据块的访问频率的不均衡。

3.1 算法思想

MongoDB 对数据的操作都是以块为单位,每次 MongoDB 在进行增读改删(CRUD)操作前都是向配置服务器发送请求以获取将要操作的目标块的信息,因此可以记录下访问每个块的时间间隔,用访问时间间隔去衡量该数据块的活跃程度,访问时间间隔越小表示该块工作量越大,就是一个活跃数据

块。因此,当开始对数据进行负载均衡时,在原本只考虑数据块量负载的基础上,再将平均访问时间间隔作为一个因素考虑进去,可以一定程度上解决上述所提到的问题。

3.2 平均时间间隔的计算

假设每个块的平均时间访问间隔 $Avg_Time_C_i$, 每个块进行增读改删 CRUD 操作的次数分别为 C_i, R_i, U_i, D_i , 对应的累计的时间间隔分别为 $C_AllTime_C_i, R_AllTime_C_i, U_AllTime_C_i, D_AllTime_C_i$, 且在最后一次操作到计算平均时间间隔这段时间 $Wait_AllTime_C_i$ 也应该加在操作时间之中,则有如下公式(1):

$$Avg_Time_C_i = C_AllTime_C_i / C_i + R_AllTime_C_i / R_i + U_AllTime_C_i / U_i + D_AllTime_C_i / D_i + Wait_AllTime_C_i \quad (1)$$

但是对于每一个块来说,增读改删操作对系统带来的负担是不同的,因此每一个操作都应该对应一个权重因子,权重因子矩阵如下所示:

$$WEIGHT = (\omega_c, \omega_R, \omega_U, \omega_D, \frac{\omega_c + \omega_R + \omega_U + \omega_D}{4})^T \quad (2)$$

因此修正后的平均时间间隔公式如下所示:

$$Avg_Time_C_i = (C_AllTime_C_i / C_i, R_AllTime_C_i / R_i, U_AllTime_C_i / U_i, D_AllTime_C_i / D_i, Wait_AllTime_C_i) \cdot WEIGHT \quad (3)$$

实际情况中,当 CRUD 操作其中一项的累计时间间隔为 0 即当前无任何该操作,对应的时间间隔矩阵的项应置为 0,例如若在本次平均时间间隔计算中无删除操作,平均时间间隔应为:

$$Avg_Time_C_i = (C_AllTime_C_i / C_i, R_AllTime_C_i / R_i, U_AllTime_C_i / U_i, 0, Wait_AllTime_C_i) \cdot WEIGHT$$

假设 MongoDB 每个片均包含 N 个块,因此每个片的平均访问时间间隔为 $Avg_Time_S_i$,

$$Avg_Time_S_i = \sum_{i=0}^N \frac{Avg_Time_C_i}{N} \quad (4)$$

由于平均访问时间间隔和分片服务器活跃程度载成负相关,即平均时间间隔越短则该数据块越活跃,为了在后面与数据块量负载度量统一,需要取平均时间间隔的倒数,即表示每秒分片服务器的平均操作数:

$$Avg_Oprate_S_i = \frac{1}{Avg_Time_S_i} \quad (5)$$

当集群开启负载均衡机制时,将每个片的平均访问时间间隔作为该片数据访问冷热程度的因素,

可以一定程度上即满足数据存储的量负载均衡,也可以满足对数据块的访问负载均衡。

3.3 基于热负载和量负载的负载均衡算法

本小节描述了基于数据块访问频率的负载均衡策略,主要由以下三个部分组成:第一,确定待迁移的源片以及迁移的目标片;第二,选择源片中待迁移的数据块;第三,选择适当的迁移时机以及迁移数据。

首先,选择待迁移的源片与目标片,步骤如下所示:

(1) MongoDB 内置的 Balance 程序收集当前各个分片服务器上 chunk 的信息包括数量以及访问频率。

(2) 确定包括 chunk 数量对多的分片服务器 Chunk_Max 及 chunk 数量最少的分片服务器 Chunk_Min. 如果二者只差达到某一阈值 Threshold, 则说明此时需要进行数据迁移, 将所有与 Chunk_Max 的量负载差值大于 Threshold 的分片服务器组成一个最小量负载分片集 MIN_NUM_S, 在这个集合中, 通过热负载和量负载的判定策略来决定目标分片, 具体策略如下:

对于每个分片服务器具有量负载和热负载两个参数, 热负载通过中平均访问时间间隔 Avg_Oprate_S_{ii} 来表示, 量负载则由分片数 Chunk_Num_S_i 来表示, 假设最小量负载分片集 MIN_NUM_S 中有 K 个分片服务器, 则有:

$$\text{MIN_NUM_S} = \begin{bmatrix} \text{Chunk_Num_S}_1 & \text{Avg_Oprate_S}_1 \\ \vdots & \vdots \\ \text{Chunk_Num_S}_K & \text{Avg_Oprate_S}_K \end{bmatrix}$$

因为热负载和量负载的水平指标相差可能会比较大, 且热负载和量负载单位不同, 无法直接相加去表征分片服务器的总负载, 因此要对分片服务器数据进行归一化处理. 这样的好处是可以使量负载和热负载对分片服务器的影响同趋化, 然后相加就可以比较准确的衡量分片服务器的负载情况. 归一化后如下所示:

$$\text{MIN_NUM_S} = \begin{bmatrix} \frac{\text{Chunk_Num_S}_1}{\sum_1^K \text{Chunk_Num_S}_i} & \frac{\text{Avg_Oprate_S}_1}{\sum_1^K \text{Avg_Oprate_S}_i} \\ \vdots & \vdots \\ \frac{\text{Chunk_Num_S}_K}{\sum_1^K \text{Chunk_Num_S}_i} & \frac{\text{Avg_Oprate_S}_K}{\sum_1^K \text{Avg_Oprate_S}_i} \end{bmatrix}$$

最小量负载集矩阵的每一行表示一个分片服务器的量负载 N 和热负载 H 的表征参数, O 和总负载成正相关, 即平均操作数越大, 热负载越大; N 和量负载也成正相关, 即分片服务器包含块数量越多, 则量负载越大。

$$\text{MIN_NUM_S} = \begin{bmatrix} N_1 & , & O_1 \\ \vdots & \vdots & \vdots \\ N_k & , & O_k \end{bmatrix}$$

二者之和就可以表示当前分片服务器的总负载情况, 可得到最小量负载分片集的总负载矩阵:

$$\text{MIN_LOAD_S} = \begin{bmatrix} N_1 + O_1 \\ N_2 + O_2 \\ \vdots \\ N_K + O_K \end{bmatrix} \quad (6)$$

将最小量负载集的总负载排序后, 找出最小负载的分片服务器即为目标片 T, 负载最大的分片即为源片 S.

(3) 确定目标块. 在确定目标片 T 之后, 则需要确定目标片的哪一块作为目标块, 主要有下面两种情况:

1. S(AVG_Time_Si) > T(AVG_Time_Si), 说明源片数据的活跃程度较低, 因此选择将源片中活跃程度较低的数据块迁移到目标分片服务器上, 避免增大目标片的访问频率. 2. S(AVG_Time_Si) < T(AVG_Time_Si), 说明源片数据的活跃程度较高, 因此选择将源片中活跃程度较高的数据块迁移到目标分片服务器上, 减少源片服务器的访问频率.

(4) 将步骤三确定的数据块进行备份, 将副本迁移到目标分片, 更新是整个集群系统的配置信息, 使所有集群信息同步.

(5) 跳至第一步, 直到 Chunk_Max 和 Chunk_Min 的量负载之差小于 Threshold, 则本次负载均衡结束.

图 2 展示了算法的详细流程.

4 性能测试及分析

本文基于 MongoDB 数据库搭建的存储系统性能要求为满足 100 000 台终端设备运行. 下面是根据要求计算每秒最大并发数的过程.

4.1 平台的搭建

本次实验配置三台 MongoDB 分片服务器作为集群, 每台服务器都是相同的硬件配置: 8 GB 内存, CPU 频率 2.6 GHz. 每台服务器带宽是 100

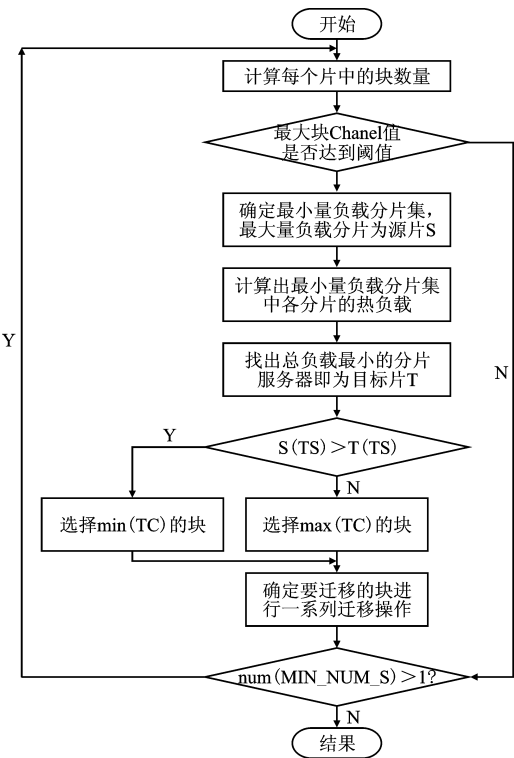


图 2 算法流程

Mbps. MongoDB 版本是 3.11.0, 自动分片集群 3 个分片, 每个分片都包含副本集. 每个副本集包含 3 个节点, 一个主节点, 一个次节点, 一个仲裁节点.

(1)配置副本集;配置其他分片服务器以及配置服务器同上;

(2)配置 MongoDB 配置服务器 config;

(3)连接 mongos 节点, 添加分片: db.runCommand({addshard:"shard1/127.0.0.1:13211,127.0.0.1:13212,127.0.0.1:13213"})
db.runCommand({addshard:"shard2/127.0.0.1:13221,127.0.0.1:13222,127.0.0.1:13223"})
db.runCommand({addshard:"shard2/127.0.0.1:13231,127.0.0.1:13232,127.0.0.1:13233"})

4.2 算法并发性测试

将新的基于数据块热负载和量负载的负载均衡算法在 MongoDB 环境下实现. 为保证和实际情况一样, 随机产生 10 000 000 条数据, 在相同数量的数据下去对比新旧算法不同并发数下的系统读写性能情况. 根据系统反应时间反映具体性能, 单位为 s:

集群系统在不同并发数和不同算法下插入数据的时间使用情况如表 2 所示. 以下数据均为相同条件下取 10 次测试的平均值所得.

表 2 并发写性能比较

并发数	1	10	25	50
旧算法	580	823	1 368	1 892
新算法	564	777	1 272	1 795

新旧两种算法在不同并发数下的读取速率如下所示. 以下数据均为相同条件下取 10 次测试平均值所得.

表 3 并发读性能比较

并发数	1	10	25	50
旧算法	823	729	500	466
新算法	819	700	489	443

5 结束语

本文首先以 MongoDB 为研究对象, 展开了对 MongoDB 的工作原理的阐述, 针对 MongoDB 现有的负载均衡策略未考虑对各个分片服务器访问频率的问题, 基于数据块访问频率 (DBAF) 算法的负载均衡策略, 并验证了新算法可以明显提升集群系统的并发读写性能.

由于 CRUD 四种数据访问操作给系统带来的资源开销是不同的. 因此下一步的工作应该对 CRUD 四种操作对系统带来的影响进行建模分析, 调整权重因子 weight, 从而更准确的完善系统, 进一步提高系统性能.

参考文献:

[1] LI W. NoSQL data base technology on big data environments[J]. Information & Communications, 2014, 28(7): 72-73.
[2] LIU H, CHEN Y. Research on technology and application of NoSQL in the environment of cloud computing[J]. Software Guide, 2013, 12(10): 138-140.
[3] WANG X Y, FAN X, CHEN J C, et al. Automatic data distribution in large-scale OLTP applications[J]. International Journal of Database Theory and Application, 2014, 7(4): 37-46. DOI: 10.14257/ijdt.2014.7.4.04.
[4] CHODOROW K, DIROLF M. MongoDB: the definitive guide[M]. CHENG X F, trans. Beijing: The People's Posts and Telecommunications Press, 2011: 105-171.
[5] XIN X Y. Design and research study on data model of document database[D]. Guangzhou: Sun Yat-Sen University, 2015.

- 12 — 20). https://www.xilinx.com/support/documentation/sw_manuals/xilinx2017_4/ug940-vivado-tutorial-embedded-design.pdf.
- [3] DIERICKX B, MEYNANTS G. Missing pixel correction algorithm for image sensors[C]//Proceedings of SPIE 3410, Advanced Focal Plane Arrays and Electronic Cameras II. Zurich, Switzerland: SPIE, 1998: 200-203. DOI: 10.1117/12.324015.
- [4] QIAN F, JIA J J, ZHANG L, et al. Defective pixel correction of spot-detecting camera in satellite-to-ground laser communication ATP system[J]. Chinese Journal of Lasers, 2014, 41(5): 133-139. DOI: 10.3788/CJL201441.0505007.
- [5] CAO J, YUAN Y, SU L J, et al. Adaptive bad pixel correction method for interference-modulated images based on weighted least squares support vector machines (WLS-SVM)[J]. Applied Spectroscopy, 2019, 73(4): 454-463. DOI: 10.1177/0003702819830776.
- [6] LI J S J, RANDHAWA S. Adaptive order-statistics multi-shell filtering for bad pixel correction within CFA demosaicking [C]//Proceedings of 2009 IEEE Region 10 Conference. Singapore: IEEE, 2009: 1-6. DOI: 10.1109/TENCON.2009.5395817.
- [7] HU Y S, HU Y L, YAN M, et al. Detection and correction of dead pixels in an image[J]. Journal of Shanghai University (Natural Science), 2018, 24(5): 755-762. DOI: 10.12066/j.issn.1007-2861.1853.
- [8] ZHU S C, ZHOU D L. Review on image corner detection[J]. Computer Systems & Applications, 2020, 29(1): 22-28. DOI: 10.15888/j.cnki.csa.007237.
- [9] TRAJKOVIC M, HEDLEY M. Fast corner detection [J]. Image and Vision Computing, 1988, 16(2): 75-87. DOI: 10.1016/S0262-8856(97)00056-5.
- [10] HUANG J J, ZHOU G Q, ZHOU X, et al. A new FPGA architecture of FAST and BRIEF algorithm for on-board corner detection and matching[J]. Sensors, 2018, 18(4): 1014. DOI: 10.3390/s18041014.

作者简介:

田宗浩 男, (1991-), 博士. 研究方向为图像处理/智能弹药.

刘 桢(通讯作者) 男, (1981-), 硕士, 副教授. 研究方向为图像处理/嵌入式机器视觉. E-mail: 1109180769@qq.com.

陈 栋 男, (1983-), 博士, 副教授. 研究方向为智能弹药武器系统.

(上接第 55 页)

作者简介:

冯国军 男, (1994-), 硕士研究生. 研究方向为嵌入式系统结构. E-mail: 1214155913@qq.com.

贺占庄 男, (1962-), 研究员, 硕士生导师. 研究方向为计算机系统结构, 嵌入式系统结构.

吕 瑛 男, (1981-), 高级工程师. 研究方向为嵌入式系统结构、物联网无线射频.