文章编号: 1000-5641(2017)05-0020-10

异构 Redis 集群大规模评论数据存储负载均衡设计

张敬伟[1], 丁志均[1], 杨 青[2], 张会兵[1], 张海涛[1], 周 娅[1]

(1. 桂林电子科技大学 广西可信软件重点实验室, 广西 桂林 541004;
2. 桂林电子科技大学 广西自动检测技术与仪器重点实验室, 广西 桂林 541004)

摘要: 大规模评论数据的存储与查询性能对构建于其上的各类应用的快速响应具有重要影响. 同时, 异构计算环境中各计算节点性能呈现差异, 如何充分开采各节点的计算和存储性能, 优化大规模评论数据的存储与查询性能, 是一个关键挑战. 基于 Redis 集群的数据管理优势, 首先提出了一种同构环境下基于卡槽存储平衡的大规模评论数据存储模型; 然后论证了卡槽数目与节点查询效率的关系, 以"负载与访问性能相平衡"的原则分配卡槽, 进一步设计了异构环境下的集群节点负载计算和存储分配方法, 充分开采了异构 Redis 集群中不同节点的性能. 实验结果表明, 提出的存储模型具有很好的存储平衡效果, 提升了集群的整体查询效率.

关键词: 大规模评论数据; 存储负载均衡; 查询优化

中图分类号: TP315 **文献标志码**: A **DOI:** 10.3969/j.issn.1000-5641.2017.05.003

Storage and load balancing for large-scale comment data on heterogeneous Redis cluster

ZHANG Jing-wei[1], DING Zhi-jun[1], YANG Qing[2], ZHANG Hui-bing[1], ZHANG Hai-tao[1], ZHOU Ya[1]

(1.*Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin Guangxi* 541004, *China*;
2.*Guangxi Key Laboratory of Automatic Detection Technology and Instrument, Guilin University of Electronic Technology, Guilin Guangxi* 541004, *China*)

Abstract: The storage and query performance for large-scale comment data have a great influence on those applications built on the above data. In a heterogeneous computing environment, each node has different performance on storage and computation, it presents a key challenge for optimizing the storage and query performance for large-scale comment data by taking full advantage of the performance of each node. Based on the ability of Redis cluster, we design a storage model for large-scale comment data in a homogeneous Redis cluster, which provides the storage balancing in Redis slots. And then, we discuss

收稿日期: 2017-06-30

基金项目: 国家自然科学基金(61363005, 61462017, U1501252); 广西自然科学基金(2014GXNSFAA118353, 2014GXNSFAA118390); 广西自动检测技术与仪器重点实验室基金(YQ15110); 广西高校中青年教师基础能力提升项目(ky2016YB156)

第一作者: 张敬伟, 男, 博士, 副教授, 研究方向为海量数据管理. E-mail: gtzjw@hotmail.com.

通信作者: 杨 青, 女, 副教授, 研究方向为智能信息处理. E-mail: gtyqing@hotmail.com.

the relationship between the number of Redis slots and query efficiency to design a method for allocating storage on the real load of each computing node for heterogeneous Redis clusters, which can make full use of the performance of each node and can guide to allocate slots to nodes by balancing the query performance and storage loading. Our experimental results show that the proposed model has a good effect on storage loading and improve the query efficiency of the heterogeneous Redis cluster.

Key words: large-scale comment data;　storage and load balancing;　query optimization

0　引　　言

用户在电子商务网站、Web 论坛等贡献了海量的评论数据, 其蕴含了用户的兴趣偏好、主观观点、潮流取向等信息, 对在线推荐、舆情监控、用户画像、公司商品服务决策等分析应用具有重要价值. 上述分析应用通常要求以用户、评论对象、时间段等维度为查询约束开展高 IO 的数据获取操作, 以有效支撑上层的计算与分析, 这对大规模评论数据的管理带来了挑战. 同时, 快速发展的计算机硬件, 例如多核CPU和大容量内存, 使得应对大数据的分布式计算环境也不断更新变化, 异构特征显著, 硬件的革新给大规模数据的管理带来了新理念[1]. 如何利用大内存带来的存储性价比优势进行有效的负载均衡设计, 充分开采异构集群中各计算节点的性能, 是为基于大规模用户评论数据的分析应用及其快速响应提供数据访问优化的关键, 具有重要的应用价值.

本文首先分析了基于评论数据分析应用的查询需求与特征, 基于 Redis 的内存数据管理优势, 设计了一种同构 Redis 集群下的评论数据平衡存储模型, 保证了各卡槽数据的均匀分配; 进而根据卡槽与键值的对应关系, 测试配有不同卡槽数目的节点的查询效率, 并基于测试结果、运用最小二乘法拟合出各节点卡槽数目与查询性能的关系, 最后以“负载与访问性能相平衡”的原则分配卡槽, 从而充分开采了异构 Redis 集群中不同性能节点的工作能力, 提高了集群的整体查询效率. 测试并不考虑其他因素, 只根据当前的查询性能进行判断, 避免了不可估计的参数影响, 让建议的方案具有较强的实时性.

本文内容组织如下, 第 1 节阐述了相关工作; 第 2 节对具体问题需求进行了分析; 第 3 节和第 4 节详细介绍了建议的评论数据存储模型, 以及针对异构 Redis 集群的负载计算和存储分配的具体设计方案; 第 5 节对提出的方法进行了实验验证; 第 6 节对本文工作进行了总结.

1　相关工作

根据设计与实现方式的策略异同, 现有的分布式存储系统架构主要分为两个派别: 主从结构分布式存储系统和对等网络分布式存储系统. 这两类系统的主要区别在于有无中心控制节点.

HBase、BigTable[2]、HDFS[3]是前者的典型代表, 其通过管理节点 master 管理整个集群, 监控各个存储节点 slave 并维护 slave 节点的信息, 系统的写入与读取均需要与 master 进行交互. 因此, 主从结构分布式存储结构将系统角色划分为 master 和 slave, 让 master 节点负责 slave 节点系统负载、存储平衡、故障恢复等问题. 这种管理模式简单, 便于系统维护, 但会造成 master 节点的压力过大, 成为整个系统的瓶颈, 以及 master 节点的系统故障导致系统可用性问题[4].

对等网络分布式存储系统主要通过分布式哈希算法建立逻辑映射的方式, 将数据与存储节点关联. 这种方式不需要设置中心管理节点, 每个节点拥有并维护整个集群的信息, 通过广播方式向其他节点报告自身信息. 这种分布式存储架构具有很好的拓展性, 但由于没有中心节点的控制, 使得整个系统不能很好地根据当前状态进行负载的动态均衡. Redis 即采用无中心节点的对等网络存储模式, 建立有效的动态存储负载均衡策略对 Redis 集群整体性能提升具有重要意义.

具体到对等网络的存储负载均衡问题, 当前的研究主要包含三个方面: 1) 构建动态平衡存储结构, 维持整个集群的的平衡性. 文献 [5] 将各集群中主机节点构建成一棵虚拟的平衡二叉树, 同时给出了虚拟二叉树的路由方法以及节点的加入及删除方法, 提高了对等网络的空间均衡和可用性. 文献 [6] 通过累积延迟周期调整逻辑分区的大小, 并基于一种有向无环图的方法实现逻辑分区的实时性分配. 这种方式需要消耗大量时间来保证系统平衡, 不适用于频繁更新的存储环境. 2) 设置数据的多个副本实现负载平衡. 文献 [7-8] 通过副本备份, 让集群中任意两点的存储内容都有存储交集, 用户发出请求时, 可以同时向多个节点获取内容. 这种方式可增加系统的局部处理能力及可靠性, 但随着副本数量的增加, 维护多副本之间的一致性的开销变大, 增加了系统的负担. 3) 采用数据迁移技术实现系统负载均衡. 文献 [9-10] 通过权衡数据迁移获得的收益与消耗的代价, 提出分布式数据迁移算法, 并借助于存储位置优化节点间负载平衡, 提高了系统运行效率. 本文的工作主要聚焦于构建存储平衡模型和负载迁移技术, 来解决异构 Redis 集群环境下的动态存储优化问题.

2 问题分析

在商品观点分析、用户画像等基于评论数据的分析应用中, 通常需要以商品 ID、时间范围或用户 ID、时间范围为条件展开查询操作以获取数据. 在采用 key-value 模式进行数据组织时, 上述属性需要封装在 key 中, 以建立与评论数据的一对一映射关系. 多属性封装 key 影响数据存储 key 的数量以及 key 包含数据的规模. 当较多数目的属性或分割粒度较大的属性被封装成 key 时, 键值的数目相对较少, 容易实现整个集群的存储平衡, 但在进行部分属性查询时, 需要以扫描的形式填充未给出属性, 降低了查询效率; 当较少数目的属性或分割粒度较小的属性被封装成 key 时, 会使键值数目偏大, 易造成存储和访问负载的不平衡.

在异构集群环境下, 高性能的节点会比低性能的节点具有更快的响应速度, 但并发访问效率通常受低性能节点制约. 如图 1(a)所示, 在负载不均匀分配情况下, 给予四个节点分配相同的负载量, 虽然 2∼4 节点的查询时间小于 20 ms, 但最终运行时间由 1 号节点决定; 而相对于负载均衡的负载分配(图 1(b)), 所有节点返回的时间相近, 从而提高了整个集群的查询性能. Redis 采用无中心控制节点的对等网络分布式存储结构, 支持手动或默认平均分配的方式分配卡槽实现数据存储, 这种分配方式并没有考虑异构集群环境下的节点性能差异, 需要面向节点性能差异建立动态的负载均衡方法.

针对上述问题, 本文首先设计满足大规模评论数据查询需求的平衡存储模型, 保证 Redis 中每个卡槽内的数据量相对均匀; 进而以同构环境下存储平衡的存储模型为基础, 针对 Redis 集群在异构环境下存储分配的不足, 提出了动态负载计算和访问性能预判的存储分配方法, 提高了异构 Redis 集群的并发访问效率.

3 基于 Redis 的大规模评论数据平衡存储模型设计

评论数据主要包含商品 ID、评论用户、评论时间、评论内容 4 个属性. 基于大规模评

论数据的分析应用主要以商品 ID、时间范围或评论用户、时间范围开展查询操作. 同时, 用户对商品的评论存在稀疏性, 即一个用户只对少量的商品进行过评论, 而商品 ID 属性对评论数据具有很好的聚合. 依据查询方式的不同, 将评论数据的商品 ID 属性与用户属性分别处理, 商品 ID 属性作为评论数据的承载单元, 用户属性作为辅助索引.

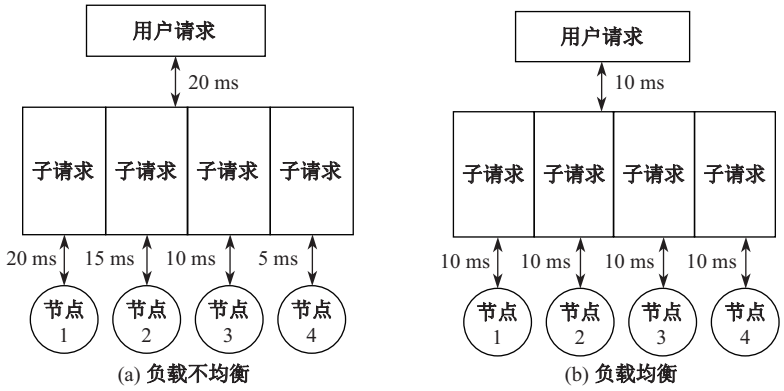

图 1　不同存储负载的访问示例

Fig. 1　Accessing illustration on different storage loading

由于不同商品拥有的评论数目不同, 导致基于商品 ID 存储的数据规模不同. 如果商品拥有的评论数目过大, 将造成集群存储不均衡. 因此, 本文提出以特定的评论数目(LineNum)为基准, 对商品评论的时间属性进行分割, 构建商品评论的二级索引. 例如当 LineNum 为 1 000 时, 即将评论数据以 1 000 个评论条目为单位进行分割, 并记录每个分割单元的起始时间, 使用 Redis 有序集合实现存储. 具体索引结构如表1所示, 其以“商品 ID(ItemID)”为键值构建第一层索引, “起始时间戳(StartTime)”为排序值构建第二层索引, “商品 ID: 分割编号(ItemID: Number)”对应评论数据集合. 表2描述评论数据的具体组织结构, 也采用有序集合进行存储, 其中具体值的内容为“用户 ID: 评论内容(UserID: comment)”, “评论时间的时间戳(Timestamp)”仍然作为排序值使用.

表 1　评论数据二级索引结构

Tab. 1　Two-level index for comment data

键名	值	
	排序值	值内容
ItemID	StartTime	ItemID: Number

表 2　评论数据存储结构

Tab. 2　Storage structure for comment data

键名	值	
	排序值	值内容
ItemID: Number	Timestamp	UserID: comment

其次, 使用“用户 ID(UserID)”属性构建辅助索引, 其中存储内容形式为“ItemID: Number: Timestamp”, 并通过“, ”加以区分. 在进行评论数据查询时, 只要涉及用户ID属性的查询, 通过检索辅助索引的方式找到对应的信息, 基于用户ID的辅助索引结构如表 3 所示.

表 3 基于用户 ID 的辅助索引结构

Tab. 3 A secondary index on UserID

键名	值
UserID	ItemID: Number: Timestamp, ItemID: Number: Timestamp, ...

上述存储模型能满足大规模评论数据的存储与查询需求, 同时以商品 ID 属性和 Redis 的卡槽数据组织模式为主要考虑因素让评论数据分布更加均衡, 确保每个卡槽的存储数据量是相近的.

4 异构Redis集群下基于负载计算和访问性能预判的存储分配方法

在异构集群环境下, 由于各节点的性能不一致, 不合理的任务分配会降低整个集群的工作效率. 鉴于 Redis 集群只提供了手动或默认均匀分配存储的方案, 没有提供根据工作当前状态优化存储分配的方法, 本文设计了面向异构 Redis 集群的负载计算和访问性能预判的存储分配方法, 根据节点的实际查询性能实现均衡的存储分配.

设计的方法包含 4 个阶段. (1) 建立卡槽节点映射表与卡槽测试键列表: 根据键值与 Redis 的卡槽映射关系, 为每个卡槽生成一个测试键值对, 将全部的测试键值对存储在 Redis 集群中, 并统计 Redis 集群信息, 包括卡槽与节点关系、每个节点拥有的卡槽数目; (2) 节点查询性能测试: 根据卡槽节点映射表与卡槽测试键列表生成各个节点的卡槽测试键值表, 并根据节点的卡槽数目将该节点的卡槽测试键值集合均匀地分成 10 等份, 以份数为单位, 对各个节点进行访问性能测试; (3) 计算当前状态下的存储负载: 运用最小二乘法拟合出各节点的卡槽数目与查询性能的关系, 求得节点查询性能与查询负载相平衡的卡槽数目分配比例; (4) 迁移卡槽数据实现存储均衡: 根据新、旧的卡槽数目运用最大卡槽转出数目与最大卡槽移入数目相匹配的方式计算需要转移的卡槽, 执行节点间卡槽转移.

4.1 建立卡槽测试键及卡槽节点映射表

该阶段的主要任务是生成卡槽测试键表以及卡槽机器映射表, 记录各机器卡槽数目等. 卡槽测试键是基于 A∼Z 或 a∼z 字符集合随机生成的 4 位字符串, 将该字符串作为卡槽测试键值. 鉴于 Redis 的最大卡槽数目是 16 384 个, 将上述测试键值的 CRC16 码与 16 384 取余为测试键值, 建立卡槽分配函数, 基于获取的卡槽索引号将字符串存储在卡槽测试键列表(SlotString)中. 以<key, index>键值对形式存储到 Redis 中, 其中 key 为生成的 4 位字符串, index 为卡槽测试键列表的索引值, 即 Redis卡槽编号, 从而使得每个卡槽都存有一个卡槽测试键, 卡槽测试键表如图 2 所示.

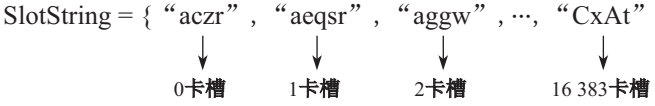

图 2 16 384 个卡槽测试键表

Fig. 2 Illustrating test key table for 16 384 slots

读取卡槽节点映射表信息, 统计每个节点分配的卡槽数目, 并存储在节点卡槽数目列表中. 卡槽节点映射表(SlotToMachine)、节点卡槽数目列表(SlotNum)具体格式如下. SlotToMachine=$\{1,2,1,\cdots,n,\cdots\}$, 集合内每个元素表示节点的索引号, 一个卡槽对应一个节点, 列表长度为 16 384, 列表的索引值表示卡槽编号. SlotNum=$\{N_1, N_2, \cdots, N_n\}$, 其中$N_n$表示编号为$n$的节点拥有的卡槽数目.

4.2　负载测试

上述预处理过程使得 Redis 集群中每个卡槽都具有唯一的一个测试键, 同时设计的评论数据存储模型保证了各卡槽映射的数据相对均匀, 这使得可以将对卡槽数目的负载测试转化为对键值数目的测试. 每次查询操作可简化成基于测试键值计算卡槽, 依据卡槽归属信息获得节点, 再从节点中提出数据的过程. 因此, 对于特定节点测试是相对独立的, 测试过程如图 3 所示, 具体描述如下.

(1) 遍历卡槽节点映射表, 根据卡槽节点映射表的节点索引号 n 以及在表元素的索引值i, 获取卡槽测试键列表中对应各卡槽的卡槽测试键集合SlotString[i], 并进而建立各节点的卡槽测试键列表;

(2) 将各节点的卡槽测试键列表均匀地分割 10 等份, 当测试键值(或卡槽数目)无法满足被 10 整除时, 将剩余的测试键值全部放入到最后一个等份中;

(3) 以一份测试键集合为一个测试单位, 对 Redis 集群进行查询性能测试, 第一次查询第一份测试键集合中的所有测试键值, 第二次查询第一份和第二份两个测试键集合中的所有测试键值, 以此类推. 每一次的测试运行 5 次, 将查询测试的时间存储在测试时间列表中 ResultList=$\{t_{1,1}, t_{1,2}, t_{1,3}, t_{1,4}, t_{1,5}, t_{2,1}, \cdots, t_{p,q}, \cdots, t_{10,5}\}$, 其中 p 表示参加测试的份数, q 表示当前的测试次数;

(4) 将各节点的测试结果根据节点索引号排序, 共产生 $n \times 50$ 个测试结果, n 为节点的数目.

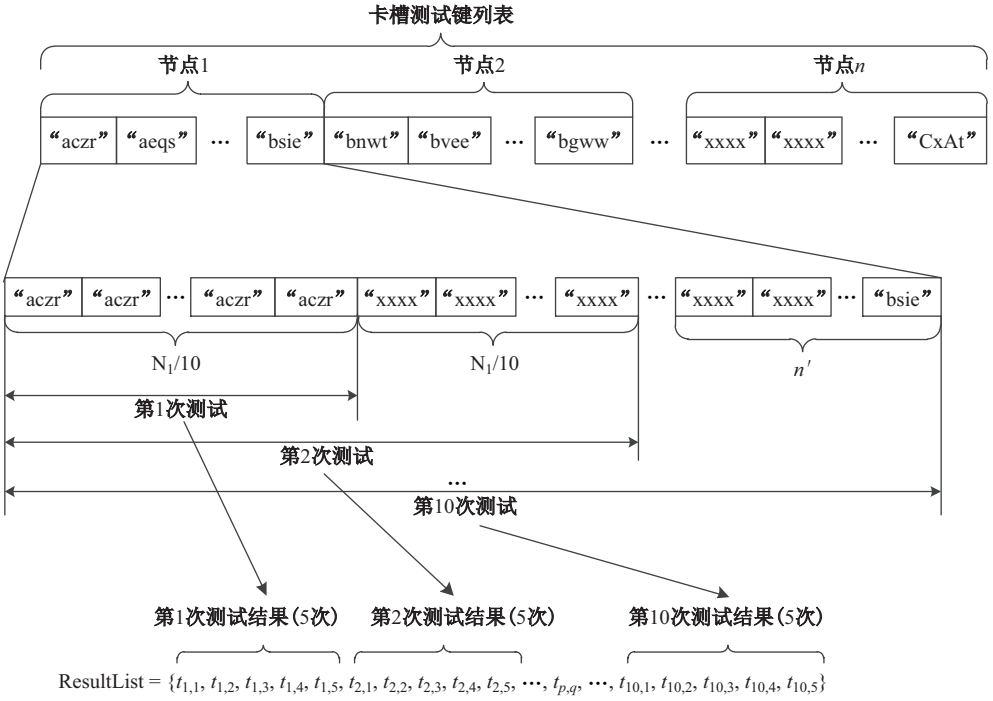

图 3　查询负载测试过程

Fig. 3　The test process for query performance

4.3　访问性能预测与存储分配

上节的查询性能测试中, 影响查询效率的因素主要有节点本身的性能, 已有负载和网络传输, 鉴于节点的查询效率与测试键数目成正比例关系. 由于键值数目可转化为卡槽数目, 得如下方程(1).

$$T = vN, \tag{1}$$

其中, T 表示卡槽集合查询所需要的时间, v 表示测试机器在当前状态下的查询性能, N 表示卡槽的数目. 使用最小二乘法求得方程中 v 的值, 即保证预测的结果与实际值的误差最小, 给出最小误差平方和公式(2).

$$w = \sum_{p=1}^{10} (t_p - n_p^{\mathrm{T}} v)^2, \tag{2}$$

其中, t 表示键值集合的查询时间, v 表示节点的查询性能, n 为当前测试键集合的键值数目(等于卡槽数目), p 为当前参与测试键值集合的份数. 该公式转换为矩阵表示, 即$(T-Nv)^{\mathrm{T}}(T-Nv)$, 对 v 求导得到$-2N^{\mathrm{T}}(T-Nv)$, 令其等于零, 则有各个节点的卡槽数目与查询时间的关系如(3)所示.

$$v = (N^{\mathrm{T}}N)^{-1}N^{\mathrm{T}} \cdot T. \tag{3}$$

根据各节点卡槽数目与查询时间的线性关系, 保证查询负载与查询性能相平衡, 即不同节点完成不同查询任务的时间应相同, 有

$$T_1 = T_2 = T_3 = \cdots = T_n,$$
$$v_1N_1 = v_2N_2 = v_3N_3 = \cdots = v_nN_n.$$

因此, 得到不同节点卡槽数目的关系应为

$$N_1 : N_2 : N_3 : \cdots : N_n \Leftrightarrow \frac{1}{v_1} : \frac{1}{v_2} : \frac{1}{v_3} : \cdots : \frac{1}{v_n},$$

且$N_1 + N_2 + N_3 + \cdots + N_n = 16\,384$. 在实际计算中, N 若不能刚好被 16 384 整除, 有如下约定, 当$\sum_{\mathrm{i}=1}^{n} N_i < 16\,384$时, 将剩余的应分配卡槽分配到 v 值最小的节点上; 当$\sum_{\mathrm{i}=1}^{n} N_i > 16\,384$ 时, 将多出的冗余卡槽数目从 v 值最大的节点中移除, 从而得到节点查询负载平衡的新卡槽数目, 将新得到的卡槽数目存储到NewSlotNum中, NewSlotNum= $\{cN_1, cN_2, \cdots, cN_n\}$, 其中, cN_n表示编号为n的节点存储的卡槽数目.

4.4 迁移数据卡槽

为保证数据转移效率, 本文采用最大转出卡槽数目节点与最大移入卡槽数目节点相匹配的策略计算需要转移的卡槽, 从而保证每一次匹配都可移除一个节点的卡槽转移计算.

首先以新卡槽数目为目标, 求出各节点需要转移卡槽的数目, 节点卡槽数目增多为正值, 节点卡槽数目减少为负值, 生成节点卡槽转移数目列表 NodeSlotMigrateList= $\{MN_1, MN_2, MN_3, \cdots, MN_n\}$, 其中, MN为节点卡槽转移的数目, n 为集群节点的数目; 再从NodeSlotMigrateList中寻找节点卡槽增加的最大值 P 和节点卡槽减少的最大值Q, 将两个卡槽节点的数目进行匹配, 即将卡槽减少最多的节点转向卡槽增加最多的节点. 扫描卡槽节点列表(SlotToMachine)提取转移的卡槽, 并以<FromMachine, ToMachine, Index1, Index2,...,Indexi>形式存储转移卡槽记录, Index 表示需要转移的卡槽编号, 更新 SlotToMachine列表, 同时将 NodeSlotMigrateList 中的 P 与 Q 绝对值较小的值置为 0, 较大的值置为 $P+Q$. 重复上述操作, 直至节点转移卡槽的值全部为 0; 最后, 根据卡槽节点映射表 SlotToMachine 记录的卡槽转移记录执行卡槽转移.

5 实 验

为了测试在异构条件下的 Redis 集群存储分配优化方案, 本文在一台Dell PowerEdge R370 服务器上基于 VMware 云平台搭建了 Redis 集群, 服务器拥有 20 个 CPU, 128 G 内存. 实现的异

构集群由3台虚拟机节点构成, 节点的处理核心数分别为 1、2、4, 内存分别为 4 G、4 G、8 G, 操作系统为 red hat linux 5.6. 节点与测试机在同一个局域网内, 网卡为千兆以太网卡, Redis 集群版本为 3.2.6. 实验数据为真实的京东评论数据, 大小为 900 M, 472 件商品, 共 5 241 992 条评论记录, 评论数目最多的商品拥有 45 万多条评论, 用户 2 642 152 人.

实验分为三个部分, 第一部分为评论数据存储模型的分割跨度的设定, 选出存储平衡性最好的存储跨度(即以特定数目 LineNum 对评论进行分割); 第二部分测试在默认平均分配卡槽的 Redis 集群下, 通过本文提出的算法进行存储负载的测试, 验证负载与卡槽数目线性关系的正确性, 并计算出符合负载的卡槽分配数量; 第三部分为卡槽移动前后的 Redis 集群查询性能对比.

5.1　模型分割参数实验

以默认平均分配卡槽的方式构建 Redis 集群, 3 个节点拥有卡槽数目分别为5 462、5 461、5 461, 共 16 384 个卡槽. 将上述评论数据集以 10 000 为跳步, 依次按照本文提出的存储模型开展存储效果测试, 测试结果如表4所示. 表 4 的测试结果显示, 2 万条数目进行分割的存储平衡效果最好, 随着分割参数的增大, 平衡的效果逐渐弱化, 最后平衡的效果趋近于不分割的存储平衡效果. 由于 Redis 系统本身以及管理数据的结构所占用的空间, 其总量约为数据本身大小的 1.5 倍左右.

表 4　存储平衡分割参数(LineNum)的测试结果

Tab. 4　Experimental results for parameter(LineNum) of storage partition

分割参数(LineNum)/万	节点1/M	节点2/M	节点3/M	标准差
1	502.40	501.26	460.20	19.63
2	495.12	489.26	471.28	10.14
3	509.81	486.51	421.12	37.54
4	486.44	526.01	433.86	37.74
5	557.57	414.99	450.43	60.61
6	644.51	362.38	410.35	123.26
7	664.95	376.70	384.30	134.13
8	661.56	400.65	373.29	129.92
9	648.79	469.65	339.60	126.76
10	602.16	471.74	341.24	106.52

5.2　存储负载测试

首先, 基于默认值搭建 Redis 集群, 则各节点分别被分配 5 462、5 461、5 461 个卡槽. 使用存储负载管理器对Redis集群进行测试, 测试结果显示卡槽数目与读取时间的回归曲线如图 4 所示. 该图验证了卡槽读取的数目与读取时间的线性关系, 在系统不稳定时, 测试会稍现波动, 例如, 图 4 中配置有 2 核心 CPU、4G 内存的节点的性能有分散形状.

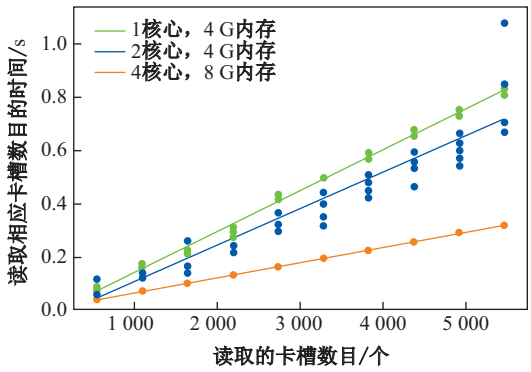

图 4　节点查询性能测试

Fig. 4　Performance test for node query

图 4 显示 1 核、2 核、4 核节点的查询速度分别为 $v_1 = 0.151\ 55$ 毫秒/槽、$v_2 = 0.130\ 67$ 毫秒/槽、$v_3 = 0.058\ 305\ 2$ 毫秒/槽. 根据节点拥有卡槽数量与速度成反比的关系, 得到新的卡槽数量应分别为 3 449、4 000、8 935. 卡槽转移前后 Redis 集群的数据存储如图 5 所示, 卡槽移动后的存储比例如表 5 所示.

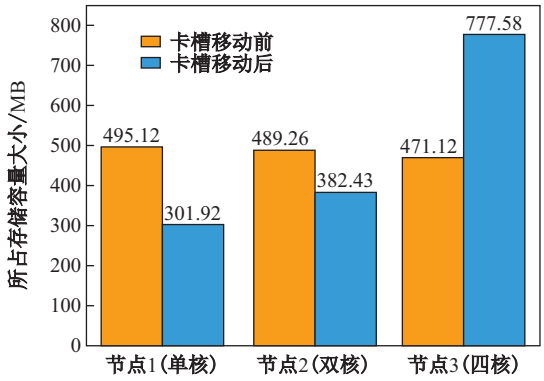

图 5 迁移卡槽前后存储数据量对比

Fig. 5 Comparison of data volume before and after shifting slots

表 5 卡槽转移后存储数据比例

Tab. 5 Ratios after shifting slots

项目	节点(1核、2核、4核)	比例值
键值	235、299、620	1: 1.272: 2.638
卡槽转移后存储容量	301.92 MB、382.43 MB、777.58 MB	1: 1.266: 2.639
卡槽数量	3 449、4 000、8 935	1: 1.159: 2.590

5.3 查询性能对比

从数据集中随机选出 10 件商品, 具体数据规模如表 6 所示. 测试查询 10 件商品分别在 1 日、1 月、半年、1 年、1 年半时间范围内的所有评论所需要的时间, 测试结果如表7所示, 结果显示查询效率有不同程度的提高, 但并无明显的变化规律. 这是由于评论数据本身随时间的密集程度不一以及单条评论内容大小是不确定的. 从表 6 中可以发现商品 4、5 虽然数目比商品2多, 数据的大小却小于商品 2. 数据的密集程度将影响查询结果的数目, 单条评论内容大小同时影响查询结果, 二者的不确定导致性能提高的不确定.

表 6 查询数据表

Tab. 6 Data fact for testing queries

商品ID	评论数目/条	容量/M
1	117 908	27.1
2	58 182	15.1
3	212 708	40.5
4	85 236	11.4
5	104 352	9.07
6	93 431	8.73
7	53 937	4.65
8	26 439	2.60
9	1 691	1.12
10	985	0.508

表 7　范围查询测试结果

Tab. 7　The experimental results for queries

查询范围项目	卡槽移动前/s	卡槽移动后/s	速度提高率/%
1日	0.022 64	0.018 35	23.4
1月	0.278 66	0.223 21	24.8
半年	1.112 17	1.032 61	7.71
1年	1.944 21	1.833 71	6.00
1年半	2.404 70	2.168 68	10.9

6　总　　结

本文主要针对异构 Redis 集群环境下评论数据存储的负载均衡问题, 详细分析了大规模评论数据的查询需求, 提出了一种同构环境下以商品 ID 为键值存储评论数据, 以用户 ID 为键值建立辅助索引的存储模型, 并以一定的分割阀值对存储数据进行分割, 从而保证存储系统的平衡性. 然后, 根据建立的存储平衡, 提出了异构环境下的集群节点负载计算和存储分配方法, 通过对测试键的查询得到卡槽与节点查询效率的关系, 以"负载与访问性能相平衡"的原则来分配卡槽, 并进行了理论分析. 建议的方法充分开采了异构 Redis 集群中不同性能节点的工作能力, 提高了集群的整体查询效率.

[参　考　文　献]

[1] INTEL. A yearly product cadence moves the industry forward in a predictable fashion that can be planned in advance[EB/OL]. [2017-05-10]. https://www.intel.com/content/www/us/en/silicon-innovations/intel-tock-model-general.html.

[2] CHANG F, DEAN J, GHEMAWAT S. et al. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems, 2006, 26(2): 205-218.

[3] BORTHAKUR D. The Hadoop distributed file system: Achitecture and design [EB/OL]. [2017-06-02]. http://hadoop.apache.org/common/docs/r0.180/hdfs_design.pdf.

[4] 申德荣, 于戈, 王习特, 等. 支持大数据管理的 NoSQL 系统研究综述[J]. 软件学报, 2013(8): 1786-1803.

[5] 何亚农, 宋玮, 赵跃龙. 基于平衡结构的对等网络存储系统研究[J]. 计算机工程与设计, 2011, 32(8): 2611-2613.

[6] KALA K A, CHITHARANJAN K. Locality Sensitive Hashing based incremental clustering for creating affinity groups in Hadoop-HDFS-An infrastructure extension[C]// International Conference on Circuits, Power and Computing Technologies. IEEE, 2013: 1243-1249.

[7] ROWSTRON A, DRUSCHEL P. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility [C]// Proceedings of the 18th ACM Symposium on Operating Systems Principles. ACM, 2001: 188-201.

[8] OKCAN A, RIEDEWALD M. Processing theta-joins using MapReduce[C]// Proceedings of SIGMOD International Conference on Management of Data. ACM, 2011: 949-960.

[9] WEI Q, VEERAVALLI B, GONG B, et al. CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster[C]// IEEE International Conference on CLUSTER Computing. 2010: 188-196.

[10] XIE C, CAI B. A decentralized storage cluster with high reliability and flexibility[C]// Proceedings of 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. IEEE, 2006: 1-8.

(责任编辑: 林　磊)