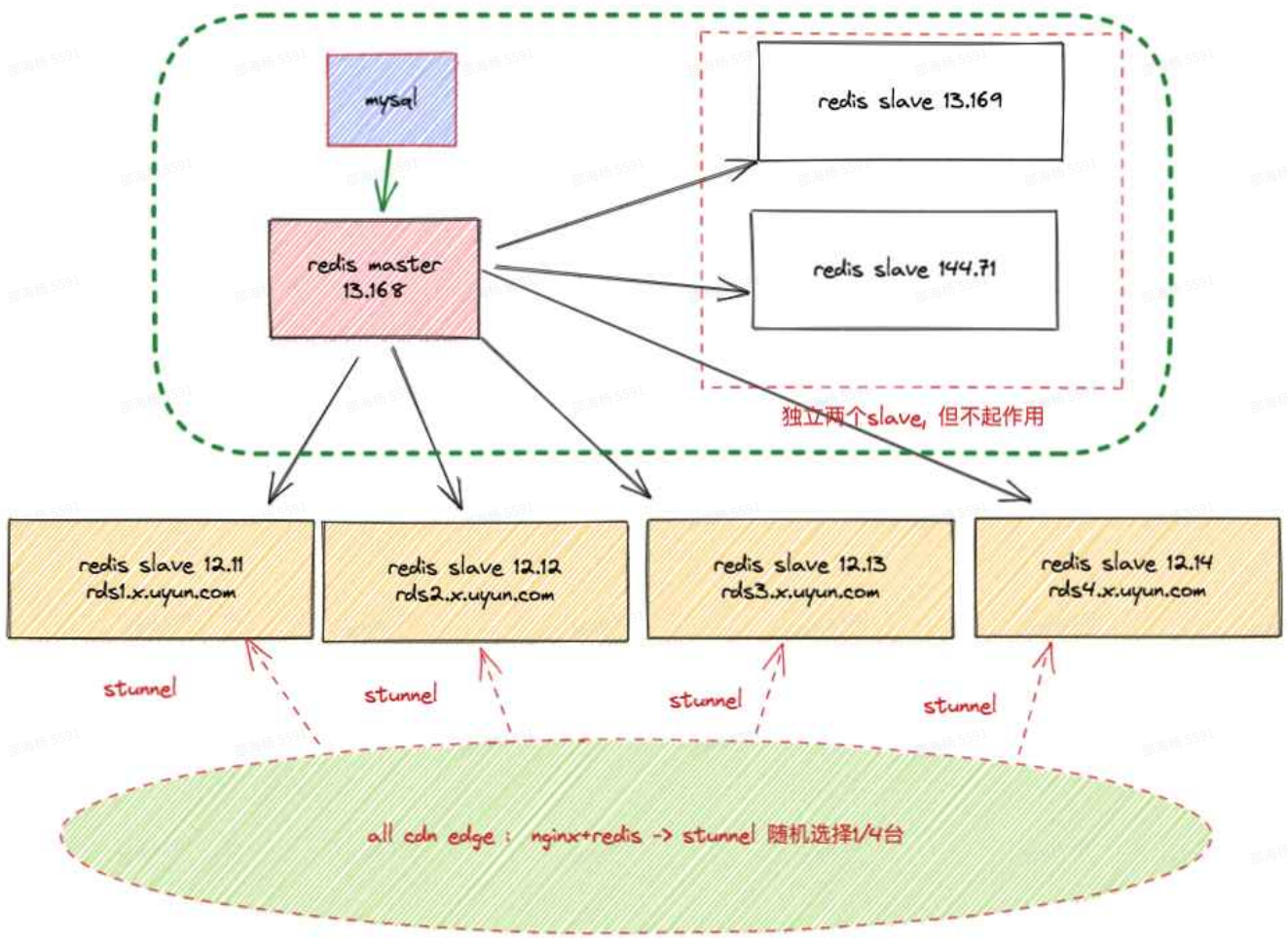
+又拍云之Redis的改进之路

👍 又拍云作为国内首创可编程的专业CDN服务提供商, 利用CDN 边缘网络规模和性能，可以自定义编写规则来满足常用业务场景服务大量的客户,这些客户的源数据**如边缘重定向、请求限速、自定义错误页面、访问防盗链控制、 HTTP 头部管理等元数据管理**都需要快速的同步到边缘的节点服务器上, 因此,对比了几个方案以后, 在2014年初就使用了redis 2.8版本作为数据同步的解决方案. 下图就是我们最初的架构设计:



这个方案在过去的几年里确实起到重要的作用, 但是随着边缘服务器的日益增长和同步数据量的增加, 再加上服务器硬件的老化故障等原因, 这个架构的缺点也很明显, 因此也埋下了技术债的引子

❌
1. 出于安全考虑，相互redis之间的通信数据都需要加密，但redis本身不支持ssl加密，因此所有的边缘服务器都通过stunnel套接做中转服务器, 实际工作状态下，stunnel的性能不足,造成服务器的cpu负载高
2. redis的数据主从都是长连接且尽量保持从同一源做同步，因此早期边缘服务器都是通过域名解析的方式来获取源服务器的IP地址，好处是实施部署简单，缺点是dns无法获知后端服务器的处理能力, 造成每台机器上的长连接负载不均衡; 后端服务出故障后dns也无法自动处理, 就算dns上及时切换解析，也会因ttl生效时间内导致真空期数据不一样， 只能提供旧数据应急。
3. 因为历史遗留原因, 边缘redis版本大都是2.x低版本, 而低版本只能通过sync做全量同步, 因此中转服务器和主服务器的异常,都会造成全网的雪崩效应从而同步阻塞, 无法快速的同步元数据到边缘。
4. 因为用的redis非常早期,当初只有主从模式可以采用, 什么哨兵和集群改造也没有实现, 如今主服务器是个单点风险,很容易造成源头上的重大故障。

> 🚫 技术负债又称技术债，指开发人员为了加速软件开发，在应该采用最佳方案时进行了妥协，改用了短期内能加速软件开发的方案，从而在未来给自己带来的额外开发负担。这种选择就像一笔债务一样，虽然眼前看起来可以得到好处，但必须在未来偿还。

因此, 我们必须要付出额外的时间和精力进行重构, 修复之前的妥协所造成的问题及副作用，把架构改善为最佳实现方式。

我们把改造过程分成几个步骤：

1. 加强SSL的安全防护, 建议升级到openssl最新的稳定版本

> 💡 SSL可能是大家接触比较多的互联网安全协议之一，看到某个网站地址用了“https://”开头，就是采用了SSL安全协议。**OpenSSL是一种开放源码的SSL实现，用来实现网络通信的高强度加密，现在被广泛地用于各种网络应用程序中。如此重要的项目多年来始终面临着资金和人手不足的窘境，多数工作都要由为数不多的黑客和爱好者及志愿者来完成。幸好现在纳入Linux基金会资金资助对象, 但依然还是有不断的新漏洞暴露出来,因此我们需要及时关注和跟进.**

参考最新的openssl漏洞危险等级报告

漏洞等级：中危

漏洞类型：命令执行和拒绝服务

受影响版本：

CVE-ID	影响版本	安全版本
CVE-2022-1292	OpenSSL 1.0.2-1.0.2zd OpenSSL 1.1.1-1.1.1n OpenSSL 3.0.0、3.0.1、3.0.2	OpenSSL 1.0.2：升级至 1.0.2ze（仅限高级支持客户） OpenSSL 1.1.1：升级至 1.1.1o OpenSSL 3.0 ：升级至 3.0.3
CVE-2022-1343	OpenSSL 3.0.0、3.0.1、3.0.2	OpenSSL 3.0 ：升级到 3.0.3
CVE-2022-1434	OpenSSL 3.0.0、3.0.1、3.0.2	OpenSSL 3.0 ：升级到 3.0.3
CVE-2022-1473	OpenSSL 3.0.0、3.0.1、3.0.2	OpenSSL 3.0 ：升级到 3.0.3

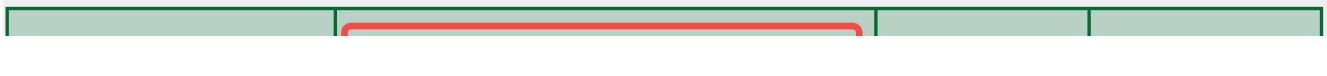
Plain Text

```
./config --prefix=/opt/openssl -fPIC enable-tls1_3 enable-shared enable-zlib no-asm  no-rc4
```

> 💊 鉴于RC4算法安全漏洞太多，建议编译时就禁用了。

2. 使用最新的stunnel版本，优化性能，基于安全openssl依赖库，支持TLSv1.2+以上

从截图的红色框中可以看出stunnel在某些算法下的性能最强的, 因此在配置文件中推荐优先使用:



Data throughput	ECDHE-RSA-AES128-GCM-SHA256	688 MB/s	**5.5 Gbit/s**
	ECDHE-RSA-AES256-GCM-SHA384	648 MB/s	5.2 Gbit/s
	ECDHE-RSA-AES128-SHA256	244 MB/s	2.0 Gbit/s
	ECDHE-RSA-AES256-SHA384	204 MB/s	1.6 Gbit/s
	DES-CBC3-SHA	28 MB/s	0.22 Gbit/s
New connections	New session(1,2)	750 connections/s	
	Resumed session(2)	4 700 connections/s	
	PSK authentication(3)	4 460 connections/s	
Concurrent sessions	Unix poll()	limited by available memory(4)	
	Unix select()	500	
	64-bit Windows build	limited by available memory	
	32-bit Windows build	1 000	

Plain Text

```
1  ./configure --prefix=/opt/stunnel --with-ssl=/opt/openssl
```

推荐配置中的优化选项

✅ verify = 3

sslVersionMax = TLSv1.3

sslVersionMin = TLSv1.2

options = NO_SSLv2

options = NO_SSLv3

.....

ciphers = ECDHE-RSA-AES128-GCM-SHA256:ECDHE:ECDH:AES:HIGH:!NULL:!aNULL:!MD5:!ADH:!RC4:!DH:!DHE

📌 可以通过亚洲诚信的网站来做https的可信等级检测和验证

检测部署SSL/TLS的服务是否符合行业最佳实践，PCI DSS支付卡行业安全标准，Apple ATS规范。

开启HSTS后能够提升到A+,具体请参考 《HTTPS安全与兼容性配置指南》

4. 编译最新的redis-6.2.x稳定版，功能强大丰富，无需依赖高版本的GCC要求

📌 Redis 6.2与7.0的区别在于Redis 7.0 几乎包括了对各个方面的增量改进。其中最值得注意的是 Redis Functions、ACLv2、command introspection 和 Sharded Pub/Sub，7.0 版添加了近 50 个新命令和选项来支持这种演变并扩展 Redis 的现有功能。

总而言之, redis 7.0更加强大, 但为了考虑到与原来的redis代码的完全兼容性和对生产环境稳定性的综合考虑, 我们认为redis 6.2优点也足够多和强大, 满足我们产线上的要求

❤️ 多线程 IO（Threaded I/O）

众多新模块（modules）API

更好的过期循环（expire cycle）

支持SSL

ACLs 权限控制

RESP3 协议

客户端缓存（Client side caching）

无盘复制&PSYNC2

Redis-benchmark支持集群

Redis-cli 优化、重写 Systemd 支持

Redis 集群代理与 Redis 6 一同发布（但在不同的 repo）

RDB更快加载

SRANDMEMBER和类似的命令具有更好的分布

STRALGO 命令

带有超时的 Redis 命令更易用

重点介绍一下PSYNC2的特性, 这也是我们架构改进升级的重点特性之一:

👍 在redis cluster的实际生产运营中，实例的维护性重启、主实例的故障切换（如cluster failover)操作都是比较常见的(如实例升级、rename command和释放实例内存碎片等）。而在redis4.0版本前，这类维护性的处理，redis都会发生全量重新同步，导到性能敏感的服务有少量受损。**而psync2主要让redis在从实例重启和主实例故障切换场景下，也能使用部分重新同步。**

直接下载源代码编译:

```
Plain Text
1  # make BUILD_TLS=no
```

推荐配置，添加以下选项增强性能

✅ io-threads-do-reads yes

io-threads 8

aof-use-rdb-preamble yes

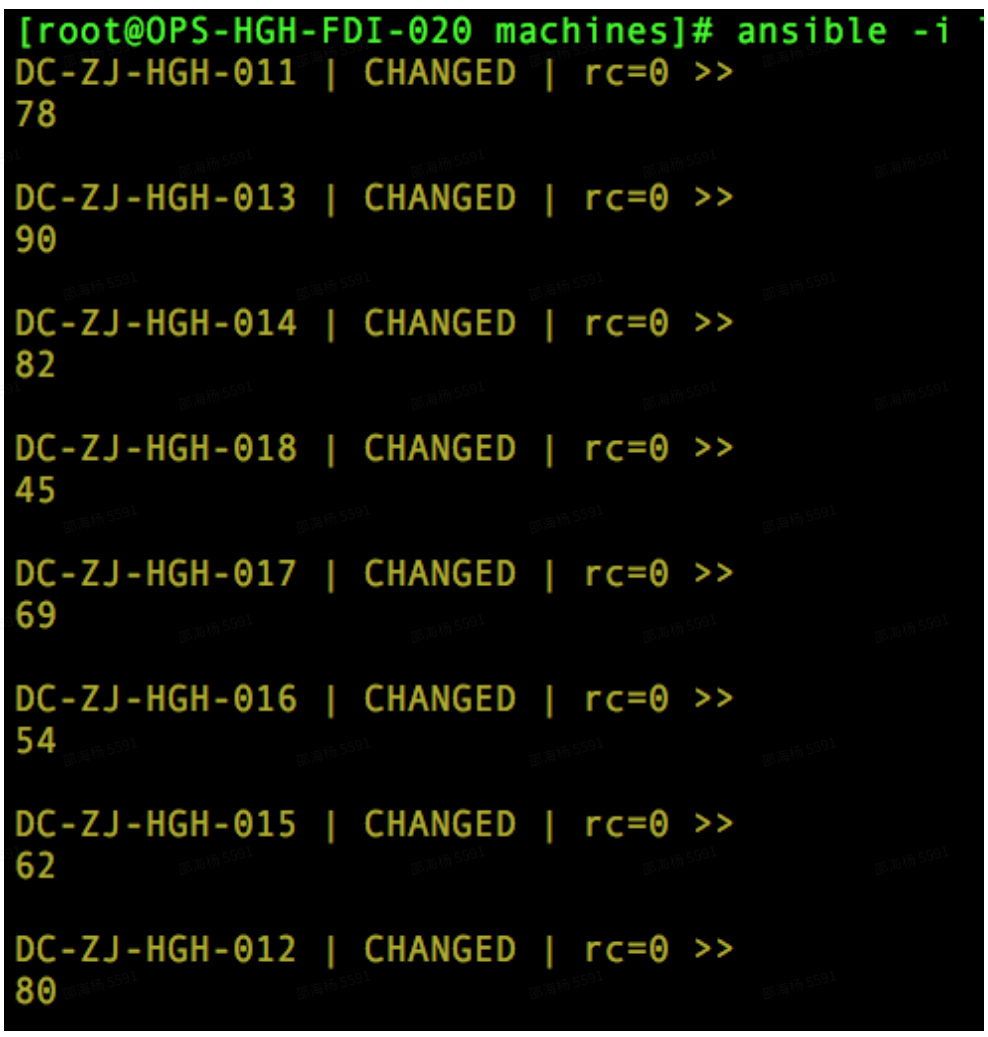
在我们的测试过程中, 发现redis+tls有几个问题:

🚫
- redis开启tls后, 性能下降30%
- redis对openssl的强依赖性, 考虑到openssl的过往高危漏洞不断, 如果要不断修复漏洞要重新编译redis,导致运维更新成本过高
- redis升级后, 要重新同步数据, 增加了出故障的机率或让生产停摆

✅ 所以, 我们还是决定使用第三方程序stunnel来加固安全，方便升级和修复漏洞,又不影响后端连接, 从而保障了redis的工作连续性和稳定可靠性

5. 基于APISIX+TLS托管，基于TCP的哈希一致性做负载均衡来替换dns的轮询，效能显著

apisix做tcp代理, 因为与redis改造没什么大的关联, 直接配置一下就可以使用了, 在这里就不展开了,直接上改造后的连接数统计截图, 从实际的apisix的连接数也可以看出负载被分摊到不同的后端, 数量比较均衡, 而且边缘服务器重启也会利用psync2做快速的增量同步



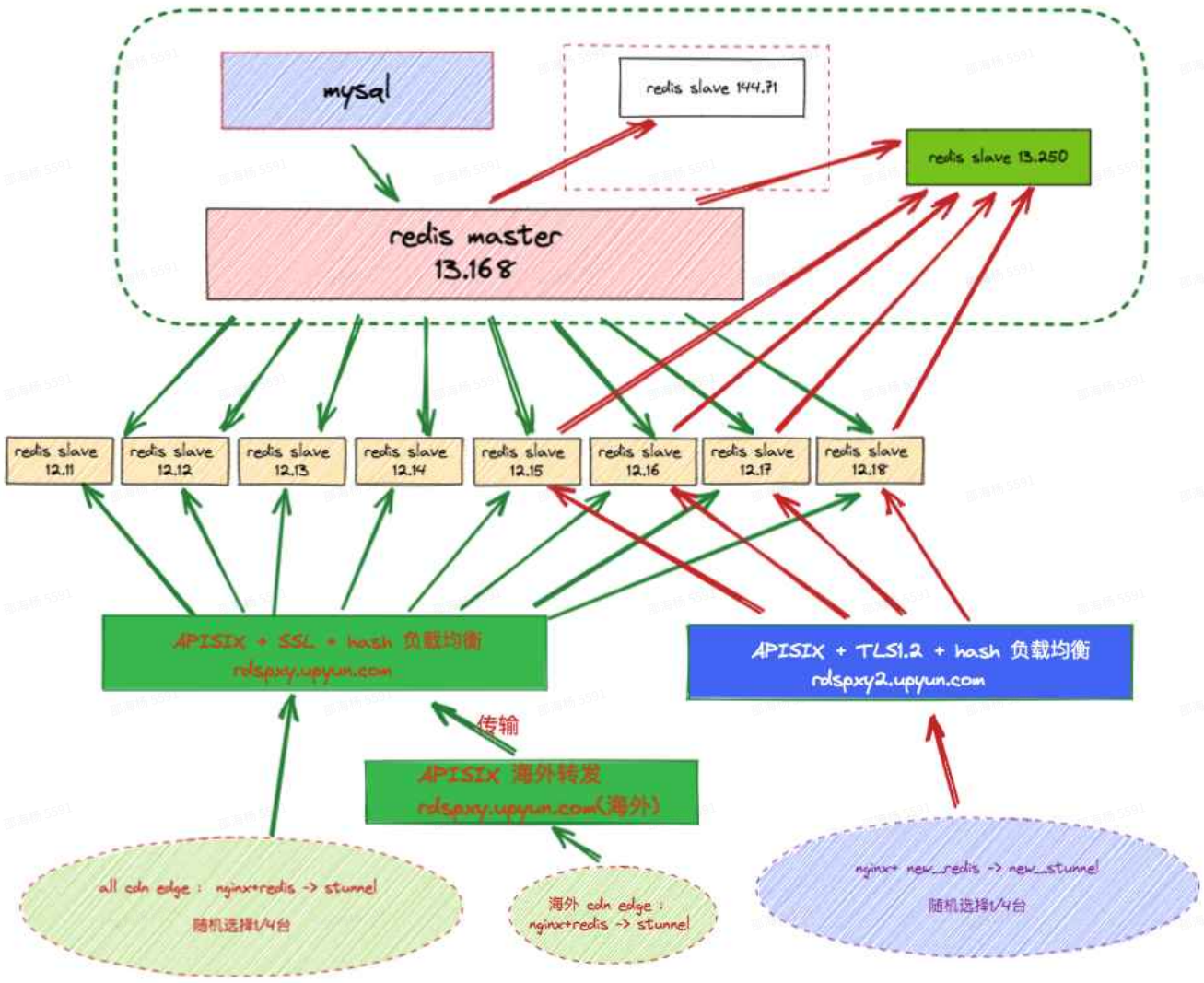
6. 使用redis-shake做定制化的数据同步

在架构改进的过程中, redis-shake这个工具也可以说说, 它是阿里云Redis&MongoDB团队开源的用于redis数据同步的工具。它支持**解析、恢复、备份、同步**四个功能。以下主要介绍同步sync。

- 恢复restore：将RDB文件恢复到目的redis数据库。
- 备份dump：将源redis的全量数据通过RDB文件备份起来。
- 解析decode：对RDB文件进行读取，并以json格式解析存储。
- **同步sync**：支持源redis和目的redis的数据同步，支持全量和增量数据的迁移，
- **同步rump：支持源redis和目的redis的数据同步，仅支持全量的迁移。采用scan和restore命令进行迁移，支持不同云厂商不同redis版本的迁移。**

> 👍 我们原来有一个做过源代码修改过的redis,只会同步想要的空间, 虽然好用,但真的要在新代码上重新编译一个, 却已经找不到原来的负责人, 这也是很多项目年久失修的通病, 但通过redis-shake这样的开源工具,只要通过它简单配置一下就可以实现我们想要的功能
>
> - filter.db.whitelist / blacklist
> - filter.key.whitelist / blacklist
> - filter.command.whitelist / blacklist

7. 现在的架构及未来的展望



在现在的架构中, 我们在原来的三层架构基础上, 又拆分和强化了三层架构:

1. DNS层解析到vip, VIP利用了bgp/ospf的动态网关路由协议,对应后面一组服务器集群服务
2. 负载均衡层: 利用 apisix + tls1.2+ + tcp的哈希一致性连接 , 把redis的主从连接均衡,故障转移
3. 边缘CDN节点, 利用redis高版本所带来的技术红利,psync的增量同步, 加上stunnel+tls1.2实现了加密传输

4. 下一个阶段, 还要继续把数据中心的redis主改造成redis哨兵模式(考虑到程序代码要对哨兵模式做兼容性改造, 第一阶段先不上, 一切都为了生产环境中的稳定性)

参考文档:

1. 如何检查网站的TLS版本
2. redis特性之复制增强版 PSYNC2
3. 通俗易懂的 Redis 架构模式详解