关于redis性能问题分析和优化

一、如何查看Redis性能

info命令输出的数据可以分为10个分类，分别是：

server,clients,memory,persistence,stats,replication,cpu,commandstats,cluster,keyspace

为了快速定位并解决性能问题，这里选择5个关键性的数据指标，它包含了大多数人在使用Redis上会经常碰到的性能问题

二、内存

上图中used_memory 字段数据表示的是：由Redis分配器分配的内存总量，以字节（byte）为单位。 其中used_memory_human和used_memory是一样的，以G为单位显示

```
info memory
# Memory
used_memory:8589645288
used_memory_human:8.00G
used_memory_rss:9439997952
used_memory_peak:9082282776
used_memory_peak_human:8.46G
used_memory_lua:35840
mem_fragmentation_ratio:1.10
mem_allocator:jemalloc-3.6.0
```

used_memory是Redis使用的内存总量，包含了实际缓存占用的内存和Redis自身运行所占用的内存(如元数据、lua)，是由Redis使用内存分配器分配的内存，所以这个数据不包括内存碎片浪费掉的内存，其他字段代表的含义，都以字节为单位：

- used_memory_rss：从操作系统上显示已经分配的内存总量。
- mem_fragmentation_ratio： 内存碎片率。
- used_memory_lua： Lua脚本引擎所使用的内存大小。
- mem_allocator： 在编译时指定的Redis使用的内存分配器，可以是libc、jemalloc、tcmalloc。

1、因内存交换引起的性能问题

内存使用率是Redis服务最关键的一部分。如果Redis实例的内存使用率超过可用最大内存(used_memory > 可用最大内存)，那么操作系统开始进行内存与swap空间交换，把内存中旧的或不再使用的内容写入硬盘上（硬盘上的这块空间叫Swap分区），以便留出新的物理内存给新页或活动页(page)使用。

如果Redis进程上发生内存交换，那么Redis和依赖Redis上数据的应用会受到严重的性能影响。 通过查看used_memory指标可知道Redis正在使用的内存情况，如果used_memory>可用最大内存，那就说明Redis实例正在进行内存交换或者已经内存交换完毕。

2、跟踪内存使用率

若是在使用Redis期间没有开启rdb快照或aof持久化策略，那么缓存数据在Redis崩溃时就有丢失的危险。因为当Redis内存使用率超过可用内存的95%时，部分数据开始在内存与swap空间来回交换，这时就可能有丢失数据的危险。

当开启并触发快照功能时，Redis会fork一个子进程把当前内存中的数据完全复制一份写入到硬盘上。因此若是当前使用内存超过可用内存的45%时触发快照功能，那么此时进行的内存交换会变的非常危险(可能会丢失数据)。 倘若在这个时候实例上有大量频繁的更新操作，问题会变得更加严重。

通过减少Redis的内存占用率，来避免这样的问题，或者使用下面的技巧来避免内存交换发生：

- 假如缓存数据小于4GB，就使用32位的Redis实例。因为32位实例上的指针大小只有64位的一半，它的内存空间占用空间会更少些。 这有一个坏处就是，假设物理内存超过4GB，那么32位实例能使用的内存仍然会被限制在4GB以下。 要是实例同时也共享给其他一些应用使用的话，那可能需要更高效的64位Redis实例，这种情况下切换到32位是不可取的。 不管使用哪种方式，Redis的dump文件在32位和64位之间是互相兼容的， 因此倘若有减少占用内存空间的需求，可以尝试先使用32位，后面再切换到64位上。
- 尽可能的使用Hash数据结构。因为Redis在储存小于100个字段的Hash结构上，其存储效率是非常高的。所以在不需要集合(set)操作或list的push/pop操作的时候，尽可能的使用Hash结构。比如，在一个web应用程序中，需要存储一个对象表示用户信息，使用单个key表示一个用户，其每个属性存储在Hash的字段里，这样要比给每个属性单独设置一个key-value要高效的多。 通常情况下倘若有数据使用string结构，用多个key存储时，那么应该转换成单key多字段的Hash结构。 如上述例子中介绍的Hash结构应包含，单个对象的属性或者单个用户各种各样的资料。Hash结构的操作命令是HSET(key, fields, value)和HGET(key, field)，使用它可以存储或从Hash中取出指定的字段。
- 设置key的过期时间。一个减少内存使用率的简单方法就是，每当存储对象时确保设置key的过期时间。倘若key在明确的时间周期内使用或者旧key不大可能被使用时，就可以用Redis过期时间命令(expire,expireat, pexpire, pexpireat)去设置过期时间，这样Redis会在key过期时自动删除key。 假如你知道每秒钟有多少个新key-value被创建，那可以调整key的存活时间，并指定阀值去限制Redis使用的最大内存。
- 回收key。在Redis配置文件中(一般叫Redis.conf)，通过设置“maxmemory”属性的值可以限制Redis最大使用的内存，修改后重启实例生效。 也可以使用客户端命令config set maxmemory 去修改值，这个命令是立即生效的，但会在重启后会失效，需要使用config rewrite命令去刷新配置文件。 若是启用了Redis快照功能，应该设置“maxmemory”值为系统可使用内存的45%，因为快照时需要一倍的内存来复制整个数据集，也就是说如果当前已使用45%，在快照期间会变成95%(45%+45%+5%)，其中5%是预留给其他的开销。 如果没开启快照功能，maxmemory最高能设置为系统可用内存的95%。

当内存使用达到设置的最大阀值时，需要选择一种key的回收策略，可在Redis.conf配置文件中修改“maxmemory-policy”属性值。 若是Redis数据集中的key都设置了过期时间，那么“volatile-ttl”策略是比较好的选择。但如果key在达到最大内存限制时没能够迅速过期，或者根本没有设置过期时间。那么设置为“allkeys-lru”值比较合适，它允许Redis从整个数据集中挑选最近最少使用的key进行删除(LRU淘汰算法)。Redis还提供了一些其他淘汰策略，如下：

- volatile-lru：使用LRU算法从已设置过期时间的数据集合中淘汰数据。
- volatile-ttl：从已设置过期时间的数据集合中挑选即将过期的数据淘汰。
- volatile-random：从已设置过期时间的数据集合中随机挑选数据淘汰。
- allkeys-lru：使用LRU算法从所有数据集合中淘汰数据。
- allkeys-random：从数据集合中任意选择数据淘汰
- no-enviction：禁止淘汰数据。

通过设置maxmemory为系统可用内存的45%或95%(取决于持久化策略)和设置“maxmemory-policy”为“volatile-ttl”或“allkeys-lru”(取决于过期设置)，可以比较准确的限制Redis最大内存使用率，在绝大多数场景下使用这2种方式可确保Redis不会进行内存交换。倘若你担心由于限制了内存使用率导致丢失数据的话，可以设置noneviction值禁止淘汰数据。

三、命令处理数

在info信息里的total_commands_processed字段显示了Redis服务处理命令的总数，其命令来自一个或多个Redis客户端

```
info stats
# Stats
total_connections_received:843391006
total_commands_processed:3946780282
instantaneous_ops_per_sec:1447
total_net_input_bytes:5060670300797
total_net_output_bytes:13788457111609
instantaneous_input_kbps:1399.63
instantaneous_output_kbps:2863.71
rejected_connections:0
sync_full:2
sync_partial_ok:1
sync_partial_err:0
expired_keys:231497375
evicted_keys:0
keyspace_hits:613100363
keyspace_misses:252710911
pubsub_channels:0
pubsub_patterns:0
latest_fork_usec:60179
```

分析命令处理总数，诊断响应延迟

在Redis实例中，跟踪命令处理总数是解决响应延迟问题最关键的部分，因为Redis是个单线程模型，客户端过来的命令是按照顺序执行的。比较常见的延迟是带宽，通过千兆网卡的延迟大约有200μs。倘若明显看到命令的响应时间变慢，延迟高于200μs，那可能是Redis命令队列里等待处理的命令数量比较多。 如上所述，延迟时间增加导致响应时间变慢可能是由于一个或多个慢命令引起的，这时可以看到每秒命令处理数在明显下降，甚至于后面的命令完全被阻塞，导致Redis性能降低。要分析解决这个性能问题，需要跟踪命令处理数的数量和延迟时间。

比如可以写个脚本，定期记录total_commands_processed的值。当客户端明显发现响应时间过慢时，可以通过记录的total_commands_processed历史数据值来判断命理处理总数是上升趋势还是下降趋势，以便排查问题。

使用命令处理总数解决延迟时间增加

通过与记录的历史数据比较得知，命令处理总数确实是处于上升或下降状态，那么可能是有2个原因引起的:

- 命令队列里的命令数量过多，后面命令一直在等待中
- 几个慢命令阻塞Redis

下面有三个办法可以解决，因上面2条原因引起的响应延迟问题。

1. 使用多参数命令：若是客户端在很短的时间内发送大量的命令过来，会发现响应时间明显变慢，这由于后面命令一直在等待队列中前面大量命令执行完毕。有个方法可以改善延迟问题，就是通过单命令多参数的形式取代多命令单参数的形式。举例来说，循环使用LSET命令去添加1000个元素到list结构中，是性能比较差的一种方式，更好的做法是在客户端创建一个1000元素的列表，用单个命令LPUSH或RPUSH，通过多参数构造形式一次性把1000个元素发送的Redis

服务上。下面是Redis的一些操作命令，有单个参数命令和支持多个参数的命令，通过这些命令可尽量减少使用多命令的次数。

```
set  -> mset
get  -> mget
lset -> lpush, rpush
lindex -> lrange
hset -> hmset
hget -> hmget
```

2. 管道命令：另一个减少多命令的方法是使用管道(pipeline)，把几个命令合并一起执行，从而减少因网络开销引起的延迟问题。因为10个命令单独发送到服务端会引起10次网络延迟开销，使用管道会一次性把执行结果返回，仅需要一次网络延迟开销。Redis本身支持管道命令，大多数客户端也支持，倘若当前实例延迟很明显，那么使用管道去降低延迟是非常有效的。

3. 避免操作大集合的慢命令：如果命令处理频率过低导致延迟时间增加，这可能是因为使用了高时间复杂度的命令操作导致，这意味着每个命令从集合中获取数据的时间增大。 所以减少使用高时间复杂的命令，能显著的提高的Redis的性能。

四、延迟时间

Redis的延迟数据是无法从info信息中获取的。可以用 Redis-cli工具加 --latency参数运行，如:

```
redis-cli --latency -h 127.0.0.1 -p 6379
```

由于当前服务器不同的运行情况，延迟时间可能有所误差，通常1G网卡的延迟时间是200μs，Redis的响应延迟时间以毫秒为单位

```
[root@localhost ~]# redis-cli --latency -h 127.0.0.1 -p 6379
min: 0, max: 1, avg: 0.07 (12596 samples)
```

跟踪Redis延迟性能

Redis之所以这么流行的主要原因之一就是低延迟特性带来的高性能，所以说解决延迟问题是提高Redis性能最直接的办法。拿1G带宽来说，若是延迟时间远高于200μs，那明显是出现了性能问题。虽然在服务器上会有一些慢的IO操作，但Redis是单核接受所有客户端的请求，所有请求是按良好的顺序排队执行。因此若是一个客户端发过来的命令是个慢操作，那么其他所有请求必须等待它完成后才能继续执行。

使用延迟命令提高性能

一旦确定延迟时间是个性能问题后，这里有几个办法可以用来分析解决性能问题。

1. 使用slowlog查出引发延迟的慢命令：Redis中的slowlog命令可以让我们快速定位到那些超出指定执行时间的慢命令，默认情况下命令若是执行时间超过10ms就会被记录到日志。slowlog只会记录其命令执行的时间，不包含io往返操作，也不记录单由网络延迟引起的响应慢。通常1gb带宽的网络延迟，预期在200μs左右，倘若一个命令仅执行时间就超过10ms，那比网络延迟慢了近50倍。 想要查看所有执行时间比较慢的命令，可以通过使用Redis-cli工具，输入slowlog get命令查看，返回结果的第三个字段以微妙位单位显示命令的执行时间。假如只需要查看最后10个慢命令，输入slowlog get 10即可

```
slowlog get
 1) 1) (integer) 12849
    2) (integer) 1495630160
    3) (integer) 61916
    4) 1) "KEYS"
       2) "20170524less*"
 2) 1) (integer) 12848
    2) (integer) 1495629901
    3) (integer) 59368
```

```
    3) (integer) 59506
    4) 1) "KEYS"
       2) "20170524more*"
 3) 1) (integer) 12847
    2) (integer) 1495629504
    3) (integer) 59522
    4) 1) "KEYS"
       2) "sou_dzmore_16_*"
 4) 1) (integer) 12846
    2) (integer) 1495629504
    3) (integer) 57941
    4) 1) "KEYS"
       2) "sou_dz_16_*"
 5) 1) (integer) 12845
    2) (integer) 1495629504
    3) (integer) 15053
    4) 1) "KEYS"
       2) "list_dingzhis_16_*"
 6) 1) (integer) 12844
    2) (integer) 1495629504
    3) (integer) 24391
    4) 1) "KEYS"
       2) "cache_kwnew_*"
 7) 1) (integer) 12843
    2) (integer) 1495629469
    3) (integer) 57001
    4) 1) "KEYS"
       2) "sou_dzmore_15_*"
 8) 1) (integer) 12842
    2) (integer) 1495629469
    3) (integer) 61131
    4) 1) "KEYS"
       2) "sou_dz_15_*"
 9) 1) (integer) 12841
    2) (integer) 1495629469
    3) (integer) 10035
    4) 1) "KEYS"
       2) "ztlistnew_dingzhi_15_*"
10) 1) (integer) 12840
    2) (integer) 1495629469
    3) (integer) 17974
    4) 1) "KEYS"
       2) "list_dingzhis_15_*"
```

图中字段分别意思是：

- 1、日志的唯一标识符
- 2、被记录命令的执行时间点，以 UNIX 时间戳格式表示
- 3、查询执行时间，以微秒为单位
- 4、执行的命令，以数组的形式排列。完整命令是config get *

倘若你想自定义慢命令的标准，可以调整触发日志记录慢命令的阀值。若是很少或没有命令超过10ms，想降低记录的阀值，比如5毫秒，可在Redis-cli工具中输入下面的命令配置：

```
config set slowlog-log-slower-than 5000
```

也可以在Redis.config配置文件中设置，以微妙位单位。

2.监控客户端的连接：因为Redis是单线程模型(只能使用单核)，来处理所有客户端的请求， 但由于客户端连接数的增长，处理请求的线程资源开始降低分配给单个客户端连接的处理时间，这时每个客户端需要花费更多的时间去等待Redis共享服务的响应。这种情况下监控客户端连接数是非常重要的，因为客户端创建连接数的数量可能超出预期的数量，也可能是客户端端没有有效的释放连接。在Redis-cli工具中输入info clients可以查看到当前实例的所有客户端连接信息。如下图，第一个字段(connected_clients)显示当前实例客户端连接的总数：

```
info clients
```

```
info clients
# Clients
connected_clients:21
client_longest_output_list:0
client_biggest_input_buf:13856
blocked_clients:0
```

Redis默认允许客户端连接的最大数量是10000。若是看到连接数超过5000以上，那可能会影响Redis的性能。倘若一些或大部分客户端发送大量的命令过来，这个数字会低的多。

3.限制客户端连接数：自Redis2.6以后，允许使用者在配置文件(Redis.conf)maxclients属性上修改客户端连接的最大数，也可以通过在Redis-cli工具上输入config set maxclients 去设置最大连接数。根据连接数负载的情况，这个数字应该设置为预期连接数峰值的110到150之间，若是连接数超出这个数字后，Redis会拒绝并立刻关闭新来的连接。通过设置最大连接数来限制非预期数量的连接数增长，是非常重要的。另外，新连接尝试失败会返回一个错误消息，这可以让客户端知道，Redis此时有非预期数量的连接数，以便执行对应的处理措施。 上述二种做法对控制连接数的数量和持续保持Redis的性能最优是非常重要的，

4.加强内存管理：较少的内存会引起Redis延迟时间增加。如果Redis占用内存超出系统可用内存，操作系统会把Redis进程的一部分数据，从物理内存交换到硬盘上，内存交换会明显的增加延迟时间。关于怎么监控和减少内存使用，可查看used_memory介绍章节。

5. 性能数据指标：分析解决Redis性能问题，通常需要把延迟时间的数据变化与其他性能指标的变化相关联起来。命令处理总数下降的发生可能是由慢命令阻塞了整个系统，但如果命令处理总数的增加，同时内存使用率也增加，那么就可能是由于内存交换引起的性能问题。对于这种性能指标相关联的分析，需要从历史数据上来观察到数据指标的重要变化，此外还可以观察到单个性能指标相关联的所有其他性能指标信息。这些数据可以在Redis上收集，周期性的调用内容为Redis info的脚本，然后分析输出的信息，记录到日志文件中。当延迟发生变化时，用日志文件配合其他数据指标，把数据串联起来排查定位问题。

五、内存碎片率

info信息中的mem_fragmentation_ratio给出了内存碎片率的数据指标，它是由操系统分配的内存除以Redis分配的内存得出：

```
mem_fragmentation_ratio = used_memory_rss / used_memory
```

used_memory和used_memory_rss都包含的内存分配有：

- 用户定义的数据：内存被用来存储key-value值。
- 内部开销： 存储内部Redis信息用来表示不同的数据类型。

used_memory_rss的rss是Resident Set Size的缩写，表示该进程所占物理内存的大小，是操作系统分配给Redis实例的内存大小。除了用户定义的数据和内部开销以外，used_memory_rss指标还包含了内存碎片的开销，内存碎片是由操作系统低效的分配/回收物理内存导致的。

操作系统负责分配物理内存给各个应用进程，Redis使用的内存与物理内存的映射是由操作系统上虚拟内存管理分配器完成的。

举个例子来说，Redis需要分配连续内存块来存储1G的数据集，这样的话更有利，但可能物理内存上没有超过1G的连续内存块，那操作系统就不得不使用多个不连续的小内存块来分配并存储这1G数据，也就导致内存碎片的产生。

内存分配器另一个复杂的层面是，它经常会预先分配一些内存块给引用，这样做会使加快应用程序的运行。

理解资源性能

跟踪内存碎片率对理解Redis实例的资源性能是非常重要的。内存碎片率稍大于1是合理的，这个值表示内存碎片率比较低，也说明redis没有发生内存交换。但如果内存碎片率超过1.5，那就说明Redis消耗了实际需要物理内存的150%，其中50%是内存碎片率。若是内存碎片率低于1的话，说明Redis内存分配超出了物理内存，操作系统正在进行内存交换。内存交换会引起非常明显的响应延迟，可查看used_memory介绍章节。

```
info memory
# Memory
used_memory:21189222536
used_memory_human:19.73G
used_memory_rss:21901688832
used_memory_peak:27350156888
used_memory_peak_human:25.47G
used_memory_lua:35840
mem_fragmentation_ratio:1.03
mem_allocator:jemalloc-3.6.0
```

用内存碎片率预测性能问题

倘若内存碎片率超过了1.5，那可能是操作系统或Redis实例中内存管理变差的表现。下面有3种方法解决内存管理变差的问题，并提高Redis性能：

1. 重启Redis服务器：如果内存碎片率超过1.5，重启Redis服务器可以让额外产生的内存碎片失效并重新作为新内存来使用，使操作系统恢复高效的内存管理。额外碎片的产生是由于Redis释放了内存块，但内存分配器并没有返回内存给操作系统，这个内存分配器是在编译时指定的，可以是libc、jemalloc或者tcmalloc。 通过比较used_memory_peak, used_memory_rss和used_memory_metrics的数据指标值可以检查额外内存碎片的占用。从名字上可以看出，used_memory_peak是过去Redis内存使用的峰值，而不是当前使用内存的值。如果used_memory_peak和used_memory_rss的值大致上相等，而且二者明显超过了used_memory值，这说明额外的内存碎片正在产生。 在Redis-cli工具上输入info memory可以查看上面三个指标的信息：

在重启服务器之前，需要在Redis-cli工具上输入shutdown save命令，意思是强制让Redis数据库执行保存操作并关闭Redis服务，这样做能保证在执行Redis关闭时不丢失任何数据。 在重启后，Redis会从硬盘上加载持久化的文件，以确保数据集持续可用。

2.限制内存交换： 如果内存碎片率低于1，Redis实例可能会把部分数据交换到硬盘上。内存交换会严重影响Redis的性能，所以应该增加可用物理内存或减少实Redis内存占用。 可查看used_memory章节的优化建议。

3.修改内存分配器：Redis支持glibc's malloc、jemalloc11、tcmalloc几种不同的内存分配器，每个分配器在内存分配和碎片上都有不同的实现。不建议普通管理员修改Redis默认内存分配器，因为这需要完全理解这几种内存分配器的差异，也要重新编译Redis。这个方法更多的是让其了解Redis内存分配器所做的工作，当然也是改善内存碎片问题的一种办法。

六、回收key

info信息中的evicted_keys字段显示的是，因为maxmemory限制导致key被回收删除的数量。回收key的情况只会发生在设置maxmemory值后，不设置会发生内存交换。 当Redis由于内存压力需要回收一个key时，Redis首先考虑的不是回收最旧的数据，而是在最近最少使用的key或即将过期的key中随机选择一个key，从数据集中删除。

这可以在配置文件中设置maxmemory-policy值为"volatile-lru"或"volatile-ttl"，来确定Redis是使用lru策略还是过期时间策略。 倘若所有的key都有明确的过期时间，那过期时间回收策略是比较合适的。若是没有设置key的过期时间或者说没有足够的过期key，那设置lru策略是比较合理的，这可以回收key而不用考虑其过期状态。

```
# Stats
total_connections_received:843708918
total_commands_processed:3947987793
instantaneous_ops_per_sec:1360
```

```
total_net_input_bytes:5061895225788
total_net_output_bytes:13791028024582
instantaneous_input_kbps:1247.52

instantaneous_output_kbps:2756.92
rejected_connections:0
sync_full:2
sync_partial_ok:1
sync_partial_err:0
expired_keys:231544806
evicted_keys:0
keyspace_hits:613324172
keyspace_misses:252815503
pubsub_channels:0
pubsub_patterns:0
latest_fork_usec:60179
```

根据key回收定位性能问题

跟踪key回收是非常重要的，因为通过回收key，可以保证合理分配Redis有限的内存资源。如果evicted_keys值经常超过0，那应该会看到客户端命令响应延迟时间增加，因为Redis不但要处理客户端过来的命令请求，还要频繁的回收满足条件的key。

需要注意的是，回收key对性能的影响远没有内存交换严重，若是在强制内存交换和设置回收策略做一个选择的话，选择设置回收策略是比较合理的，因为把内存数据交换到硬盘上对性能影响非常大(见前面章节)。

减少回收key以提升性能

减少回收key的数量是提升Redis性能的直接办法，下面有2种方法可以减少回收key的数量：

1.增加内存限制：倘若开启快照功能，maxmemory需要设置成物理内存的45%，这几乎不会有引发内存交换的危险。若是没有开启快照功能，设置系统可用内存的95%是比较合理的，具体参考前面的快照和maxmemory限制章节。如果maxmemory的设置是低于45%或95%(视持久化策略)，通过增加maxmemory的值能让Redis在内存中存储更多的key，这能显著减少回收key的数量。 若是maxmemory已经设置为推荐的阀值后，增加maxmemory限制不但无法提升性能，反而会引发内存交换，导致延迟增加、性能降低。 maxmemory的值可以在Redis-cli工具上输入config set maxmemory命令来设置。

需要注意的是，这个设置是立即生效的，但重启后丢失，需要永久化保存的话，再输入config rewrite命令会把内存中的新配置刷新到配置文件中。

2.对实例进行分片：分片是把数据分割成合适大小，分别存放在不同的Redis实例上，每一个实例都包含整个数据集的一部分。通过分片可以把很多服务器联合起来存储数据，相当于增加总的物理内存，使其在没有内存交换和回收key的策略下也能存储更多的key。假如有一个非常大的数据集，maxmemory已经设置，实际内存使用也已经超过了推荐设置的阀值，那通过数据分片能明显减少key的回收，从而提高Redis的性能。 分片的实现有很多种方法，下面是Redis实现分片的几种常见方式：

- a. Hash分片：一个比较简单的方法实现，通过Hash函数计算出key的Hash值，然后值所在范围对应特定的Redis实例。
- b. 代理分片：客户端把请求发送到代理上，代理通过分片配置表选择对应的Redis实例。 如Twitter的Twemproxy，豌豆荚的codis。
- c. 一致性Hash分片
- d. 虚拟桶分片