基于 Redis 单位最大效益自适应迁移策略研究

韦 立[1,2],陈珊珊[1,2]
(1. 南京邮电大学 计算机学院,江苏 南京 210003;
2. 江苏省大数据安全与智能处理重点实验室,江苏 南京 210003)

摘 要:Key-Value 存储以其高扩展性和强一致性在如今热门的大数据时代中扮演着重要角色。而实现其扩展性重要的关键技术在于数据迁移机制的完善,数据迁移机制同时直接影响集群节点的负载均衡。Redis 集群采用哈希算法将数据均匀分布在集群节点上,然而随着节点的不断扩展,Redis 集群手动分配槽的迁移机制日益不足。针对上述问题,提出单位最大效益自适应迁移策略。一方面,负载自适应平衡策略通过实时判断集群负载情况解决 Redis 手动迁移的不足,有策略的数据迁移使其静态存储均衡化,从而实现系统之间的自适应调控;另一方面,单位最大效益迁移方案每次优先选择单位效益最大的节点进行迁移,直至迁移完成。和 Redis 性能相比,自适应迁移策略的迁移时间大概缩短了 6.2% ~14.1%,吞吐量约提高了 4.9% ~12.5%。

关键词:Redis;Key-Value;负载平衡;自适应迁移策略;单位最大迁移效益
中图分类号:TP39 **文献标识码**:A **文章编号**:1673-629X(2018)10-0053-06
doi:10.3969/j.issn.1673-629X.2018.10.011

Research on an Adaptive Migration Strategy of Unit Maximum Benefit Based on Redis

WEI Li[1,2],CHEN Shan-shan[1,2]
(1. School of Computer Science,Nanjing University of Posts and Telecommunications,Nanjing 210003,China;
2. Jiangsu Key Laboratory of Big Data Security & Intelligent Processing,Nanjing 210003,China)

Abstract:Nowadays,Key-Value storage has played an important role in the era of big data for its high extension and strong consistency. The key technology to achieve extensibility is to improve the data migration mechanism which also has affected load balancing of cluster node. Redis cluster utilizes hash algorithm to uniformly distribute data over cluster nodes. However,with the increasing of nodes,the mechanism of manually allocating slots in Redis cluster becomes inadequate. In order to solve the above problems,we present an adaptive migration strategy of unit maximum benefit. On the one hand,the strategy solves the shortage of manual migration of Redis by judging load of cluster system real-time and achieves adaptive control between systems by strategic migration,which leads to static storage equilibrium. On the other hand,we propose a migration scheme of unit maximum benefit to optimize migration plan. The scheme will choose the optimal node and find the best strategy until all migration are finished. Compared to Redis,the migration time of adaptive strategy has shortened by 6.2% ~14.1% and the throughput has increased by 4.9% ~12.5%.

Key words:Redis;Key-Value;load balance;adaptive migration strategy;unit maximum benefit

1 概 述

为了应对海量数据带来的挑战和解决传统数据库访问大规模数据的瓶颈问题,NoSQL[1]型数据库应运而生。以 MongoDB[2]、Redis[3]、Cassandra[4]为代表的 NoSQL 产品以其高性能、强扩展性和高容错性受到了人们的青睐,并在数据库领域掀起了一场新的革命。NoSQL 数据库常常以 Key-Value 形式存储,Key-Value 存储技术为应用提供低延时、高可用、可伸缩的数据访问服务。对于 Key-Value 存储系统,数据迁移[5]是实现节点动态扩展与弹性负载均衡[6]的关键技术,而 Key-Value 常用的数据分布策略是一致哈希。一致性哈希可以达到很好的分布式均匀性,并且由于特殊

收稿日期:2017-12-04 **修回日期**:2018-04-10 **网络出版时间**:2018-05-28
基金项目:国家自然科学基金(61572263)
作者简介:韦 立(1992-),男,硕士研究生,研究方向为基于内存分布式键值存储;陈珊珊,博士,副教授,研究方向为基于内存分布式键值存储。
网络出版地址:http://kns.cnki.net/kcms/detail/61.1450.TP.20180525.1610.076.html

的放置规则,其在节点数据发生变动时可以将影响控制在局部区间内,从而保证非常少的数据迁移。当然,必要的迁移代价必不可少,因此如何有效降低迁移代价、提高迁移效益[7]是 Key-Value 存储需着力解决的问题。

Redis 从 3.0 版本开始支持集群功能,集群采用无中心节点方式,实现客户端直接与 Redis 集群的每个节点连接,节点之间通过 Gossip 协议交换互相的状态和探测节点扩展、删除信息(见图 1)。Redis 集群中有一个长度为 16 384 的哈希槽,这个槽是虚拟的,并不是真正存在的。正常工作时,Redis 集群中的每个节点都会负责一部分的槽,当有某个 Key 被映射到某个主节点负责的槽,那么这个主节点负责为这个 Key 提供服务,至于哪个主节点负责哪个槽,可以由用户指定,也可以在初始化时自动生成。所以,当集群添加节点时,用户可以从其他节点随机各抽取一部分槽到新节点,也可以指定某些节点上的槽迁移到新节点。但是目前 Redis 集群数据迁移操作必须手动进行,而且随机选择的迁移方案也并不是最优的。为了实现数据有策略的迁移,文中提出了单位最大效益自适应迁移策略。

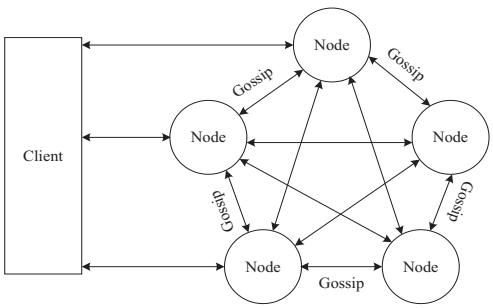

图 1 Redis 集群无中心节点模式

秦秀磊等[8]提出基于面积的迁移代价模型和开销敏感迁移算法来完善 Key-Value 迁移机制。他们用二维空间面积衡量迁移节点之间的迁移代价,任意的迁移代价可用对应的矩形面积来表示,因此将迁移代价选择转为矩形面积的选择。开销敏感迁移算法基于迁移代价模型,算法每次选择代价最小的节点进行迁移,这样选取的迁移策略确实减少了迁移代价,然而开销敏感迁移算法并没有考虑数据量的问题,因此选取的迁移方案并不是最优的。通过分析开销敏感迁移算法存在的不足,文中基于面积迁移代价模型,提出的单位最大效益自适应迁移策略包含两部分:负载自适应平衡策略和单位最大效益迁移方案。负载自适应平衡策略通过实时判断集群系统负载情况解决 Redis 手动迁移的不足,有策略的数据迁移使其静态存储均衡化,从而实现系统之间的自适应调控;单位最大效益迁移方案每次优先选择单位效益最大的节点进行迁移,直至迁移完成。单位最大效益自适应迁移策略主要实现三个过程:数据迁移时机的选择,即在什么情况下触发迁移;迁出节点和迁入节点的选择,即迁出节点和迁入节点的判定;迁移效益的选取,即选取效益最大的迁移方案。

2 单位最大效益自适应迁移策略

2.1 系统负载

对于负载均衡的考虑,模型引入信息熵[9]的概念来衡量集群的负载。假设节点 i 的负载水平为 l_i,标准化节点负载 P_i 见式 1,平均负载见式 2:

$$P_i = \frac{l_i}{\sum_{i=1}^{n} l_i} \tag{1}$$

$$l_{avg} = \frac{\sum_{i=1}^{n} l_i}{n} \tag{2}$$

集群所需要的是节点负载的均匀程度,信息熵完全可以作为系统有序化和均匀程度的一个度量。因此,自适应迁移策略采用信息熵表示集群的负载分布情况,如式 3:

$$H(P) = -\sum_{i=1}^{n} P_i \log P_i \tag{3}$$

熵值越高,表明负载分布越均匀。当各节点负载相等时,系统取得最大熵值 $H(P)_{max} = \log n$。

为了更加直观地表示负载分布情况,自适应迁移策略还构建了一个均衡度函数 $F \in (0,1)$,即标准化熵值,见式 4:

$$F = -\sum_{i=1}^{n} P_i \log P_i / H(P)_{max} = \sum_{i=1}^{n} P_i \log P_i / \log \frac{1}{n} \tag{4}$$

2.2 自适应平衡策略

为了实现 Redis 数据有策略的迁移,使其静态存储均衡化和负载均衡,构建了一个负载均衡的自适应的数据迁移框架,如图 2 所示。

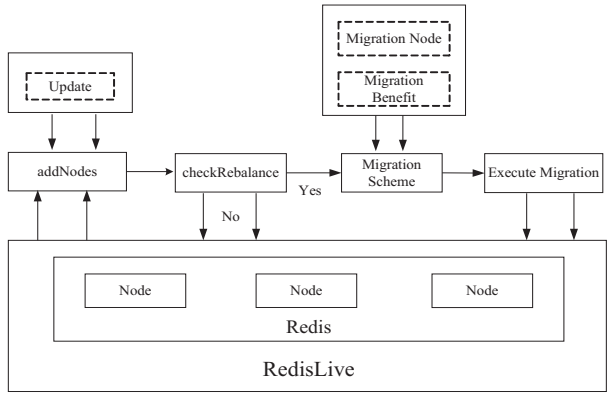

图 2 自适应负载平衡框架

数据信息通过哈希算法[10]将数据分布在各个虚

拟节点实现数据的均匀分布;当数据更新时,RedisLive 对集群进行实时监测并收集信息。如果整个系统的负载低于均衡度阈值就会触发数据迁移操作,反之,仍然保持当前状态;迁移方案的选择尤为关键,迁出迁入节点的选择和迁移代价的比较等因素都是衡量的重点,统筹这些因素,制定出迁移最大效益的方案;最后,执行数据迁移。

Redis 集群通过哈希算法将数据分布在各个虚拟节点实现数据的均匀分布,因此数据可以维持数据平衡。然而,当 Redis 集群为实现横向扩展而增加新节点时,平衡策略自适应地将进行有策略的数据迁移。算法 1 主要包含三个步骤:判断、挑选、迁移。

```
Algorithm1:The checkRebalance procedure
procedure checkRebalance(add or remove nodes)
calculate F
if F <threshold
engage(nodes)
do migration S
return
Else
return
End
```

判断:通过信息熵计算方法,集群计算出当前系统的均衡度函数,并预先设定一个阈值。开始阶段整个集群系统比较稳定,但是随着新节点的添加,当前负载平衡系统被打破,触发节点进行数据迁移,这也是对应了迁移时机的考虑。当 F 小于阈值,CheckRebalance 算法触发数据迁移操作,自适应调节节点存储情况。

选取:一般来说,数据迁移操作不会选择整体节点进行操作,这样做目标繁多,而且带来的开销也很巨大。因此,文中有目的性地选择一些需要参加的节点进行数据迁移操作,以大大减少迁移代价。根据信息熵,每个节点 i 都会有各自的节点负载 l_i 和平均负载 l_{avg}。于是,所有参加操作的节点分成两类,分别是迁入节点和迁出节点。当节点 i 的负载水平 $l_i < l_{avg}$ 时,节点 i 归入 MI 迁入节点集合;当节点 j 的负载水平 $l_j > l_{avg}$ 时,节点 j 归于 MO 迁出节点。MI 集合中的每个迁入节点会附带一个排序信息,记录迁出节点到该节点的单位迁移效益排序,这样有利于最佳迁移方案迁移节点的选取。

迁移:上面已经确定迁移时机,并选取部分参加的节点进行迁移操作,接着就是选择最大效益迁移方案。Redis 集群数据迁移随机选择节点进行数据迁入或者需要手动迁移槽;开销敏感数据迁移方案(CA)每次选取效益最大的节点迁移;单位效益最大迁移方案(Packet)选择单位迁移数据量效益最大的节点迁移。通过三种方案的比较,单位最大效益迁移方案的迁移效益最高。

2.3　单位最大效益迁移方案

在数据迁移过程中,为了实现操作的规范性,每个节点的一系列操作可以表示成一个四元组 $<j,i,T,B>$,其中 j 和 i 分别表示迁出节点和迁入节点,T 表示迁移分区,B 表示迁移带宽。秦秀磊等[8]提出的开销敏感数据迁移算法计算出使迁移开销代价最小的带宽值 B 以及该节点的最小开销代价,然后选取开销代价最小的节点作为迁入节点。该算法虽然每次选择代价最小的节点进行迁移,但是没有考虑到数据量的问题,数据量的大小制约节点的重新选择,局部开销代价最小并不能保证整体开销代价最小。因此,很容易将数据迁移节点选择问题规约为一般背包[11]问题。

在开销敏感数据迁移算法的基础上,提出单位效益最大迁移算法。该算法运用贪心准则,每一次迁入节点优先选择单位效益最大的迁出节点进行数据迁移。在固定迁移数据量的前提下,根据基于面积的迁移代价模型,首先计算节点 j 到节点 i 的迁移代价 $\mathrm{cost}_{(j,i)}$,迁移代价越小则说明迁移效益越高,所以定义 $p_{(j,i)}$ 表示节点 j 到节点 i 的收益;其次,根据单位迁移效益 $p_{(j,i)}/w_j$ 递减排序,依次选择迁出节点,每个迁入节点生成最优方案;最后,将 MI 集合中每个迁入节点方案汇总成最终数据迁移方案。

$$p_{(j,i)} = k/\ \mathrm{cost}_{(j,i)} \tag{5}$$

图 3 为 9 号迁入节点最优迁移方案生成过程。

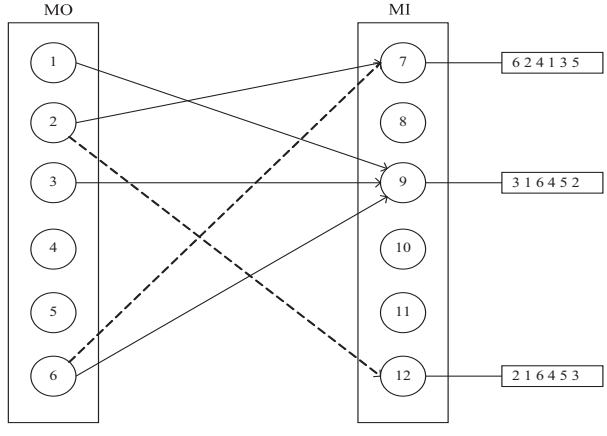

图 3　9 号迁入节点最佳迁入方案

(1)1 ~ 6 号节点分配到 MO,7 ~ 12 号节点归入 MI 集合。

(2)当 9 号节点需要数据迁入时,首先分别计算 1 ~ 6号节点到 9 号节点的单位迁移效益,然后进行排序(3 1 6 4 5 2)。

(3)优先迁移 3 号节点数据到 9 号节点;当 3 号节点数据迁移完成后,迁移 1 号节点数据到 9 号节点;当 1 号节点迁移完成后,迁移 6 号节点一半数据到 9 号节点,至此 9 号节点饱和,6 号节点仍有剩余一半数据

迁移到其他节点。

(4)9 号迁入节点选择的迁出节点为(3 1 6),分别迁入数据量为(1,1,1/2)。

不断重复上述算法最终得到最优化的迁移方案,当负载均衡的信息熵高于特定阈值,数据迁移结束。

最优迁移方案如算法 2 所示。

Algorithm2:The optimal scheme for migration
Input:MI_set,MO_set;
Output:Migration S
Initialize a migration S ,a temporary partition sets T
Let l_{avg} become the average load
repeat
$m_i = l_{avg} - l_i, i \in$ MI_set
Add m_i to T
for each node in T do
if $T \neq \varnothing$ then
for each node in MI_set do
$w_j = l_j - l_{avg}, j \in$ MO_set
let $cost_{(j,i)}$ be the cost from j to i
$p_{(j,i)} = k / cost_{(j,i)}$
add $p_{(j,i)}/w_j$ to s_i
sort s_i
update the state of j and i
else
return
add s_i to S
end
until F >threshold

2.4 贪心法求解

问题描述:数据迁移问题的最优方案 $S = \{X_1, X_2, \cdots, X_i\}$,n 为节点数量,节点 i 迁移方案可以表示成一个 n 元组:$X_i = (x_0, x_1, \cdots, x_{n-1})$,$0 \leqslant x_i \leqslant 1$;$m_i$ 表示迁入 i 节点需要迁入的数据量,即 $m_i = l_{avg} - l_i$;w_j 表示迁出节点 j 需要迁出的数据量,即 $w_j = l_j - l_{avg}$;所以约束条件为:

$$\sum_{i=0}^{n-1} w_j x_i \leqslant m_i, 0 < i \leqslant j < n \tag{6}$$

最优化问题的目标函数用于衡量一个可行解是否是最优解,文中目的是使迁移效益最大,所以最优解使目标函数取最大值:

$$\max \sum_{i=0}^{n-1} p_{(j,i)} x_i, 0 < i \leqslant j < n \tag{7}$$

定理:如果 $p_0/w_0 \geqslant p_1/w_1 \geqslant \cdots \geqslant p_{n-1}/w_{n-1}$,则式 7 求得的解是最优解[12]。

证明:设 $X_i = (x_0, x_1, \cdots, x_{n-1})$,$0 \leqslant x_i \leqslant 1$ 是贪心算法的解。若节点 i 所有迁出数据量都能迁移到目标节点,即 $x_i = 1$,则显然是最优解。不然,则设 m 是使 $x_m \neq 1$ 的最小下标。从贪心算法可知,解的形式为:

$$X_i = (1, \cdots, 1, x_m, 0, \cdots, 0), 0 \leqslant x_m < 1 \tag{8}$$

如果 X_i 不是最优解,而另有可行解 $Y_i = (y_0, y_1, \cdots, y_k, \cdots, y_{n-1})$ 是最优解,使得

$$\sum_{i=0}^{n-1} p_{(i,j)} y_i > \sum_{i=0}^{n-1} p_{(i,j)} x_i \tag{9}$$

设 k 是使得 $y_k \neq x_k$ 的最小下标,显然这样的下标必定存在。下面证明 $y_k < x_k$。可以分三种情况:

(1) 若 $k < m$,则因为 $x_k = 1, y_k \neq x_k$,所以得出 $y_k < x_k$。

(2) 若 $k = m$,则 x_k 是第 k 个节点能迁入目标节点的最多数据量,从而 $y_k > x_k$ 是不可能的。由于 $y_k \neq x_k$,必定有 $y_k < x_k$。

(3) 若 $k > m$,这是不可能的,因为 $x_i = 0, m < i < n$。若 $y_k \neq 0$,则目标节点必定超载。

综上所述,必有 x_k。

假定以 x_k 替换 $Y_i = (y_0, y_1, \cdots, y_k, \cdots, y_{n-1})$ 中的 y_k,则 $Z_i = (z_0, z_1, \cdots, z_k, z_{k+1}, \cdots, z_{n-1})$,替换前 $z_i = y_i = x_i, 0 \leqslant i \leqslant k-1$,替换后 $z_k = x_k$。为了保证 Z_i 是可行解,必须满足:

$$\sum_{i=k+1}^{n-1} w_i (y_i - z_i) = w_k (z_k - y_k) \tag{10}$$

这意味着因为第 k 个节点迁入目标节点的数据由 y_k 增加到 x_k (即 z_k),则需要减少从 Y_i 中第 $k+1$ 到 $n-1$ 节点迁入目标节点的数据量,减少的数据量必须等于增加的数据量,即应当满足式 10。经过这样处理后得到的可行解 Z_i ,其前 k 个分量均与 X_i 相同。式 11 计算说明可行解 Z_i 的效益大于等于假定的最优解 Y_i 的效益:

$$\begin{aligned}\sum_{i=0}^{n-1} p_{(i,j)} z_i &= \sum_{i=0}^{n-1} p_{(i,j)} y_i - \sum_{i=0}^{n-1} (y_i - z_i)\left(\frac{w_i}{w_i}\right) p_{(i,j)} = \\ &\sum_{i=0}^{n-1} p_{(i,j)} y_i - (y_k - z_k)\left(\frac{w_k}{w_k}\right) p_{(k,j)} - \\ &\sum_{i=k+1}^{n-1} (y_i - z_i)\left(\frac{w_i}{w_i}\right) p_{(i,j)} \geqslant \sum_{i=0}^{n-1} p_{(i,j)} y_i - \\ &(y_k - z_k)(w_k)\left(\frac{p_{(k,j)}}{w_k}\right) - \sum_{i=k+1}^{n-1} (y_i - \\ &z_i)(w_i)\left(\frac{p_{(i,j)}}{w_i}\right) = \sum_{i=0}^{n-1} p_{(i,j)} y_i\end{aligned} \tag{11}$$

重复上面的替换过程,每替换一个分量得到的新可行解,与贪心法的解 X_i 相比,新增了一个值相等的分量,最终或者得到一个与 X_i 完全相等的解,或者表明 Y_i 不是最优解。从式 11 可知,每一次分量得到的新的可行解的效益不小于前一步的可行解。所以,X_i 的效益必定不小于 Y_i 的效益,这就证明了贪心法求解得到的数据迁移方案的可行解必定是最优解。

3　实验评估

在实验中,采用 RedisLive 监测整个集群系统,收集节点性能信息以及负载情况。以 Redis-3.2.8 作为评估标准,探讨扩展节点对 Redis 集群整体性能的影响。表 1 显示了实验的软件和硬件的配置信息。在 4 台服务器上分别部署 2 ~ 8 个 Redis 节点组,通过不断增加节点来查看性能情况。单位最大效益迁移算法和 Redis 集群的迁移以及开销敏感迁移算法分别从迁移时间、CPU 使用率和吞吐量进行比较。采用 YCSB (Yahoo! cloud serving benchmark)[13]进行测量。

表 1　实验配置信息

OS	Memory	Network	CPU
CenOS 7.0	16 G DDR3	1 Gbps	Intel(R) Core(TM) i5-4200M CPU

(1)迁移时间。

在实验过程中,记录不同数量节点的迁移的开始时间和负载平衡的结束时间,得到三种算法(Redis、CA 和 Packet)的平均迁移时间。图 4 比较了三种算法的迁移时间。当节点数较少时,三种算法迁移时间大致相同,然而随着节点数的增多,单位最大效益迁移算法越来越优于其他算法。和 Redis 相比,Packet 迁移算法迁移时间优化了 6.2% ~14.1%,和 CA 相比,迁移时间减少了 4.8% ~10.9%。可以想象,当节点较少时,迁移策略节点的选择基本差不多,导致迁移时间没有明显差异,但是当节点较多时,不同迁移策略的选择会导致不同的迁移结果。

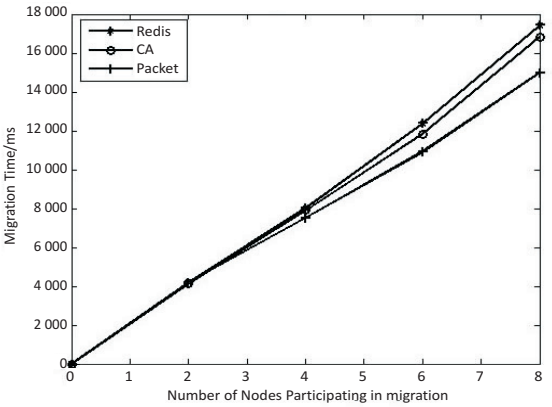

图 4　迁移固定数据量对应的迁移时间比较

(2)CPU 使用。

Redis 集群不断添加新的节点,CPU 的使用率逐渐加大。图 5 给出了三种算法在迁移过程中 CPU 的使用情况。可以看出,无论是 User 使用率还是 Idle 使用率,单位最大效益迁移算法都是有优势的。Packet 迁移算法可以以最小的 CPU 使用率获得最大的迁移效益。当然,随着节点数的增加,Xu 等[14]提到的 CPU 的使用可能会出现瓶颈,但是文中主要研究迁移策略的选择,所以这方面不做考虑。

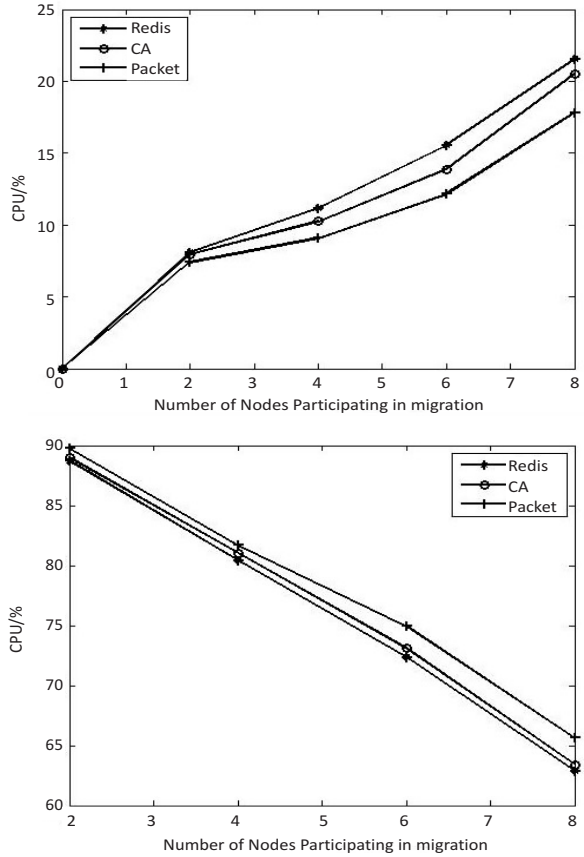

图 5　User 和 Idle CPU 的使用情况

(3)吞吐量。

Redis 采用基于内存的单线程 KV 数据库,官方显示吞吐量可以达到 10^5 左右[3]。图 6 显示了三种迁移算法吞吐量的对比。吞吐量和迁移时间相似,Packet 迁移算法的吞吐量优于其他两种算法。可以看出,Packet 比 Redis 提高了 4.9% ~12.5%,比 CA 优化了 1.8% ~6.7%。随着节点的扩展,Packet 迁移算法会逐渐显示它的优势,节点越多优势越明显。

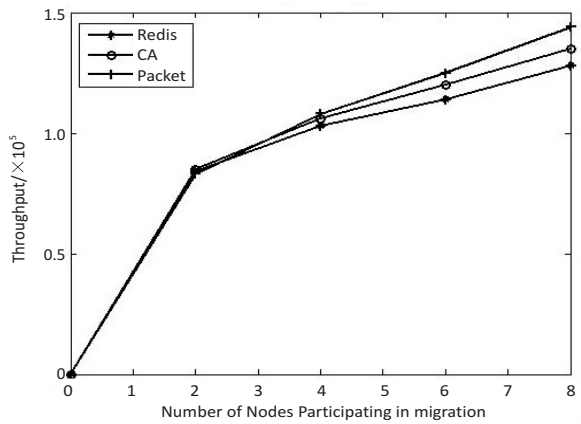

图 6　吞吐量的比较

4　相关工作

为了解决 Key-Value 高可扩展问题,Paiva 等[15]

通过利用本地模式优化副本放置,以减少节点通信代价,于是他们提出两种新的技术。第一,每个节点以分散的方式优化最能产生数据位置的远程操作;第二,一致哈希通过结合一种新的数据结构来提供有效的概率数据放置。而 Miranda 等[16]从数据分片入手,从基于表的、基于规则的和伪随机的散列策略中吸取经验,正确评估随机分片策略,提出一种简单而高效的大数据处理策略。这种随机分片策略维护一个表,这个表里保存着之前存储系统插入和删除操作的信息,这样做减少了传递完美负载分配时所需的随机性数量。百万兆级数据的访问,更多的 Key-Value 存储系统被应用在多核服务器上,常常导致可用带宽的利用率不足,能源效率不高和重载响应时间变长。为了解决这些问题,Xu 等[15]提出了 Hippos,一种高吞吐量、低延迟和高效的 Key-Value 存储模型。Hippos 将 KV 移到操作系统内核中,从而消除了大量网络开销和系统调用。Hippos 使用 Netfilter 框架快速处理 UDP 数据包,几乎完全消除了 UDP 的请求开销,并结合无锁多线程数据访问,Hippos 消除了内部和外部的几个性能瓶颈,大大提高了整体性能。

目前存在的数据库,像关系型数据库和 Key-Value 数据库因缺少低代价的数据迁移技术,从而制约了数据库的弹性扩展。Das 等[17]提出一种迁移技术,称为 Albatross,该技术中持久的数据库映像存储在网络连接的存储中,Albatross 迁移数据库通过缓存和活动事物状态来确保在允许事物时对事务执行的影响最小,同时保证在故障时的正确性。

云服务的普及使得虚拟机技术日益重要,一个重要的特点是虚拟机技术可以用于负载平衡。虚拟机迁移两个评估标准是总体迁移时间和故障停机时间,目前大多数研究会在两者之间做出权衡。Zhang 等[18]从理论上分析了在保证迁移时间和故障停机时间情况下所需要的带宽,接着假设一个确定性模型作为一个简单的例子,这个例子服从伯努利分布,最后在虚拟机运行的工作负载中分析典型的统计特征,并构建基于互惠的工作负载模型,理论上给出满足实时虚拟机迁移的性能指标所需要的带宽值。对于负载管理、节能、日常服务器维护和服务设备质量来说,多层应用程序的实时迁移起着至关重要的作用。和单一虚拟机迁移相比,多个虚拟机迁移是紧密相关的,可能导致一系列相关的迁移问题。在云服务中,Liu 等[19]设计并实现了一种迁移虚拟机相关好友的协调系统,该系统中提出了一种自适应网络带宽分配算法,在迁移完成时间、网络阻塞和迁移故障时间方面减少迁移代价。

而 Basin 等[20]则更加系统详细地介绍了负载均衡策略,提出了一种可以同时支持大规模 scans 和实时访问的原子 KV 映射的 KiWi 模型。KiWi 实现了 scans 无等待和 put 无锁操作,大大提高了处理效率。同时,融入数据结构优化内存管理,通过 chunks 定期维护内存,改善内部组织和空间利用,实现了数据再平衡过程。KiWi 负载再平衡策略分为七个步骤,通过两个主要函数:rebalance 和 normalize,利用 chunk 的三种状态:infant、normal 和 frozen,将不平衡数据实现再平衡。

5 结束语

Redis 最大效益自适应迁移策略是一种提高 Key-Value 存储扩展性的自适应策略,理论分析和实验证明该策略可以完善 Redis 集群节点扩展所带来的不足,提高 Redis 集群系统的性能。Redis 单位最大效益自适应迁移策略有以下两点优势:第一,通过实时判断集群系统负载情况解决 Redis 手动迁移的不足,有策略的数据迁移使其静态存储均衡化,实现系统之间的自适应调控;第二,提出单位效益最大迁移方案来优化迁移策略,此方案每次优先选择单位效益最大的节点进行迁移,直至迁移完成。

但是,Redis 是基于内存的存储系统,迁移过程中不可避免产生一定的网络开销,而如何处理热点数据是另一个难点。文中着重考虑迁移策略的改进,对于其他因素有待更深入的探究。

参考文献:

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

[2] CHODOROW K,DIROLF M. MongoDB:the definitive guide powerful and scalable data storage[M]. Sebastopol:O'Reilly Media,2010.

[3] CARLSON J L. Redis in action[M]. American:Manning Publications Co.,2013.

[4] LAKSHMAN A,MALIK P. Cassandra:a decentralized structured storage system[J]. ACM SIGOPS Operating Systems Review,2010,44(2):35-40.

[5] 黄冬梅,杜艳玲,贺 琪. 混合云存储中海洋大数据迁移算法的研究[J]. 计算机研究与发展,2014,51(1):199-205.

[6] 徐传福,车永刚,王正华,等. 一种均衡可扩展计算机体系结构分布式模拟方法[J]. 软件学报,2014,25(8):1844-1857.

[7] 郑 湃,崔立真,王海洋,等. 云计算环境下面向数据密集型应用的数据布局策略与方法[J]. 计算机学报,2010,33(8):1472-1480.

[8] 秦秀磊,张文博,王 伟,等. 面向云端 Key/Value 存储系统的开销敏感的数据迁移方法[J]. 软件学报,2013,24(6):1403-1417.

(下转第 63 页)

[4] ZENG Deze, GUO Song, HU Jiankun. Reliable bulk-data dissemination in delay tolerant networks[J]. IEEE Transactions on Parallel & Distributed Systems, 2014, 25(8): 2180-2189.

[5] CAO Wei, CAO Guohong, PORTA T L, et al. On exploiting transient social contact patterns for data forwarding in delay-tolerant networks[J]. IEEE Transactions on Mobile Computing, 2013, 12(1): 151-165.

[6] MENG Tong, WU Fan, YANG Zheng, et al. Spatial reusability-aware routing in multi-hop wireless networks[J]. IEEE Transactions on Computers, 2016, 65(1): 244-255.

[7] WANG Ruhai, WEI Zhiguo, ZHANG Qinyu, et al. LTP aggregation of DTN bundles in space communications[J]. IEEE Transactions on Aerospace & Electronic Systems, 2013, 49(3): 1677-1691.

[8] MING Zhongxing, XU Mingwei, WANG Dan. Age-based cooperative caching in information-centric networking [C]//International conference on computer communication and networks. Shanghai, China: IEEE, 2012: 1-8.

[9] CHANG M, PENG S, LIAW J. Deferred-query: an efficient approach for some problems on interval graphs[J]. Networks, 2015, 34: 1-10.

[10] WITTHAUT D, TIMME M. Nonlocal failures in complex supply networks by single link additions[J]. European Physical Journal B, 2013, 86: 377.

[11] TAGUCHI A. Braess′ paradox in a two-terminal transportation network[J]. Journal of the Operations Research Society of Japan, 2017, 25(4): 376-389.

[12] XU Meng, WANG Guangmin, GRANT-MULLER S, et al. Joint road toll pricing and capacity development in discrete transport network design problem[J]. Transportation, 2017, 44(4): 731-752.

[13] 刘　巍,曾庆山. 基于复杂网络的 Braess 悖论现象[J]. 计算机工程与设计,2015,36(4):1098-1102.

[14] 董顺珍. 蜈蚣博弈悖论理性人的认知分析[J]. 长江丛刊,2017(18):148.

[15] 余孝军,黄海军. 交通网络效率的度量和元件重要性的计算方法[J]. 系统工程理论与实践,2012,32(7):1546-1552.

[16] 傅白白,刘法胜. 管理中的 Nash 平衡与 Braess 悖论现象[J]. 运筹与管理,2004,13(1):150-155.

(上接第 58 页)

[9] 董红斌,滕旭阳,杨　雪. 一种基于关联信息熵度量的特征选择方法[J]. 计算机研究与发展,2016,53(8):1684-1695.

[10] 王　康,李东静,陈海光. 分布式存储系统中改进的一致性哈希算法[J]. 计算机技术与发展,2016,26(7):24-29.

[11] 李庆华,李肯立,蒋盛益,等. 背包问题的最优并行算法[J]. 软件学报,2003,14(5):891-896.

[12] 陈慧南. 算法设计与分析:C++语言描述[M]. 北京:电子工业出版社,2006:45-47.

[13] COOPER B F, SILBERSTEIN A, TAM E, et al. Benchmarking cloud serving systems with YCSB[C]//ACM symposium on cloud computing. Indianapolis, Indiana, USA: ACM, 2010: 143-154.

[14] XU Yuehai, FRACHTENBERG E, JIANG Song. Building a high-performance key-value cache as an energy-efficient appliance[J]. Performance Evaluation, 2014, 79: 24-37.

[15] PAIVA J, RUIVO P, ROMANO P, et al. AUTOPLACER: scalable self-tuning data placement in distributed key-value stores[C]//International conference on autonomic computing. New York: ACM, 2014.

[16] MIRANDA A, EFFERT S, KANG Y, et al. Random slicing: efficient and scalable data placement for large-scale storage systems[J]. ACM Transactions on Storage, 2014, 10(3): 9.

[17] DAS S, NISHIMURA S, AGRAWAL D, et al. Albatross: lightweight elasticity in shared storage databases for the cloud using live data migration[J]. Proceedings of the VLDB Endowment, 2011, 4(8): 494-505.

[18] ZHANG Jiao, REN Fengyuan, LIN Chuang. Delay guaranteed live migration of virtual machines[C]//Proceedings of IEEE INFOCOM. Toronto, ON, Canada: IEEE, 2014: 574-582.

[19] LIU Haikun, HE Bingsheng. VMbuddies: coordinating live migration of multi-tier applications in cloud environments [J]. IEEE Transactions on Parallel & Distributed Systems, 2015, 26(4): 1192-1205.

[20] BASIN D, BORTNIKOV E, GOLAN-GUETA G, et al. KiWi: a key-value map for scalable real-time analytics[C]// ACM SIGPLAN symposium on principles and practice of parallel programming. Austin, Texas, USA: ACM, 2017: 357-369.