云风的 BLOG
思绪来得快去得也快，偶尔会在这里停留

« 谈谈陌陌争霸在数据库方面踩过的坑(芒果篇) | 返回首页 | linode 广告时间 »

谈谈陌陌争霸在数据库方面踩过的坑(Redis 篇)

注：陌陌争霸的数据库部分我没有参与具体设计，只是参与了一些讨论和提出一些意见。在出现问题的时候，也都是由肥龙、晓靖、Aply 同学判断研究解决的。所以我对 Redis 的判断大多也从他们的讨论中听来，加上自己的一些猜测，并没有去仔细阅读 Redis 文档和阅读 Redis 代码。虽然我们最终都解决了问题，但本文中说描述的技术细节还是很有可能与事实相悖，请阅读的同学自行甄别。

在陌陌争霸之前，我们并没有大规模使用过 Redis 。只是直觉上感觉 Redis 很适合我们的架构：我们这个游戏不依赖数据库帮我们处理任何数据，总的数据量虽然较大，但增长速度有限。由于单台服务机处理能力有限，而游戏又不能分服，玩家在任何时间地点登陆，都只会看到一个世界。所以我们需要有一个数据中心独立于游戏系统。而这个数据中心只负责数据中转和数据落地就可以了。Redis 看起来就是最佳选择，游戏系统对它只有按玩家 ID 索引出玩家的数据这一个需求。

我们将数据中心分为 32 个库，按玩家 ID 分开。不同的玩家之间数据是完全独立的。在设计时，我坚决反对了从一个单点访问数据中心的做法，坚持每个游戏服务器节点都要多每个数据仓库直接连接。因为在这里制造一个单点毫无必要。

根据我们事前对游戏数据量的估算，前期我们只需要把 32 个数据仓库部署到 4 台物理机上即可，每台机器上启动 8 个 Redis 进程。一开始我们使用 64G 内存的机器，后来增加到了 96G 内存。实测每个 Redis 服务会占到 4~5 G 内存，看起来是绰绰有余的。

由于我们仅仅是从文档上了解的 Redis 数据落地机制，不清楚会踏上什么坑，为了保险起见，还配备了 4 台物理机做为从机，对主机进行数据同步备份。

Redis 支持两种 BGSAVE 的策略，一种是快照方式，在发起落地指令时，fork 出一个进程把整个内存 dump 到硬盘上；另一种唤作 AOF 方式，把所有对数据库的写操作记录下来。我们的游戏不适合用 AOF 方式，因为我们的写入操作实在的太频繁了，且数据量巨大。

第一次事故出在 2 月 3 日，新年假期还没有过去。由于整个假期都相安无事，运维也相对懈怠。

中午的时候，有一台数据服务主机无法被游戏服务器访问到，影响了部分用户登陆。在线尝试修复连接无果，只好开始了长达 2 个小时的停机维护。

在维护期间，初步确定了问题。是由于上午一台从机的内存耗尽，导致了从机的数据库服务重启。在从机重新对主机连接，8 个 Redis 同时发送 SYNC 的冲击下，把主机击毁了。

这里存在两个问题，我们需要分别讨论：

问题一：从机的硬件配置和主机是相同的，为什么从机会先出现内存不足。

问题二：为何重新进行 SYNC 操作会导致主机过载。

问题一当时我们没有深究，因为我们没有估算准确过年期间用户增长的速度，而正确部署数据库。数据库的内存需求增加到了一个临界点，所以感觉内存不足的意外发生在主机还是从机都是很有可能的。从机先挂掉或许只是碰巧而已（现在反思恐怕不是这样，冷备脚本很可能是罪魁祸首）。早期我们是定时轮流 BGSAVE 的，当数据量增长时，应该适当调大 BGSAVE 间隔，避免同一台物理机上的 redis 服务同时做 BGSAVE ，而导致 fork 多个进程需要消耗太多内存。由于过年期间都回家过年去了，这件事情也被忽略了。

问题二是因为我们对主从同步的机制了解不足：

仔细想想，如果你来实现同步会怎么做？由于达到同步状态需要一定的时间。同步最好不要干涉正常服务，那么保证同步的一致性用锁肯定是不好的。所以 Redis 在同步时也触发了 fork 来保证从机连上来发出 SYNC 后，能够顺利到达一个正确的同步点。当我们的从机重启后，8 个 slave redis 同时开启同步，等于瞬间在主机上 fork 出 8 个 redis 进程，这使得主机 redis 进程进入交换分区的概率大大提高了。

在这次事故后，我们取消了 slave 机。因为这使系统部署更复杂了，增加了许多不稳定因素，且未必提高了数据安全性。同时，我们改进了 bgsave 的机制，不再用定时器触发，而是由一个脚本去保证同一台物理机上的多个 redis 的

bgsave 可以轮流进行。另外，以前在从机上做冷备的机制也移到了主机上。好在我们可以用脚本控制冷备的时间，以及错开 BGSAVE 的 IO 高峰期。

第二次事故最出现在最近（ 2 月 27 日）。

我们已经多次调整了 Redis 数据库的部署，保证数据服务器有足够的内存。但还是出了次事故。事故最终的发生还是因为内存不足而导致某个 Redis 进程使用了交换分区而处理能力大大下降。在大量数据拥入的情况下，发生了雪崩效应：晓靖在原来控制 BGSAVE 的脚本中加了行保底规则，如果 30 分钟没有收到 BGSAVE 指令，就强制执行一次保障数据最终可以落地（对这条规则我个人是有异议的）。结果数据服务器在对外部失去响应之后的半小时，多个 redis 服务同时进入 BGSAVE 状态，吃光了内存。

花了一天时间追查事故的元凶。我们发现是冷备机制惹的祸。我们会定期把 redis 数据库文件复制一份打包备份。而操作系统在拷贝文件时，似乎利用了大量的内存做文件 cache 而没有及时释放。这导致在一次 BGSAVE 发生的时候，系统内存使用量大大超过了我们原先预期的上限。

这次我们调整了操作系统的内核参数，关掉了 cache ，暂时解决了问题。

经过这次事故之后，我反思了数据落地策略。我觉得定期做 BGSAVE 似乎并不是好的方案。至少它是浪费的。因为每次 BGSAVE 都会把所有的数据存盘，而实际上，内存数据库中大量的数据是没有变更过的。一目前 10 到 20 分钟的保存周期，数据变更的只有这个时间段内上线的玩家以及他们攻击过的玩家（每 20 分钟大约发生 1 到 2 次攻击），这个数字远远少于全部玩家数量。

我希望可以只备份变更的数据，但又不希望用内建的 AOF 机制，因为 AOF 会不断追加同一份数据，导致硬盘空间太快增长。

我们也不希望给游戏服务和数据库服务之间增加一个中间层，这白白牺牲了读性能，而读性能是整个系统中至关重要的。仅仅对写指令做转发也是不可靠的。因为失去和读指令的时序，有可能使数据版本错乱。

如果在游戏服务器要写数据时同时向 Redis 和另一个数据落地服务同时各发一份数据怎样？首先，我们需要增加版本机制，保证能识别出不同位置收到的写操作的先后（我记得在狂刃中，就发生过数据版本错乱的 Bug ）；其次，这会使游戏服务器和数据服务器间的写带宽加倍。

最后我想了一个简单的方法：在数据服务器的物理机上启动一个监护服务。当游戏服务器向数据服务推送数据并确认成功后，再把这组数据的 ID 同时发送给这个监护服务。它再从 Redis 中把数据读回来，并保存在本地。

因为这个监护服务和 Redis 1 比 1 配置在同一台机器上，而硬盘写速度是大于网络带宽的，它一定不会过载。至于 Redis ，就成了一个纯粹的内存数据库，不再运行 BGSAVE 。

这个监护进程同时也做数据落地。对于数据落地，我选择的是 unqlite ，几行代码就可以做好它的 Lua 封装。它的数据库文件只有一个，更方便做冷备。当然 levelDB 也是个不错的选择，如果它是用 C 而不是 C++ 实现的话，我会考虑后者的。

和游戏服务器的对接，我在数据库机器上启动了一个独立的 skynet 进程，监听同步 ID 的请求。因为它只需要处理很简单几个 Redis 操作，我特地手写了 Redis 指令。最终这个服务 只有一个 lua 脚本 ，其实它是由三个 skynet 服务构成的，一个监听外部端口，一个处理连接上的 Redis 同步指令，一个单点写入数据到 unqlite 。为了使得数据恢复高效，我特地在保存玩家数据的时候，把恢复用的 Redis 指令拼好。这样一旦需要恢复，只用从 unqlite 中读出玩家数据，直接发送给 Redis 即可。

有了这个东西，就一并把 Redis 中的冷热数据解决了。长期不登陆的玩家，我们可以定期从 Redis 中清掉，万一这个玩家登陆回来，只需要让它帮忙恢复。

晓靖不喜欢我依赖 skynet 的实现。他一开始想用 python 实现一个同样的东西，后来他又对 Go 语言产生了兴趣，想借这个需求玩一下 Go 语言。所以到今天，我们还没有把这套新机制部署到生产环境。

云风 提交于 March 4, 2014 04:59 PM | 固定链接

COMMENTS

请问下，你们是纯用redis，还是在redis下又用了mysql呢？

Posted by: hanframe | (43) December 20, 2014 06:00 PM

1.可以考虑对v进行字符串压缩，可以大幅节省内存和机器。
2.一般是单台slave或master出问题，重启的话也是单台，最好不要重启所有的slave。
3.每个slave机器预留30-40%的内存。
4.对冷热业务的优先级和重要性进行切割，不要将所有业务用一个redis集群。

Posted by: bestshare | (42) November 26, 2014 12:22 PM

我的看法是:
1) 个人感觉Redis做主存是存疑的。除了物理内存本身的大小限制外，还有要考虑运维系统的实现。单纯地KV系统，基本很难支持比较复杂的业务查询，比如查询充值超过100并且登录次数的玩家。

2) 如果考虑用Redis做memcache, 实际上Redis是独立进程，读取数据仍然是cs的模式，在不实用Redis做持久化的前提先，没想出它的具体使用案例。简单实现与游戏直接关联的实体对象，直接作为缓存的载体是我的倾向。当然也可能是因为我们做的游戏数据量级不同的缘故。

Posted by: BearOcean | (41) September 9, 2014 12:37 PM

用户数量爆增，应该设置一个上限值。
数据库备份确实会吃很多IO资源。

Posted by: kk | (40) August 28, 2014 01:46 PM

@sfwtoms

"前期我们只需要把 32 个数据仓库部署到 4 台物理机上即可，每台机器上启动 8 个 Redis 进程。"

原文中写了，每个机器上有 8 个 redis 进程。

另外，也不可能浪费一半内存仅仅为了让 bgsave 可以正常工作。

Posted by: Cloud | (39) August 19, 2014 12:19 PM

首先： 1) 防止机子出现毁灭性影响，应该开启REDIS缓存机制，配置Maxmemory为你本机的一半 这是前提
2) 更换redis2.8的版本，如果是2.8以前的版本 从机挂掉 重启会出现sync 这样非常消耗资源 2.8版本直接调用Psync 复制是有偏移向量的
3) 我不太懂你说的那个bgsave 同时启动多个bgsave redis是拒绝其他的bgsave操作的 也就是在同一时刻是不会有多个fork进行 求告知 嘿嘿
4)从机的硬件配置和主机是相同的，为什么从机会先出现内存不足 这个问题我想死都没有想清楚 是不是主从跨机房了？然后导致了网络传输出问题 这样也会托跨主节点啊 求告知
5) 最重要的： 不要用redis的vm 不过现在的版本都没有这个功能 这个用了只有等死的分
6)嘿嘿 问题体完了 求告知真相

Posted by: sfwtoms | (38) August 18, 2014 09:01 PM

一般镜像和aof结合使用。
每隔一段时间拖一个镜像到本地，aof日志就可以删了，从这个时间点后开始记录。恢复的时候先恢复镜像再恢复aof。在做镜像前给aof里面插入个日期。方便分割。

Posted by: Kevin | (37) July 30, 2014 03:47 PM

我们在redis 方面的最佳实践，都集成到redis-mgr里面了:

https://github.com/idning/redis-mgr

redis+twemproxy+sentinel deploy/auto-failover/monitor/migrate/rolling-upgrade 一键式管理

Posted by: idning | (36) July 11, 2014 05:19 PM

"在数据服务器的物理机上启动一个监护服务。当游戏服务器向数据服务推送数据并确认成功后，再把这组数据的 ID 同时发送给这个监护服务。它再从 Redis 中把数据读回来，并保存在本地。" 这句话的意思是，当redis数据发生变化后，就立马把数据同步到本地？是否可以采用异步同步的方式？

请教大师，谢谢

Posted by: sunli | (35) July 4, 2014 06:42 PM

其实aof的持久化方式，可以指定参数刷新aof文件，这样就可以压缩aof文件到实际的大小了。
auto-aof-rewrite-percentage 100
auto-aof-rewrite-min-size 64mb

Posted by: eric | (34) April 23, 2014 10:43 AM

分32库，按id存放，那登录帐号和id的映射要存在哪个库？那有关联设备的时候的映射关系又要存在哪里？ 求指教

Posted by: zyanfeng | (33) April 15, 2014 09:07 PM

实在不错!

Posted by: 麦_Mike | (32) April 9, 2014 10:23 PM

感觉redis数据落地策略过于粗糙。针对数据更新的数量采用不同策略。1，如果更新数据量小，redis直接是缓存，leveldb或mysql做持久，更新数据先持久再缓存。2，更新数据量大，定期dump是个不错选择。但是不要让redis自己dump，因为redis不知道和其他实例错开时间，用脚本控制很简单，保证同一个时间只有一个做dump

Posted by: Anonymous | (31) March 25, 2014 04:55 PM

是所有状态数据都存到redis吗？系统运算过程中的一些中间数据不知道是怎么处理的？比如处理事件的上下文

Posted by: Tom | (30) March 22, 2014 04:45 PM

我们也做了个类似的游戏,所有模块都使用的go语言,数据落地是定时存回mysql,策略很简单,玩家的每一部分数据都有版本号,回存时只将更改过的数据拼装成事务交给mysql,目前运行了4个月,用户量也还不小,基本没有出现过问题,也基本不需要维护. 除了mysql drive,我们没有用任何形式的库.个人感觉服务器的话还是少用各种不了解的库,会更轻松.

Posted by: nd | (29) March 19, 2014 12:25 PM

我们的做法和云风差不多，落地策略有些差异。redis只做cache，redis里的key根据具体情况设置过期时间而已，防止过度膨胀，冷数据自动过期。玩家下线时，kv数据保存成功后，把id往redis list里插，另外用个脚本定期从list里取，每次取若干个，再用这些id把kv读出持久化回mysql，这样io的曲线也很平滑，呵呵。

Posted by: nana | (28) March 12, 2014 02:41 PM

1, 软件出问题是一件幸福的事情，如果是物理内存总挂、或者主板总是坏掉，那真是让人欲哭无泪。
2， 运维报警机制要完善。
3， cache使用的是lru机制，备份时，内核要把整个文件读到内核内存里，即使类似cp指令一次只读几K。关掉cache，如果没有太多反复的读写请求是可以的。
4，fork dump这个很吊。

Posted by: www | (27) March 11, 2014 11:08 AM

Redis 文档写得很好，读读很有收获。 overcommit_memory 一定要打开。

master 不落地完全没问题，用 slave 备份。而且 Redis 推荐给 slave 配置多个 slave 来解决相对昂贵的读操作。master + 8 slaves 确实是很不好的结构，应该是 master + slave + 7 slaves of slave. 我们生产还使用 sentinel，可以解决不落地 master 挂掉后的 failover。

Twitter 的 nutcracker 也是实现高可用的好方法。

还有，分布式的关键是区分缓存还是主存角色。可以 LRU 的是缓存。主存的话就只能 mod 了（或者 twemeproxy）。

Posted by: Tin | (26) March 7, 2014 05:33 PM

Redis 文档写得很好，读读很有收获。 overcommit_memory 一定要打开。

master 不落地完全没问题，用 slave 备份。而且 Redis 推荐给 slave 配置多个 slave 来解决相对昂贵的读操作。master + 8 slaves 确实是很不好的结构，应该是 master + slave + 7 slaves of slave. 我们生产还使用 sentinel，可以解决不落地 master 挂掉后的 failover。

Twitter 的 nutcracker 也是实现高可用的好方法。

还有，分布式的关键是区分缓存还是主存角色。可以 LRU 的是缓存。主存的话就只能 mod 了（或者 twemeproxy）。

Posted by: Tin | (25) March 7, 2014 05:33 PM

@郭建炯

我觉得你这种改redis源代码的思路相当靠谱！对外redis用法没有任何变化

Posted by: 顺娃 | (24) March 6, 2014 08:37 PM

之前一直专职从事终端软件开发，最近想接触下server端的东西，求云风兄指点些学习路径

Posted by: suning | (23) March 6, 2014 07:35 PM

@zengzizhao

虽然用 “Redis Keyspace Notifications” 这种方法，容易让人找到漏洞或者话柄，看似不靠谱，但其实是非常靠谱的。

本地 Redis 连接如果异常断掉，我怀疑一般就是系统挂掉了，那种情况下，任何一种方案都存在丢失数据的风险。

再说，用于落地的 Redis 本地连接断掉而Resdis 健在，这种情况下数据不还是没丢失嘛，全部重新落地一次即可，很容易找到安全的方案。

Posted by: 路人甲(ChinarenWei) | (22) March 6, 2014 08:22 AM

谢谢分享，我也打算应用redis, 这些‘故障’经验，对我很有帮助。

Posted by: kencoder | (21) March 5, 2014 11:57 PM

Background saving is failing with a fork() error under Linux even if I've a lot of free RAM!
Short answer: echo 1 > /proc/sys/vm/overcommit_memory :)
这个方法可行吗，redis的faq上的，感觉不太靠谱

Posted by: worgen | (20) March 5, 2014 06:34 PM

我是这么想的，另外启动监听的端口，导致整体的数据库设计太过于复杂。
有一个方案，在redis本身上去修改一些东西，每个key都设置一个时间戳，到达一定的时间，把这些key落地。查询的时候，先在redis本上去查找下，如果没有再从硬盘上获取。当然这个会查询2次的情况，但是热数据，是非常快的。
为了解决查询2次的事情，我们只落地value
目前我们的游戏是采用的这套机制，k,v全部落地

Posted by: 郭建炯 | (19) March 5, 2014 05:07 PM

何不试试redis-nds呢？完全满足贵产品需求。

Posted by: 顺娃 | (18) March 5, 2014 02:16 PM

@zengzizhao

如果本地连接 redis 还会发生连接断开的话，除了系统过载，就只能是系统的 bug 了。

如果依赖这个机制，那就算清楚，保证不过载,或在过载时启用其它机制就可以了。

Posted by: Cloud | (17) March 5, 2014 01:37 PM

“Redis 有一个功能 “Redis Keyspace Notifications”

http://redis.io/topics/notifications

可以监测 key 的变更事件，如果不复杂倒是可以尝试用于数据落地。”

"Because Redis Pub/Sub is fire and forget currently there is no way to use this feature if you application demands reliable notification of events, that is, if your Pub/Sub client disconnects, and reconnects later, all the events delivered during the time the client was disconnected are lost."
基于fire and forget，Keyspace Notifications这个似乎并不适合用来做落地。

Posted by: zengzizhao | (16) March 5, 2014 11:46 AM

用redis就是为了有个持久化的功能，云云， 你这样子搞，不就是要用memcache么

Posted by: zozoiiiiii | (15) March 5, 2014 10:48 AM

感谢分享！
感觉主要还是写入量比较大这点比较关键。sync同一主机上的多个实例导致服务停止的时候 额外占内存的多少主要应该主要取决于写入量吧。这时IO紧张，写盘慢，系统对写入量的敏感程度被放大了。一方面fork时有Copy-on-Write，父进程如果被写到的脏页多 需要复制给子进程的脏页也就多（这里也有你上一篇提到的冷热数据混在一起导致无谓复制的问题），另一方面写入量大时，做快照期间的写入日志大小也会很客观。
p.s. 据说在2.8release之前weibo自己实现了比较健壮的PSYNC哈

Posted by: Jones0036 | (14) March 5, 2014 09:12 AM

>Redis Keyspace Notifications
这个是2.8才有的功能.

reids是有建议不要使用超过50%的物理内存就是担心fork的时候的问题.

另外在没有keyspace notifications时,有打算过用一个"笨办法"来处理冷数据,dump 到非生产机器(内存>=生产机器),然后使用OBJECT来确认key的REFCOUNT,并进行对部分冷数据进行移除.

Posted by: zwei | (13) March 5, 2014 12:15 AM

内存被filesystem cache干掉了,也许用DIO+AIO可以避免这件事,但是不知道你们的备份脚本能不能做到

Posted by: xkglob | (12) March 4, 2014 10:28 PM

感觉设计中Redis大部分是冷数据啊，后面提到数据更新操作少，但前面又说了写操作很频繁，不是很理解。如果更新操作不多的话 ，使用间隔长一些的snapshot + AOF是不是就可以，另外，如果可以修改一下redis，增加了对快照速度的控制，不要把磁盘的IO吃完，影响整体的性能，可能也会好一些

Posted by: hanson | (11) March 4, 2014 08:40 PM

btw，有这样的feature吗：留言有回复时候，发提醒信件到我的邮箱

Posted by: 天浪 | (10) March 4, 2014 07:54 PM

这样的机制还会有物理机单点故障的风险
增加一个同步机制如何：
逻辑实例 把redis write_op_flow发送到给基于kafka的queue，queue本身可以设置主从热备，从机把这些 op_flow恢复到本机的redis上。

数据落地可以在从机做；
游戏服务器维护时候可以同步重置queue；
主机单点故障时候，从机确认处理完queue所有消息后，接替主机服务；

Posted by: 天浪 | (9) March 4, 2014 07:51 PM

那这个一致性如何保证? 比如在 监护服务 挂了之后的措施是怎样的?

Posted by: zava | (8) March 4, 2014 07:41 PM

LevelDB不错，性能更高，尤其是写性能非常出众，其实它也自带了C的访问接口，在乎是C还是C++实现的意义不大

Posted by: dwing | (7) March 4, 2014 06:56 PM

tx有是类似的实现，不过直接就是shm，逻辑进程（read+update）+数据进程（load+writeback），不过最后还是与mysql对接，估计是为了一些运营分析方便吧，不然总是免不了需要专门写一个数据解析接口吧

Posted by: bear | (6) March 4, 2014 06:55 PM

整个游戏没有用db做落地？ 现在只有unqlite？ 那运营上的数据分析这些都是怎么获得数据呢？写一个解析接口从数据文件获得？

Posted by: bear | (5) March 4, 2014 06:51 PM

这样改就更加像我很久之前的一个实现了。。。不过我没有直接拦截redis指令。。。

Posted by: 巢鹏 | (4) March 4, 2014 06:29 PM

这个做法的本质，就不是使用数据库，而使用内存做数据交换。系统启动的时候，数据从数据库中加载到缓存；用户访问系统时，从缓存中加载数据、更新缓存；检测到缓存中有脏数据时，批量刷新到数据库中。
如果缓存崩溃，系统数据回退到上一次数据库更新状态；如果数据库崩溃而缓存还在，慢慢更新数据库即可。
是否如此？

Posted by: lifengpei | (3) March 4, 2014 06:28 PM

谢谢， 这样就更简单了。不用改原来的游戏服务器逻辑。

Posted by: Cloud | (2) March 4, 2014 05:34 PM

Redis 有一个功能 “Redis Keyspace Notifications”

http://redis.io/topics/notifications

可以监测 key 的变更事件，如果不复杂倒是可以尝试用于数据落地。

Posted by: 路人甲 | (1) March 4, 2014 05:23 PM

POST A COMMENT

非这个主题相关的留言请到：留言本

名字：

Email 地址：

为了验证您是人类，请将六加一的结果（阿拉伯数字七）填写在下面：

URL:

记住我的信息？

留言：
（不欢迎在留言中粘贴程序代码）

提交