Redis 集群性能测试分析

柳皓亮，王　丽，周阳辰
（中国科学院电子学研究所苏州研究院 存储计算组，江苏 苏州 215123）

摘　要： Redis 是一个非关系型数据库，属于内存级数据库。但是由于数据量的不断增大，单机的 Redis 物理内存远远无法满足大数据的需要，因此需要搭建分布式的 Redis，可以动态扩展内存，弥补单机 Redis 物理内存不够的缺点。本次测试旨在对 Redis 各方面性能有深入的了解，为今后的工作打好基础。本次实验的目的主要是搭建 Redis Cluster 和 TwemProxy Redis 两种集群，分别对其进行性能测试，测试出集群性能的拐点，找出性能的瓶颈有哪些，并对两套集群进行比较，以便于在不同业务场景下择优选择。

关键词： Redis Cluster；TwemProxy Redis；性能测试
中图分类号： TP23　　**文献标识码：** A　　DOI：10.19358/j.issn.1674-7720.2016.10.024
引用格式： 柳皓亮，王丽，周阳辰. Redis 集群性能测试分析[J]. 微型机与应用，2016，35(10)：70-71，78.

The analysis of the performance test of the Redis Cluster

Liu Haoliang，　Wang Li，　Zhou Yangchen
（Storage Computing Group，Suzhou Institute of Electronics，Chinese Academy of Sciences Research Institute，Suzhou 215123，China）

Abstract: Redis is a non-relational database，which belongs to the memory database. However，with the amount of data increasing quickly，the single Redis is unable to meet the needs of the large data. So we need to build a distributed Redis，which can extend memory dynamicly and make up the faults that the single Redis doesn't have enough physical memory. In order to have a good design for the Redis Cluster and play a Redis high throughput characteristics of Redis Cluster，we make the experiment about this. In this experiment，we build two clusters，which are Redis Cluster and TwemProxy Cluster. We make experiment one by one to test out the clustering performance of inflection point，meanwhile，find out what are the performance bottlenecks. So we can make a good choice under different business scenarios.

Key words: Redis Cluster；TwemProxy Redis；performance test

1　存储测试分析

本次存储测试是用 Java 程序调用 Jedis 提供的 API 向集群里面灌入数据。首先研究灌入少量数据后两种集群的数据分布在哪些节点上，然后研究灌入大量数据后两种集群的落盘情况。

1.1　Redis Cluster

（1）少量数据储存分析

用程序向某一个节点灌入 30 条数据，结果发现每个节点拥有部分数据，数据存储得很分散。由此可知，数据落盘时把一份数据分成多份存储在不同的 Redis 节点上，进行分片存储，通过调研得知这种分配方式是通过 sharding 算法分配[1]的。

（2）大量数据存储分析

首先查看单节点未插入数据前的 rdb 大小为 18 B；然后，用 Java 程序插入 10 万条数据，查看 rdb 大小为 1 289 892 B，然后改用 Java 程序向 Redis Cluster 集群中灌入 10 万条数据，查看每个节点 rdb 文件大小分别为 214 912 B、216 586 B、215 939 B、214 145 B 和 213 757 B。由此可见，单机的 rdb 大小约等于每个 Redis 节点 rdb 大小之和，并且每个节点 rdb 大小相对均衡。综上所述，这种落盘方式把一份数据平均分配到每一个节点上，也就是说每一个节点的 rdb 文件共同组成一份完整的数据。

1.2　TwemProxy Redis

（1）少量数据存储分析

向集群中插入 20 条 key 为 0～19 的数据，查看数据在各个 Redis 节点分布情况，结果发现某个节点存储第 0～9 的数据，另一个节点存储 11～19 的数据，最后一个节点没有存储数据。经过多次相同参数测试，每次落盘结果相同，由此可见 TwemProxy[2] 根据相应算法将数据落盘到各个节点中，并且分配算法是对一段连续的数据进行落盘，而不是对每一条数据进行选择存入到哪个节点中的操作，这样可以减少路由开销。

（2）大量数据存储分析

首先查看单机 Redis 节点未插入数据前的 rdb 文件大

小为 84 B; 然后插入 10 万条数据,查看 rdb 文件大小为 1.6 MB; 接着改用 Java 程序向 TwemProxy Redis[2] 集群中灌入 10 万条数据,查看各各节点 rdb 文件大小分别为 0.49 MB、0.62 MB 和 0.51 MB。由此可见,单机的 rdb 大小约等于每个 Redis 节点 rdb 大小之和,并且每个节点 rdb 大小相对均衡。由此可见,这种落盘方式把一份数据平均分配到每一个节点上,也就是说每一个节点的 rdb 文件共同组成一份完整的数据。

2 使用 Java 代码测试吞吐率

主要从 3 个方面进行测试,当 value 类型分别是 String 类型、list 类型和 map 类型时,统计吞吐率的走势,找出拐点,并分析原因[2]。

2.1 Redis Cluster

(1) String 插入测试——吞吐率随 value 大小变化情况: 当吞吐量一定时并且插入的是 String 类型数据时,如果 value 值在 1 KB 以内时,吞吐率基本保持不变; 如果 value 值大于 1 KB,吞吐率随 value 增大而减小。当 value 值达到 10 KB 且请求总量为 1 万条时,共 100 MB 的数据,内存远远没有被打满,即此时内存的使用率仍比较低,所以此时 Redis 数据存储瓶颈[3] 并不是内存。同时监控了网卡和 IO,发现均处于正常水平,所以也不是这两方面的原因。所以可以推出,此时吞吐率下降是由于 Redis 本身不能够承受过大的 value 值。

(2) String 插入测试——吞吐率随吞吐量变化情况: 当 value 大小一定时,吞吐量的增大对吞吐率没有影响。

(3) String 获取测试——吞吐率随 value 大小变化关系: 结果与(2) 相同。

(4) List 插入测试——吞吐率随 List 大小变化情况: 当List 元素大小和吞吐量一定时,吞吐率随 list 的 size 增大而减小,size 从 10 增加大 100 时吞吐率下降了一半。由此可见,Redis Cluster 对 List 的支持效果并不好,性能有待提升,不建议在以后的项目阶段用 Redis Cluster 存储 List。

(5) List 插入测试——吞吐率随 List 元素字节大小变化情况: List 的元素字节大小变化对吞吐率没有影响。

(6) List 插入测试——吞吐率随吞吐量大小的变化关系: 吞吐率与吞吐量无关。

(7) Map 插入测试——吞吐率随 Map size 大小变化关系: 当吞吐量和元素字节一定时,吞吐率随 Map 的 size 增大而减小。

(8) Map 插入测试——吞吐率随 Map 的 value 大小变化情况: 当吞吐量和 Map 的 size 一定时,吞吐率随 Map 元素字节增大而减小。

2.2 TwemProxy Redis

TwemProxy Redis[2] 采用单条读写和批量读写两种方式进行压力测试,测试结果如下。

(1) String 单条插入测试——吞吐率随 value 大小变化情况: value 值在 1 KB 以内且总请求量为 1 万时吞吐率基本保持不变; 当 value 值大于 1 KB 时,吞吐率随 value 增大而减小,单条 TwemProxy Redis 的插入吞吐率明显比 Redis Cluster 低。

(2) String 批量插入测试——吞吐率随 value 大小变化情况: 当吞吐量一定时,value 值小于 100 B 时,吞吐率随 value 增大而增大; 当 value 值大于 100 B 时,吞吐率随 value 增大而减小。由此可见,批量插入存在极值点,此外批量插入的吞吐率远远高于 TwemProxy Redis 和 Redis Cluster 的单条插入。

(3) String 单条获取测试——吞吐率随 value 大小变化关系: 测试结果与(1) 的结果相同。由此可见,TwemProxy Redis 的单条读写效率一致。

(4) String 批量获取测试——吞吐率随 value 大小变化关系: 结果与(2) 相同。

(5) String 单条插入测试——吞吐率随吞吐量的变化关系: 吞吐率与吞吐量无关,TwemProxy Redis 吞吐率只有 Redis Cluster 的一半,明显吞吐率很低。

(6) String 批量插入测试——吞吐率随吞吐量的变化关系: 随着吞吐量的增加,吞吐率也在增加。但在测试时将请求量给到 10 万条后,程序宕掉并且集群服务停止工作,说明 pipeline 批量打包的数据量有限,即性能是有限的。但是可以通过打包多次解决这个问题,批量插入的吞吐率明显高于 TwemProxy Cluster 和 Redis Cluster 的单条插入吞吐率。

(7) List 和 Map 类型的单条插入测试吞吐率变化: 吞吐率变化与 Redis Cluster 的相同,但是吞吐率低于 Redis Cluster。

(8) List 和 Map 类型的单条插入测试吞吐率变化: 吞吐率变化与 Redis Cluster 的相同,但是吞吐率高于 TwemProxy Cluster 和 Redis Cluster 的单条吞吐率。

3 结论

(1) TwemProxy Redis 的批量读写吞吐率远远高于 Redis Cluster 单条的吞吐率,Redis Cluster 单条读写的吞吐率略高于 TwemPrxoy Redis 单条吞吐率。

(2) Redis Cluster 和 TwemPrxoy Redis 对 List 和 Map 集合的吞吐率很低,不建议存储这两种类型的数据。

(3) 当需要进行 TwemProxy Redis 批量操作时,需要通过程序保证一次批量读写的数据量不宜过大,否则底层服务会宕掉。

参考文献

[1] 王敏,陈亚光. 数据库系统辅助测试工具[J]. 微型机与应用,2013,32(3):13-15,18.

(下转第 78 页)

表 3 IALM 算法与基于 CULA 优化后算法不同分辨率视频仿真数据比较

视频图像分辨率	IALM 运算时间/s	基于 CULA 优化后的 IALM 运算时间/s	加速倍数
480 ×360	24.90	20.43	1.22
640 ×480	49.80	40.06	1.24
768 ×576	71.92	57.41	1.25

表 1 表明,优化改进后的算法运行效率有所提升。表 2 取不同的视频帧数进行前景目标检测,表明实现了至少 1 倍以上加速比。最后表 3 给出了不同测试视频的运行结果,表明随着图像尺寸的加大,加速比增大。综合得知,本文的优化改进算法效率更高。

4 结论

本文应用 CULA 加速优化后的低秩矩阵恢复算法实现前景目标检测,加速优化后的算法与原有算法相比,奇异值分解次数减少,表明算法里的奇异值分解计算得到加速改进,因此算法的运行时间得以较大幅度缩短。仿真结果表明,本文加速优化后的算法有更好的前景目标检测效率。本文的前景目标检测算法在计算效率上得到改善,发挥了软硬件结合的效果,这种学习成本低廉、使用代价较小、运行效率较高的优点弥补了 MATLAB 计算速度上的缺点,也使得视觉应用的研究有了良好的开端。

参考文献

[1] CANDES E J, Li Xiaodong, Ma Yi, et al. Robust principalcomponent analysis [J]. Journal of the ACM, 2009, 58(3): 233-279.

[2] WRIGHT J, GANESH A, RAO S, et al. Robust principal component analysis: exact recovery of corrupted low-rank matrices via convex optimization [C]. Proceedings of Neural Information Processing Systems (NIPS), 2009, 87(4): 20: 3-20: 56.

[3] NVIDIA Corporation. NVIDIA CUDA programming guide [Z]. 2009.

[4] Lin Zhouchen, Chen Minming, Ma Yi. The augmented Lagrange multiplier method for exact recovery of a corrupted low-rank matrix [R]. Eprint Arxiv, 2010.

[5] GANESH A, Lin Zhouchen, WRIGHT J, et al. Fast algorithms for recovering a corrupted low-rank matrix [C]. International Workshop on Computational Advances in Multi-Sensor Adaptive Processing, 2009: 213-216.

[6] Lin Zhouchen, GANESH A, WRIGHT J, et al. Fast convex optimization algorithms for exact recovery of a corrupted low-rank matrix [C]. Proceedings of Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2009.

[7] YUAN X, YANG J. Sparse and low-rank matrix decomposition via alternating direction methods [R]. Hong Kong: Hong Kong Baptist University, 2009.

[8] Chen Minming. Algorithms and implementations of matrix reconstruction [D]. Beijing: Graduate School Chinese Academy of Science, 2010.

[9] EM Photonics Corporation. CULA reference guide [Z]. 2014.

[10] EM Photonics Corporation. CULA programmers guide [Z]. 2014.

(收稿日期: 2016-01-23)

作者简介:

李俊(1992-),男,硕士研究生,主要研究方向: 图像处理与模式识别。

周薇娜(1982-),通信作者,女,博士,讲师,主要研究方向: 图像处理,集成电路设计。E-mail: wnzhou@shmtu.edu.cn。

(上接第 71 页)

[2] 夏文忠,邹雯奇. 基于 X86 平台的高性能数据库集群技术的研究 [J]. 微型机与应用,2015,34(1): 36-39,46.

[3] 张蕾,侯瑞春,丁香乾,等. 会话保持机制在集群系统中的应用研究 [J]. 微型机与应用,2015,34(9): 32-34,50.

(收稿日期: 2016-01-20)

作者简介:

柳皓亮(1991-),男,硕士研究生,主要研究方向: 分布式在线实时流式计算。

王丽(1992-),女,硕士研究生,主要研究方向: 分布式数据挖掘与存储。

周阳辰(1992-),男,硕士研究生,主要研究方向: 集群监控。

(上接第 74 页)

[13] 刘金琨. 滑模变结构控制 MATLAB 仿真 [M]. 北京: 清华大学出版社,2005.

[14] 吴卫民,刘松培,何远彬. 单相 LCL 并网逆变器电流控制综述 [J]. 电源技术,2011,34(2): 52-58.

(收稿日期: 2016-02-23)

作者简介:

姜慧鹏(1990-),男,硕士研究生,主要研究方向: 电力电子及其控制技术。

刘宜成(1975-),男,博士,副教授,主要研究方向: 非线性控制。

蒲明(1981-),男,博士,讲师,主要研究方向: 滑模控制。