高性能高可用 Redis 客户端的设计与实现*

刘世超[1,2],杨 斌[1,2],刘卫国[1,2]
(1.山东大学 软件学院,山东 济南 250101;2.国家超级计算无锡中心,江苏 无锡 214072)

摘 要:Redis 是一个基于内存存储的非结构化数据库,以高 I/O(Input/Output)性能和高响应速度著称,在数据缓冲、消息队列、Key-Value 存储等场景都发挥着重要的作用。在其支持的众多客户端中,C/C++客户端 Hiredis 的应用尤为广泛。对 Hiredis 库做了深入分析,发现了其管道功能存在高开销、指令存储不当以及内存混淆问题。基于此,在 32 逻辑核的 X86 架构处理器以及 64 GB 内存的 Linux 服务器上,设计并实现了一个面向 C/C++的高性能高可用 Redis 客户端,通过内存预分配以及内存隔离的方法提高了大量指令批处理时的性能并解决了复杂场景下的内存混淆问题。经测试,新客户端提高了 3~7 倍的指令执行效率,同时也保证了复杂场景下的内存安全以及数据准确性。

关键词:Redis;管道;Hiredis;内存混淆;性能优化
中图分类号:TP391 **文献标识码**:A DOI:10.16157/j.issn.0258-7998.212432

中文引用格式:刘世超,杨斌,刘卫国. 高性能高可用 Redis 客户端的设计与实现[J].电子技术应用,2022,48(1):46-52,58.
英文引用格式:Liu Shichao,Yang Bin,Liu Weiguo. Design and implementation of high-performance and high-availability redis client[J]. Application of Electronic Technique,2022,48(1):46-52,58.

Design and implementation of high-performance and high-availability redis client

Liu Shichao[1,2],Yang Bin[1,2],Liu Weiguo[1,2]
(1.School of Software,Shandong University,Jinan 250101,China;2.National Supercomputing Center in Wuxi,Wuxi 214072,China)

Abstract:Redis is an unstructured database based on memory storage. It is known for high I/O(Input/Output) performance and high response speed. It plays an important role in data buffering, message queues, key-value storage and other scenarios. Among the many clients it supports, the C/C++ client Hiredis is particularly widely used. This article did an in-depth analysis of the Hiredis library and found that its pipeline function has high overhead, improper instruction storage, and memory confusion problems. Based on this, this article designs and implements a C/C++-oriented high-performance and high-availability Redis client on a 32-core X86 architecture processor and a 64 GB memory Linux server. It improves the performance of processing a large number of instructions and solves the problem of memory confusion in complex scenarios through memory pre-allocation and memory isolation. After testing, the new client has improved instruction execution efficiency by 3~7 times, while also ensuring memory safety and accuracy in complex scenarios.

Key words:Redis;pipeline;Hiredis;memory confusion;performance optimization

0 引言

随着互联网飞速发展以及大规模应用的不断涌现,目前已经步入了大数据时代。非结构化数据逐渐替代了传统结构化数据并迅速占据了主导地位,为了管理形式多样的非结构化数据,涌现了诸如 MongoDB[1]、InfluxDB[2]、Elasticsearch[3]等十分有代表性的数据库。这些数据库虽然针对非结构化数据的存取做了很多优化,但是受限于硬盘(Hard Disk Drive,HDD)等底层存储介质,往往无法满足高性能场景的需求。

为了提高性能,以 Redis 为代表的内存数据库应运而生。Redis 是一个非结构化数据库,支持使用非结构化语言(Not-only Structured Query Language,NoSQL)查询。同时,Redis 通过 I/O(Input/Output)多路复用和 DRAM(Dynamic Random Access Memory)提供了高吞吐、高并发和低时延的服务,在数据缓冲、消息队列、Key-Value 存储等场景都发挥了重要的作用。

但是随着大规模计算集群的算力逐渐增大,应用的数据规模也随之变大,计算和 I/O 之间的“存储墙”也变得愈发明显。现有的 Redis 也遇到一些网络和存储方面的问题。因此如何改进 Redis 也受到了广泛的重视,随之出现了很多 Redis 优化的相关工作,它们从各种角度对 Redis 服务端或客户端做了改进。

*基金项目:国家重点研发计划(2020YFB0204800);无锡市太湖人才计划创新领军人才项目(融合人工智能技术的新一代精细化区域气候预测系统研制)

绝大多数优化工作侧重于 Redis 服务端的优化。Wu[4]等人发现了 Redis 的 I/O 多路复用模型中的冗余监听问题，于是设计研发了新模型 Flexpoll，它根据系统负载情况动态地管理监听事件，在一定程度上降低了原有 epoll 模型的开销。Tang[5]、Mitchell[6]、Kalia[7]以及 Wang[8]等人利用 IB(Infiniband)网络下的远程内存访问(Remote Direct Memory Access，RDMA)技术，对 Redis 的网络通信接口做了改进，大大提升了 Redis 的网络性能。Liu[9]等人洞察到 Redis 内存管理中的碎片问题，于是他们重新设计了一套更易对齐的内存分配策略，同时他们基于 xxHash 函数构建了双层 hash 索引，提高了 hash 命中率。Zhang[10]等人创新性地利用机器学习，预测 Redis 的键位分布以及 rehash 周期，从而预见性地进行内存扩展，有效避免了大规模 rehash 带来的系统瘫痪风险。同样着眼于淘汰策略优化的还有 Hyperbolic Caching(HC)[11]和 pRedis[12]，它们将“未命中代价”纳入置换算法的考虑范畴，大大降低了缓存未命中的开销。

与此同时，只有少部分工作侧重于 Redis 客户端的优化。其中，Chen[13]等人发现了 Redis 集群访问的二次链接开销，通过在客户端缓存 Key-to-Node 映射表的方式，提升了近 1 倍 I/O 性能。

经过调研可以看出，关于 Redis 服务端的优化工作层出不穷，而着眼于客户端的工作却少之又少。但是用户往往都需要通过客户端来与 Reids 进行交互，客户端的性能决定了应用最终的性能。在众多 Redis 客户端中，Hiredis 是应用最为广泛的 Redis 客户端。本文以 Hiredis 为例对 Redis 的客户端展开深入的分析，并结合部署于 E 级超级计算机上的 I/O 性能监控与分析诊断系统 Beacon[14]的业务情景，发现了 Hiredis 的管道功能存在高开销、指令存储不当以及内存混淆问题。基于此，本文在 32 逻辑核的 X86 架构处理器以及64 GB 内存的 Linux 服务器上进行研发，设计并实现了一个面向 C/C++的高性能高可用 Redis 客户端，通过内存预分配以及内存隔离的方法提高了处理大量指令的性能，并解决了复杂场景下的内存混淆问题。

1 背景介绍

1.1 Redis

Redis 是一个使用 C 语言实现的轻量级内存数据库，广泛应用于数据缓冲、消息队列、Key-Value 存储等情景，且相较于传统的数据库拥有许多优势。一方面，Redis 以内存访问代替了传统磁盘访问，提升了 I/O 性能；另一方面，Redis 提供 Hash 访问的机制，降低了数据检索的复杂度。为了保证容灾和数据安全，Redis 提供了以磁盘为辅的数据备份功能。当数据量达到设定的内存阈值后，Redis 会依据设置的淘汰策略进行置换。此外，Redis 还提供了完善的集群管理功能来提高大规模集群管理的效率。用户可根据自己的需求搭建 Redis 集群，并设置主从节点间的数据复制策略等。

1.2 管道

Redis 是一个 CS(Client/Server)模式的服务，客户端需要与服务端建立链接才可以发送请求。在传统的 CS 模式中，往往采用了阻塞式的交互流程，如图 1 所示。客户端每次发送请求后，需要等待请求结果返回才可以发送下一个请求。在需要执行大量没有依赖关系的指令的场合，这种阻塞式交互无疑是效率低下的。

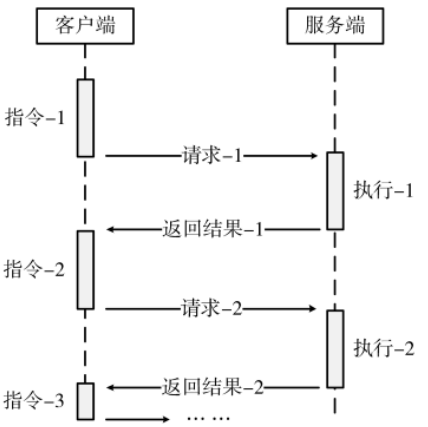

图 1 传统阻塞式 C/S 请求流程示意图

为了应对指令批处理的情景，Redis 引入了管道功能，它的实现逻辑如图 2 所示。在管道中，Redis 首先将客户端请求的批处理指令缓存起来，待执行完毕后，再将结果以队列形式统一返还。这种非阻塞的 I/O 多路复用模型的引入，有力保障了 Redis 高性能、高并发和低延迟的特性。

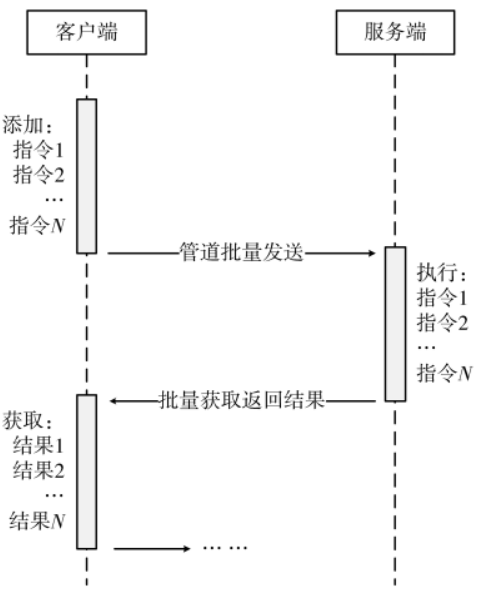

图 2 管道非阻塞式 C/S 请求流程示意图

1.3 Redis 客户端

通常说的 Redis 往往指 Redis 服务端，提供数据存储和查询服务。而 Redis 客户端在整个服务体系中也是极

其重要的一环，例如：常见的 Jedis、Redisson 以及 Lettuce。应用往往需要通过客户端来向服务端请求数据的存取服务。

其中，Hiredis 是应用广泛的 Redis 客户端之一，为 C/C++操作 Redis 提供了许多便捷的接口，同时也是一些其他客户端的下层依赖，例如：Python 中的 Hiredis-py，C++中的 Redis-plus-plus 等。Hiredis 通过 redisConnect 函数与 Redis 服务端建立链接，之后，若需执行某条指令，用户只需将指令字符串作为参数传入函数 redisCommand 即可，而指令的执行结果包含于返回值 redisReply 对象中。与此同时，Hiredis 客户端也支持管道功能。用户可通过 redisAppendCommand 函数将指令暂存至缓存区(obuf)，最后调用 redisGetReply 函数将缓存区中的指令集发送至服务端，并可获取到首条指令的执行结果，至于剩余指令的返回值，只需依次调用 redisGetReply 获取即可。

2 Hiredis 问题分析

由于 Hiredis 的应用较为广泛，且功能较为完善，故本文以 Hiredis 为例，进行了深入的测试分析。研究发现 Hiredis 在高并发和复杂场景下仍然存在明显的性能及准确性问题。

2.1 大量批处理指令执行的场景

redisAppendCommand (context，cmd_str) 函数是 Hiredis 实现管道功能的核心函数。其中参数 context 是由 redisConnect 返回的 redisContext 对象，它代表着一次 Redis 链接的上下文，其中所含的一块连续内存 obuf 作为批处理指令的缓存池使用。这块内存并非用户管理，而是由函数内部负责分配和释放。当若干指令依次写入缓存区时，每条指令之间会以符合 Redis 协议的特定符号分隔。

该方案问题在于，每次向缓存区追加指令时，都面临着内存的扩展问题。如图 3 所示，当第 N 条指令加入时，则需要先分配一块足以包含这 N 条指令的内存，再将原来的 $N-1$ 条指令拷贝到新区域，最后释放原来的内存，并把第 N 条指令追加上去。不难看出，内存拷贝的复杂度为 $O(N^2)$。

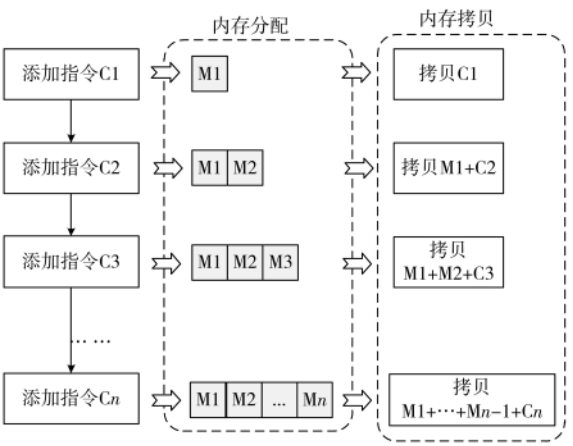

图 3 Hiredis 批处理指令内存分配示意图

在 Beacon 的业务场景中，需要使用 HMSET 指令向 Redis 频繁写入文件 I/O 信息，其中包含长达 MB 级别的文件信息，代表特定时间内的所有文件描述符(File Descriptor，FD)及其文件名。在这样的长指令面前，Hiredis 客户端的内存开销可想而知，时间复杂度亟待优化。

2.2 批处理与即时指令混合执行的场景

上文提到的是简单的批处理场景，只需一次性地将若干指令提交即可。而实际应用场景往往比较复杂，存在批处理指令中穿插使用即时指令的情况，即在批量添加指令的过程中需临时向 Redis 请求一条指令，并立即返回处理结果。

这时，首先使用 redisAppendCommand 添加若干指令，然后穿插使用 redisCommand 提交即时指令，但后者实际上是调用了 redisAppendCommand 并立即使用 redisGetReply 获取返回值。如此一来，便会提前地把缓存区中的指令提交出去，并得到错位的返回结果，如图 4 所示。

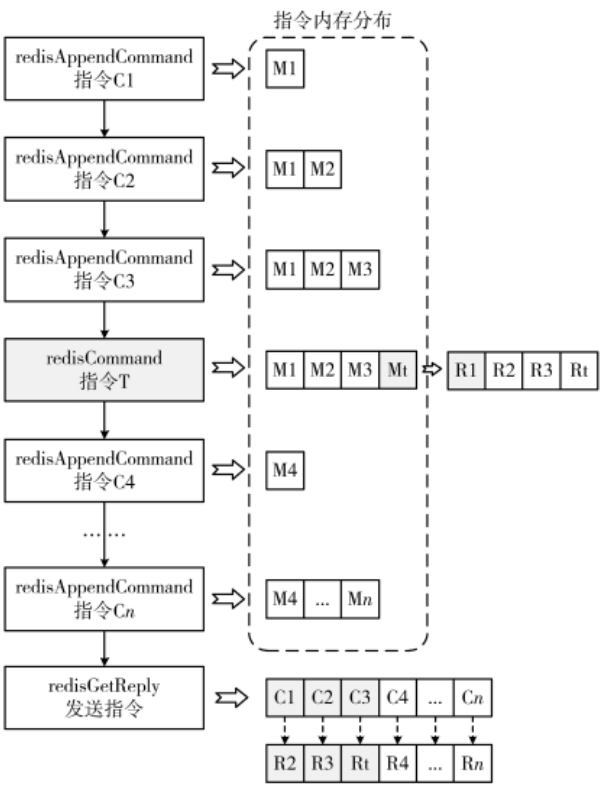

图 4 Hiredis 批处理与即时指令内存混淆示意图

首先添加 3 条批处理指令 C1、C2、C3，之后再提交一条即时指令 T。此时 Hiredis 会先将指令 T 追加到缓存区中，然后自动调用 redisGetReply 函数，并期望获得指令 T 的对应结果 Rt。但实际上，它却在不知情的情况下将指令 M1、M2、M3、Mt 一起提交出去，并返回首条指令的结果。如此一来，指令 T 便误收了指令 C1 的结果 R1。而剩余的指令结果 R2、R3、Rt 此时仍存储在返回队列中。于是当再追加指令 C4，C5，…，Cn，最终提交并依次遍历返

回队列时，却收到了错误的 R2，R3，Rt，R4，…，Rn，而不是预期的 R1，R2，R3，R4，…，Rn，即 T 指令之前的批处理指令所获得的返回值全部错位。

2.3 多线程批处理指令执行的场景

虽然 Redis 是单线程处理模型，但在客户端编程时往往会使用多线程的模式。在使用 Hiredis 客户端向同一个 redisContext 添加指令时，目前的 Hiredis 无法返回各线程所期望的结果，如图 5 所示。

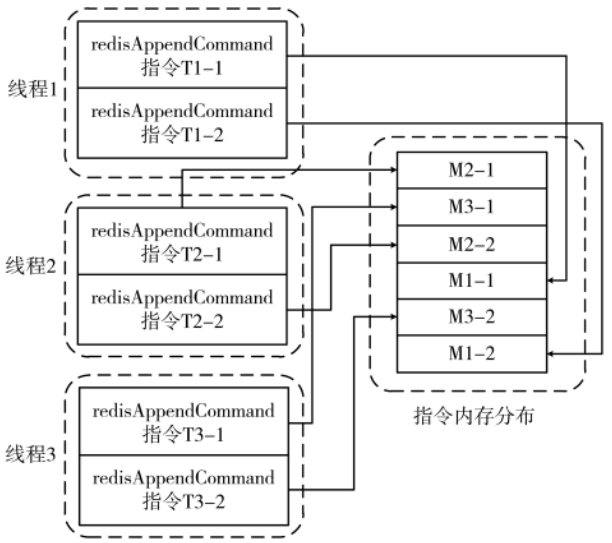

图 5 Hiredis 多线程批处理指令内存混淆示意图

可以看出，即使线程 T1、T2、T3 通过加锁解决了线程安全问题，并成功将自己的指令 Tx-1、Tx-2 写入了缓存，却难以保证指令在缓存区中的顺序。而且同理，返回值队列也存在着同样的顺序混淆问题。如此一来，各线程的指令与其结果的一致性便得不到保证。

3 高性能高可用的 Redis 客户端设计与实现

为了解决上述问题，本文设计并实现了一个高性能高可用的 Redis 客户端，通过批处理指令的内存预分配，大大降低了内存拷贝的开销；同时隔离了批处理指令与即时指令的缓存区，避免了两者的混淆；最后，也针对多线程混淆的情景，设计了合适的内存分配与隔离策略，保证了多线程的安全性和准确性。

3.1 内存预分配策略

对于 redisAppendCommand 带来的频繁内存拷贝问题，采取的策略如图 6 所示。首先，将指令缓存区提取出来，允许用户根据自己的预期在调用前预分配一块连续的、足以装若干条指令的内存；其次，通过游标(pos)去跟踪字符串尾的位置，这样若需后续的指令追加，只需从游标处继续拼接即可。

如此一来，不仅将缓存区的遍历复杂度降为 $O(N)$，而且有效地避免了内存的被动扩展所带来的拷贝开销，真正实现了“零拷贝”。

在 Beacon 中，以一分钟的 I/O 日志为一个处理批次，

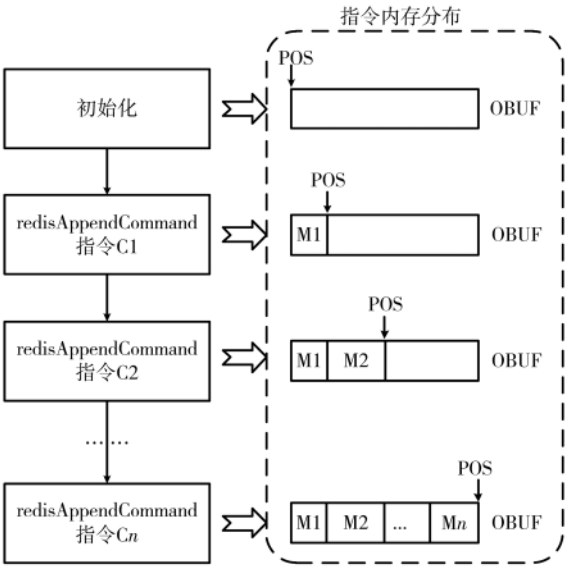

图 6 优化后的批处理指令内存预分配示意图

每批次的日志理论峰值为 GB 级别，显然，在程序中一次性分配如此多的堆内存是不合理的，因为系统不会一直保持全机峰值状态。于是每次分配 15 MB 的缓存空间，保证其足够容纳单条指令；同时为了避免指令拼接过程中的内存越界问题，每一次追加前都会进行预判，判断缓存区的剩余空间是否足够容纳本条指令。若溢出，则首先将现有指令集提交出去，将缓存清空且游标回归原点，再继续重复上述操作。权衡之下，只需在处理高密度日志时多提交几次，牺牲一点网络上的性能，便可解决内存分配过多时开销过大与分配过少时越界溢出之间的矛盾。

3.2 批处理与即时指令的内存隔离策略

在批处理指令与即时指令混合使用的情景下，为了避免混淆，考虑过以下几种解决方案。

首先，尝试仍然使用同一块缓存区来存储批处理与即时指令，但为了同时保证前者的顺序性与后者的即时性，需要使用队列与堆栈复合的数据结构存储。即当面对批处理指令时，仍然使用队列的方式向队尾添加指令，而面对即时指令时，会将其插入到队首，这样，该即时指令便可在不影响批处理指令顺序的情况下获得自己期望的结果。但是，如果频繁地向连续内存的头部插入数据，势必造成大量的内存移动，这有违初衷。其次，尝试使用不同的 Redis 上下文对象去隔离，即可避免缓存干扰问题。但该对象是与特定 Redis 链接一一对应的，无形中增加一倍的网络链接负载也并非我们所期望的。最终权衡之下，设计了如图 7 所示的策略。

本文在同一个链接的上下文中增加一个临时缓存区(obuf_tmp)，当有临时指令请求时，使用新的临时缓存区来存储它们，而对于批处理指令，仍然将它们存储到原本的缓存区(obuf)中去。这样便实现了两者的内存隔

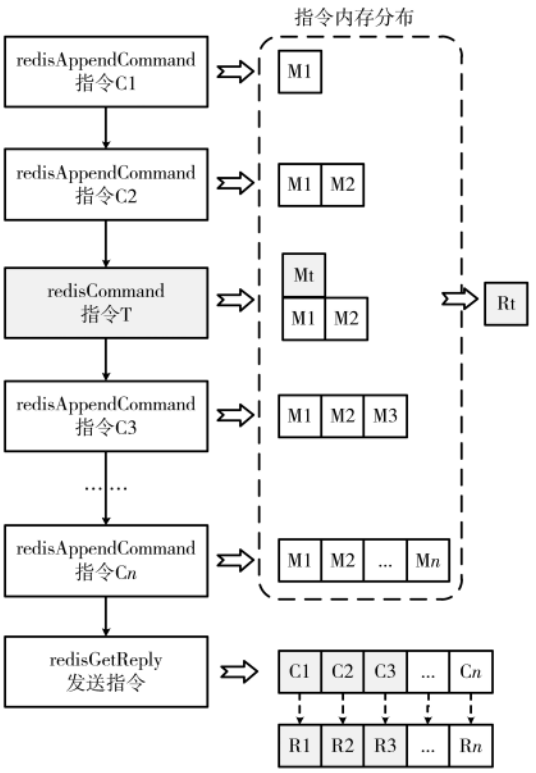

图 7　优化后的批处理与即时指令内存隔离示意图

离，且不会增加额外的网络开销。

3.3 多线程批处理指令的内存隔离策略

在解决多线程的内存混淆问题时，首先考虑到的仍是使用同一块缓存区(obuf)的情况，这时为了区分各线程的指令集，需要在每条指令中增加线程编号字段，并且一直关联到返回的结果集中去。

这样实现的问题在于，当获取指令执行结果时，需要以线程编号作为参考去遍历结果队列，直到匹配到对应的返回值为止。例如有 T 个线程，每个线程各添加 N 条指令，这时缓存区的大小为 $T\times N$ 条指令的大小，返回队列中共有 $T\times N$ 条记录。当 X 线程期望获得自己的执行结果时，需要遍历这 $T\times N$ 条记录，直到匹配到线程编号字段为 X 停止。不难看出，在最坏的情况下某线程的对应结果集沉底，那么它每次调用获取结果的函数，都需要进行 $(T-1)\times N$ 次遍历，最终平均复杂度为 $O(T\times N)$。所以，这样设计反而增加了开销，故舍弃。再者，使用多个链接上下文来避免缓存区混淆的策略也行不通，同理于前文，此方案会增加 T 倍的 Redis 链接开销。

经过以上分析，本文决定仍然在同一个 Redis 链接上下文中做改进，策略如图 8 所示。新设计的缓存区是一个字符串的一维数组，下标索引与线程的相对编号对应，数组中每个元素存储着对应线程的指令队列。

当某线程在提交批处理指令时，首先以自己的线程相对编号为索引，找到自己的缓存区下标位置，经过二次寻址后，再向其中追加对应的指令。同理返回结果的

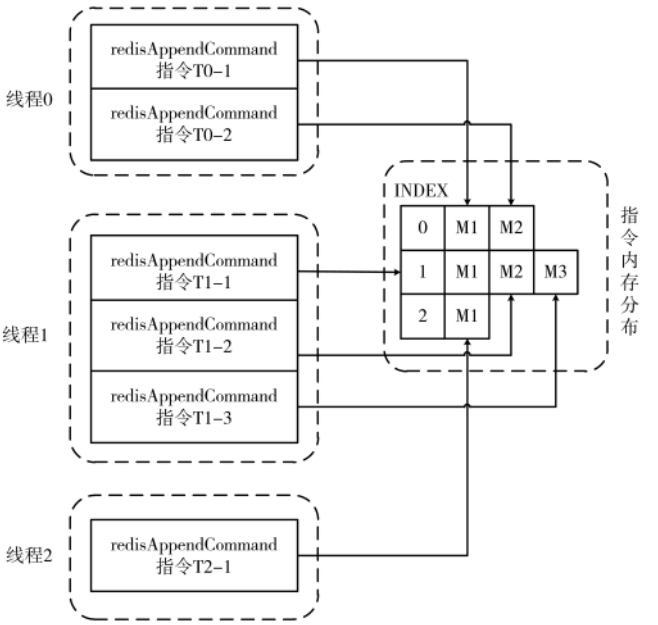

图 8　优化后的多线程批处理指令内存隔离示意图

队列也相应增加一维，通过线程编号索引自己的结果集起始地址，再从中依次获取首条指令的结果。如此一来，在不增加额外网络开销的情况下，便实现了各线程缓存区之间的安全隔离。

4 高性能高可用 Redis 客户端测试分析

本文的测试平台是 Linux 服务器，操作系统为 CentOS 7.7，X86_64 核心，内存为 64 GB。同时，使用 V4.8.5 的 GCC 编译器以及 fPIC 和 shared 选项，对 Redis 客户端进行动态库编译。最后，Redis 服务端的 IP 绑定为本地(127.0.0.1)，端口号为 5555，设置 Redis 的最大内存容量为 1 GB。

测试工作主要分两个部分，第一是性能测试，目的是验证内存预分配策略带来的执行效率提升；第二是准确性测试，验证内存隔离策略的有效性。

4.1 性能测试

本测试的主要目的是用来验证内存预分配策略所带来的性能提升，图 9 展示了测试结果。测试流程如下，首先初始化一个 Redis 服务，然后分别使用 Hiredis 客户端与本文设计的高性能高可用 Redis 客户端进行批处理指令请求。实验总共分为 4 组，每组均传入 10 条大小相等的指令，各组的指令大小分别为100 KB、1 MB、10 MB、100 MB，同时进行毫秒级别的计时。每组实验均重复 5 次，去掉最高最低值后计算平均值，即得到最终的实验结果。

图 9 中的横轴表示测试指令的大小，纵轴表示执行时间(毫秒)，深、浅两条柱子分别代表优化前、后的性能。可以看出，优化前的时耗随着指令字节数的增大，基本呈指数爆炸式增长，当指令为100 MB 时其时耗高达 16 s；相对的，优化后的时间呈线性增长态势，随着指令

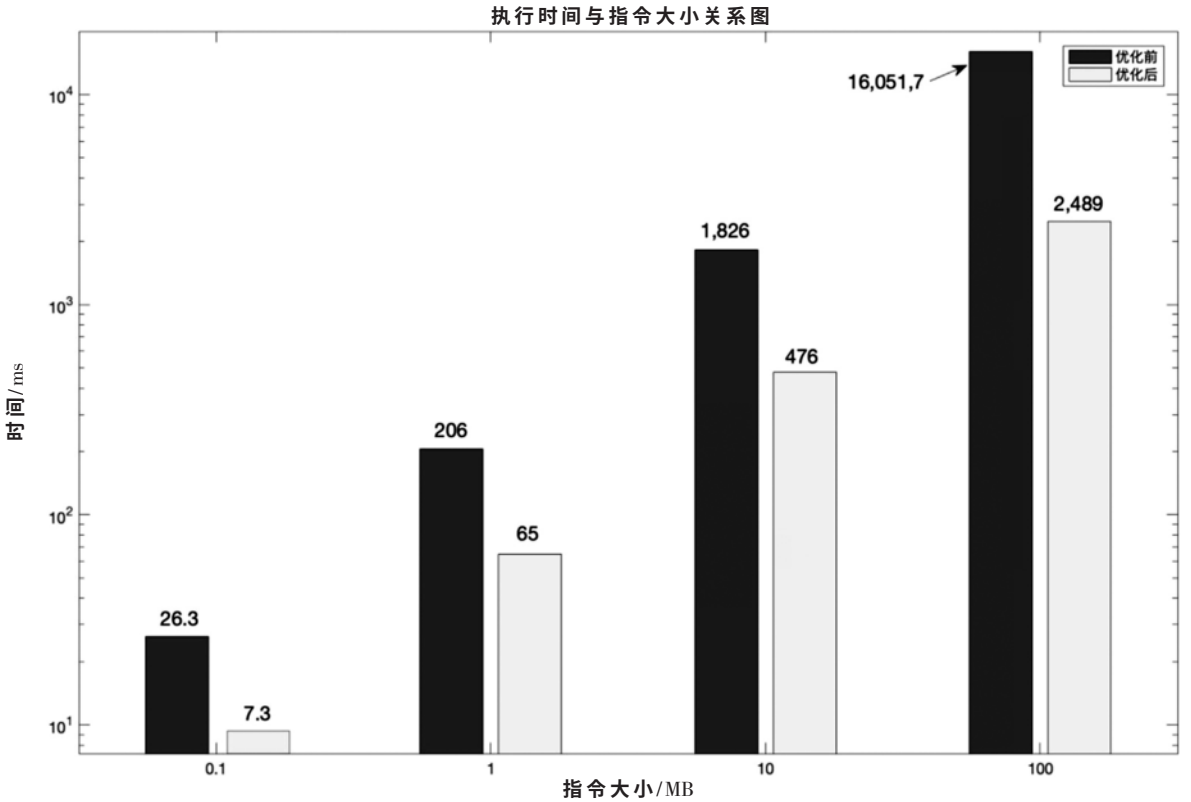

图 9 优化前后的批处理指令性能比较

大小以 10 倍速度扩增，其执行时间分别按 9 倍、7 倍和 5 倍的速度增大，时耗增加的速度逐渐变缓，与优化前的性能表现出了强烈的对比。当指令达 100 MB 时，优化后的策略只需 2.5 s，相对于优化前的 16 s 有了高达 6.4 倍的提升。

本文并没有继续测试更高规模的指令，一方面，考虑到 Hiredis 的开销已经不足以运行更大规模的用例；另一方面，100 MB 的单条指令大小已经足够覆盖绝大多数的应用场景。但通过以上的分析可以看出，在面向大量指令时，本文的 Redis 客户端可以为系统性能带来质的飞跃。

4.2 准确性测试

本测试的主要目的是验证内存隔离的准确性。测试前，首先在 Redis 中写入 6 条数据，如表 1 所示。

表 1 测试前 Redis 中 Key-Value 数据信息

Key	Value	类型
K1	Kv1	String
K2	Kv2	String
K3	Kv3	String
K4	Kv4	String
T1	Tv1	String
T2	Tv2	String

为验证批处理与即时指令的内存隔离策略，依次添加批处理指令 Get K1、Get K2、Get K3、Get K4，并在其中穿插两条即时指令 Get T1、Get T2，最后连续 4 次调用 redisGetReply，观察指令结果的返回情况。

表 2 展示了该情景下隔离前、后的结果，前两列表示指令及其提交方式，后两列分别表示隔离前、后的返回值。可以看出，内存隔离前的返回结果是异常混乱的，而在隔离后，结果队列 Kv1、Kv2、Kv3、Kv4 与批处理指令 Get K1、Get K2、Get K3、Get K4 的顺序完全一致，且两条即时指令 Get T1、Get T2 也如期获得了对应的结果 Tv1、Tv2。

表 2 优化前后批处理与即时指令混用返回值情况

指令	提交方式	隔离前	隔离后
Get K1	批处理	N/A	N/A
Get K2	批处理	N/A	N/A
Get T1	即时	Kv1	Tv1
Get K3	批处理	N/A	N/A
Get K4	批处理	N/A	N/A
Get T2	即时	Kv2	Tv2
redisGetReply	顺序执行 4 次	Tv1，Kv3，Kv4，Tv2	Kv1，Kv2，Kv3，Kv4

如表 3 所示，为验证多线程批处理提交指令时的内存隔离策略，在测试用例中使用了 3 个线程，共同向同一次的链接上下文中提交批处理指令。每个线程提交两条，并期望获得与之对应的返回结果。

从表 3 中不难看出，缓存隔离前的返回值异常混乱：线程 0 的 Get T1、Get T2 指令得到的结果是 Kv1、Tv2；线程 1 的 Get K1、Get K2 得到的结果是 Kv3、Kv4；而线程 2 的 Get K3、Get K4 却得到了 Kv2、Tv1 的返回值。这其中只有线程 0 的 Get T2 指令收到了正确的反馈，且

表 3 优化前后的多线程批处理指令返回值情况

线程号	指令	隔离前	隔离后
线程 0	Get T1	N/A	N/A
	Get T2	N/A	N/A
	redisGetReply	Kv1，Tv2	Tv1，Tv2
线程 1	Get K1	N/A	N/A
	Get K2	N/A	N/A
	redisGetReply	Kv3，Kv4	Kv1，Kv2
线程 2	Get K3	N/A	N/A
	Get K4	N/A	N/A
	redisGetReply	Kv2，Tv1	Kv3，Kv4

每次实验结果几乎各不相同，这是由于指令和结果集的内存混淆所致。当使用本文设计的 Redis 客户端进行测试时，线程0、1、2 分别得到了 Tv1、Tv2、Kv1、Kv2、Kv3、Kv4 的返回值，与各自的指令提交顺序完全一致。

5 结论

以 Redis 为代表的内存数据库由于高性能、低延迟、非结构化存储等特性，已成为当今大数据时代的宠儿。如何用好这些数据库也成为了大家关注的重点。本文以 Redis 客户端中应用最为广泛的 Hiredis 为例开展了深入分析，发现其在高并发高性能应用中存在一些问题，包括：面向大型指令时的高内存开销问题，在复杂情景下的内存混淆问题等。为了解决这些问题，本文设计并实现了一个高性能高可用的 Redis 客户端，并且借助于真实的应用场景做了深入验证。

虽然本文的工作主要以 Beacon 为例展开，但是也具有通用性，同样也可推广到其他使用 Redis 的业务中去，例如常见的日志处理系统、电商交易平台以及社交文娱业务等。总之，在面向大数据的高性能处理情景时，本文的工作可保证系统的高性能、高并发、低开销。下一步计划将该客户端应用到其他基于 Redis 服务的高性能系统中去。

参考文献

[1] GYŐRÖDI C，GYŐRÖDI R，PECHERLE G，et al.A comparative study：MongoDB vs.MySQL[C]//2015 13th International Conference on Engineering of Modern Electric Systems (EMES).IEEE，2015：1-6.

[2] NAQVI S N Z，YFANTIDOU S，ZIM?NYI E.Time series databases and influxdb[J].Studienarbeit，Université Libre de Bruxelles，2017，12.

[3] DIVYA M S，GOYAL S K.ElasticSearch：an advanced and quick search technique to handle voluminous data[J].Compusoft，2013，2(6)：171.

[4] WU X，LONG X，WANG L.Optimizing event polling for network-intensive applications：a case study on redis[C]//2013 International Conference on Parallel and Distributed Systems.IEEE，2013：687-692.

[5] TANG W，LU Y，XIAO N，et al.Accelerating redis with RDMA over InfiniBand[C]//International Conference on Data Mining and Big Data.Springer，Cham，2017：472-483.

[6] MITCHELL C，GENG Y，LI J.Using one-sided RDMA reads to build a fast，CPU-effcient key-value store[C]//2013 USENIX Annual Technical Conference，2013：103-114.

[7] KALIA A，KAMINSKY M，ANDERSEN D G.Using RDMA efficiently for key-value services[C]//Proceedings of the 2014 ACM Conference on SIGCOMM，2014：295-306.

[8] WANG Y，MENG X，ZHANG L，et al.C-hint：an effective and reliable cache management for rdma-accelerated key-value stores[C]//Proceedings of the ACM Symposium on Cloud Computing，2014：1-13.

[9] LIU Q，YUAN H.A high performance memory key-value database based on Redis[J].J. Comput.，2019，14(3)：170-183.

[10] ZHANG J，JIA Y.Redis rehash optimization based on machine learning[C]//Journal of Physics：Conference Series.IOP Publishing，2020，1453(1)：012048.

[11] BLANKSTEIN A，SEN S，FREEDMAN M J.Hyperbolic caching：flexible caching for web applications[C]//2017 USENIX Annual Technical Conference，2017：499-511.

[12] PAN C，LUO Y，WANG X，et al.pRedis：Penalty and locality aware memory allocation in Redis[C]//Proceedings of the

(下转第 58 页)

(上接第 45 页)

2004(5)：110-117.

[23] SHCHEPETKIN A F，MCWILLIAMS J C.The regional oceanic modeling system(ROMS)：a split-explicit，free-surface，topography-following-coordinate oceanic model[J].Ocean Modelling，2005，9(4)：347-404.

[24] 廖嘉文，陈璟锟，颜辉，等.有限体积近岸海洋模式 FVCOM 的并行 I/O 优化[C]//CCF HPC China，2021：13-20.

[25] QI J，CHEN C，BEARDSLEY R C，et al.An unstructured-grid finite-volume surface wave model(FVCOM-SWAVE)：implementation，validations and applications[J].Ocean Modelling，2009，28(1)：153-166.

(收稿日期：2021-12-04)

作者简介：

杨斌(1992-)，男，博士研究生，工程师，主要研究方向：文件系统、智能存储。

王敬宇(1970-)，男，硕士，高级工程师，主要研究方向：高性能计算软件调试、大规模并行调试。

刘卫国(1975-)，通信作者，男，博士，教授，主要研究方向：高性能计算、大数据处理与分析，E-mail：weiguo.liu@sdu.edu.cn。

based stability comparison between VSGs and traditional grid-connected inverters[J].IEEE Transactions on Power Electronics,2019,34:46-52.
[3] DENG X,LI H,YU W,et al.Frequency observations and statistic analysis of worldwide main power grids using FNET/GridEye[C]//2019 IEEE Power & Energy Society General Meeting,2019:1-5.
[4] 张鹏,毕天姝,贺静波.基于模态电流注入方法抑制次同步谐振的阻尼控制策略[J].中国电机工程学报,2015,35(23):6011-6017.
[5] DENG X,BIAN D,SHI D,et al.Impact of low data quality on disturbance triangulation application using high-density PMU measurements[J].IEEE Access,2019,7:105054-105061.
[6] DENG X,BIAN D,SHI D,et al.Line outage detection and localization via synchrophasor measurement[J].IEEE PES Innovative Smart Grid Technologies(ISGT),2019:3373-3378.
[7] 谢小荣,王路平,贺静波,等.电力系统次同步谐振/振荡的形态分析[J].电网技术,2017,41(4):1043-1049.
[8] ZHANG S,JIANG S,LU X,et al.Resonance issues and damping techniques for grid-connected inverters with long transmission cable[J].IEEE Trans. Power Electron.,2014,29:110-120.
[9] JIAN S.Small-signal methods for AC distributed power systems-a review[J].IEEE Transactions on Power Electronics,2009,24:2545-2554.
[10] LIAO Y,LIU Z,ZHANG G,et al.Vehicle-grid system modeling and stability analysis with forbidden region-based criterion[J].IEEE Transactions on Power Electronics,2017,32:3499-3512.
[11] CESPEDES M,SUN J.Impedance modeling and analysis of grid-connected voltage-source converters[J].IEEE Transactions on Power Electronics,2014,29:1254-1261.
[12] 王赟程,陈新,陈杰,等.基于谐波线性化的三相LCL型并网逆变器正负序阻抗建模分析[J].中国电机工程学报,2016,36(21):5890-5898,6033.
[13] JIAN S.Impedance-based stability criterion for grid-connected inverters[J].IEEE Transactions on Power Electronics,2011,26:3075-3078.
[14] JAKSIC M,ZHIYU S,CVETKOVIC I,et al.Wide-band-width identification of small-signal dq impedances of AC power systems via single-phase series voltage injection[C]//Proceedings of 2015 17th European Conference on Power Electronics and Applications,2015:1-10.
[15] ROINILA T,VILKKO M,SUN J.Online grid impedance measurement using discrete-interval binary sequence injection[J].IEEE Journal of Emerging and Selected Topics in Power Electronics,2014,2:985-993.
[16] SUN Z,LIU M,LI L,et al.Research on the AC and DC hybrid power system simulation based on RTDS[C]//2020 IEEE 4th Conference on Energy Internet and Energy System Integration(EI2),2020:2324-2329.
[17] JALILI-MARANDI V,ROBERT E,LAPOINTE V,et al.A real-time transient stability simulation tool for large-scale power systems[C]//Power and Energy Society General Meeting,2012:1-7.
[18] DENG X,JIANG Z,SUNDARESH L,et al.A time-domain electromechanical co-simulation framework for power system transient analysis with retainment of user defined models[J].International Journal of Electrical Power & Energy Systems,2021,125:106506.
[19] DENG X,BIAN D,WANG W,et al.Deep learning model to detect various synchrophasor data anomalies[J].IET Generation,Transmission & Distribution,2020,14:5739-5745.
[20] SONG X,CAI H,JIANG T,et al.Research on performance of real-time simulation based on inverter-dominated power grid[J].IEEE Access,2021,9:1137-1153.
[21] LAUSS G,STRUNZ K.Multirate partitioning interface for enhanced stability of power hardware-in-the-loop real-time simulation[J].IEEE Transactions on Industrial Electronics,2019,66:595-605.

(收稿日期:2021-12-08)

作者简介:
曹斌(1981-),女,硕士,高级工程师,主要研究方向:电网电磁暂态、新能源电力系统。
原帅(1990-),男,硕士,工程师,主要研究方向:电力系统、新能源并网特性。
辛东昊(1983-),男,硕士,工程师,主要研究方向:新能源并网特性测试分析。

扫码下载电子文档

(上接第52页)

ACM Symposium on Cloud Computing,2019:193-205.
[13] CHEN S,TANG X,WANG H,et al.Towards scalable and reliable in-memory storage system:a case study with Redis[C]//2016 IEEE Trustcom/BigDataSE/ISPA.IEEE,2016:1660-1667.
[14] 杨斌,刘世超,邵明山,等.Beacon+:面向E级超级计算机的轻量级端到端I/O性能监控与分析诊断系统[C]//CCF HPC China,2021:1-12.

(收稿日期:2021-12-06)

作者简介:
刘世超(1995-),男,硕士研究生,工程师,主要研究方向:系统监控与分析、Infiniband网络。
杨斌(1992-),男,博士研究生,工程师,主要研究方向:文件系统、智能存储。
刘卫国(1975-),通信作者,男,博士,教授,主要研究方向:高性能计算、大数据处理与分析,E-mail:weiguo.liu@sdu.edu.cn。

扫码下载电子文档

版权声明