KEYSPACE

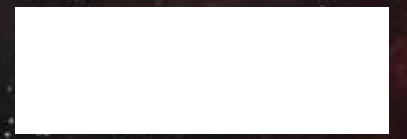
2025 / 北京

Valkey 9.0: 让集群的水平伸缩更稳更快

Atomic Slot Migration

宋平凡 | 2025/12/13

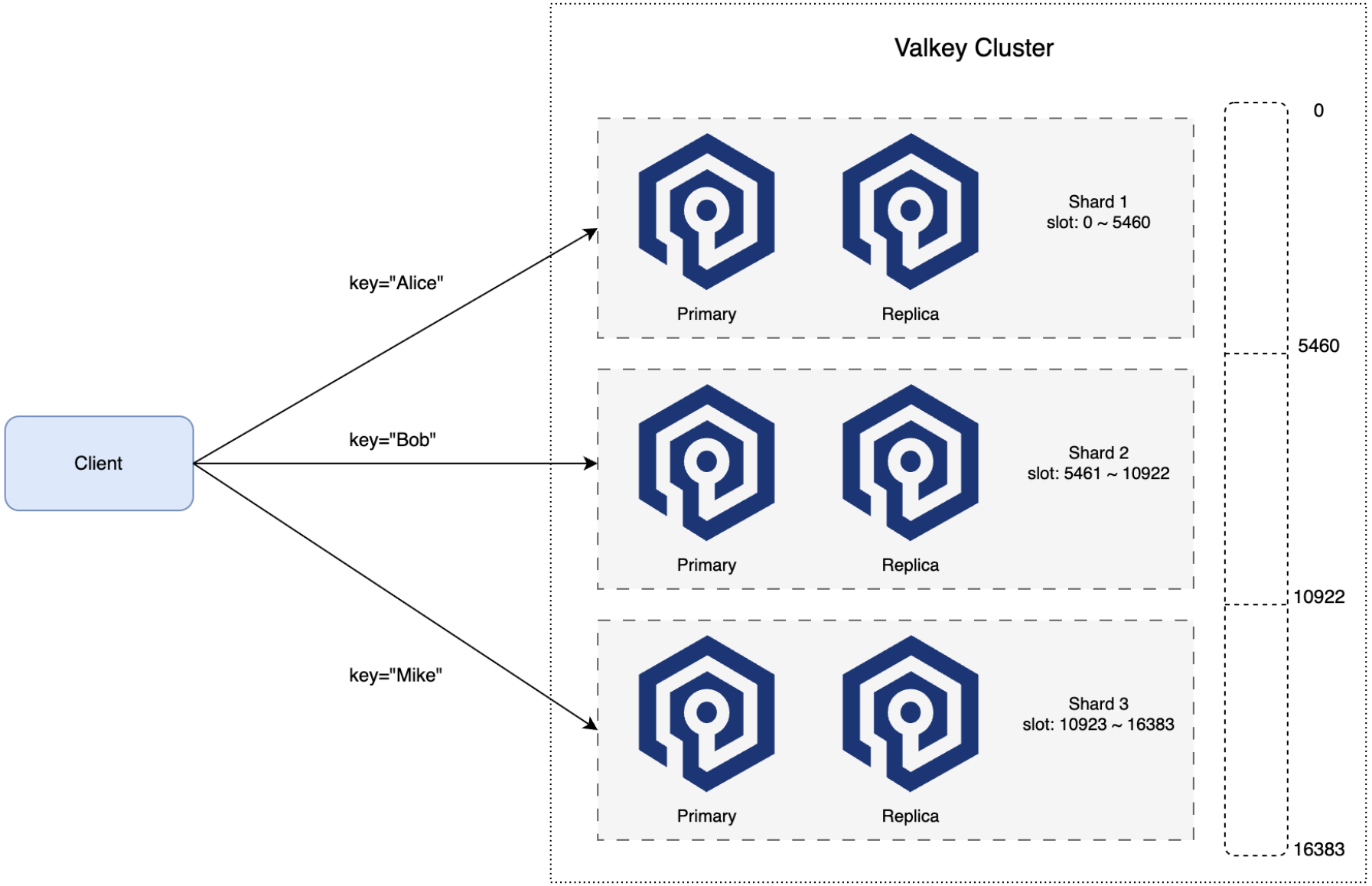
腾讯云研发



01

从集群的基础原理说起

基于hash slot分片的Valkey集群



如何确定一个key所属的hash slot

slot = CRC16(key) % 16384

```
127.0.0.1:7000> cluster keyslot Alice
(integer) 2649
127.0.0.1:7000> cluster keyslot Bob
(integer) 9277
127.0.0.1:7000> cluster keyslot Mike
(integer) 14643
127.0.0.1:7000> cluster keyslot {Mike}:goods
(integer) 14643
127.0.0.1:7000> cluster keyslot {Mike}:friends
(integer) 14643
```

https://valkey.io/commands/cluster-keyslot/

```c
/* We have 16384 hash slots. The hash slot of a given key is obtained
 * as the least significant 14 bits of the crc16 of the key.
 *
 * However, if the key contains the {...} pattern, only the part between
 * { and } is hashed. This may be useful in the future to force certain
 * keys to be in the same node (assuming no resharding is in progress). */
unsigned int keyHashSlot(char *key, int keylen) {
    int s, e; /* start-end indexes of { and } */

    for (s = 0; s < keylen; s++)
        if (key[s] == '{') break;

    /* No '{' ? Hash the whole key. This is the base case. */
    if (s == keylen) return crc16(key, keylen) & 0x3FFF;

    /* '{' found? Check if we have the corresponding '}'. */
    for (e = s + 1; e < keylen; e++)
        if (key[e] == '}') break;

    /* No '}' or nothing between {} ? Hash the whole key. */
    if (e == keylen || e == s + 1) return crc16(key, keylen) & 0x3FFF;

    /* If we are here there is both a { and a } on its right. Hash
     * what is in the middle between { and }. */
    return crc16(key + s + 1, e - s - 1) & 0x3FFF;
}
```

如何查看集群的拓扑信息

cluster

nodes

cluster slots

cluster shards

```
❯ ./src/valkey-cli -p 30001 cluster nodes
f68b6cbe2c8e341e04ae5a79e342bb7e1068f055 127.0.0.1:30002@40002 master - 0 1764392146575 2 connected 5461-10922
27daad6ff6a7e330e5297dbd4762d187610f4394 127.0.0.1:30001@40001 myself,master - 0 0 1 connected 0-5460
b3465c0acd3ac18d35bee3c0b17dd56b110b8471 127.0.0.1:30006@40006 slave 27daad6ff6a7e330e5297dbd4762d187610f4394 0 1764392146473 1 connected
38580acfdc8ef5d181019362bdea8528aa4b6e2d 127.0.0.1:30004@40004 slave f68b6cbe2c8e341e04ae5a79e342bb7e1068f055 0 1764392146575 2 connected
2c57169e884cbe134ddf40340f193a0516e422d4 127.0.0.1:30003@40003 master - 0 1764392146575 3 connected 10923-16383
7a4675e1ab1cfa659d2781fbd3fd42ac79e91125 127.0.0.1:30005@40005 slave 2c57169e884cbe134ddf40340f193a0516e422d4 0 1764392146473 3 connected
```

https://valkey.io/commands/cluster-nodes/

水平伸缩这个事为什么很重要？

业务数据量和流量的变化要求数据库有应对能力

- 所有业务都有生命周期，上升期需要水平扩容，下降期需要水平缩容。
- 部分业务正常运营中也有节假日高峰和活动高峰，高峰前需要扩容，高峰后需要缩容。

Valkey单点处理能力缺乏纵向扩展性

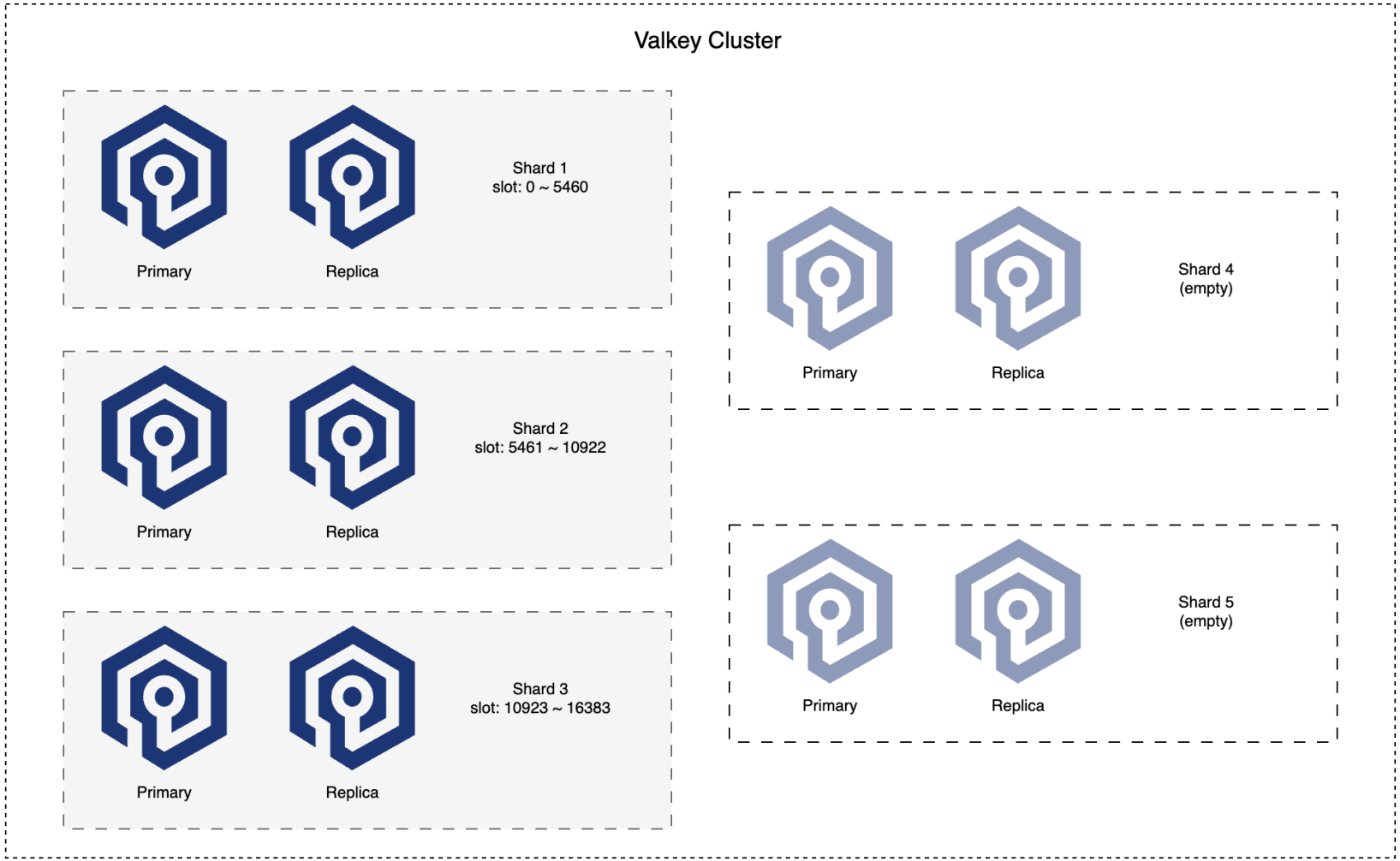
- Valkey的命令执行是单线程模型，节点主线程打满CPU，无法通过加核增加处理能力。

Valkey单点内存过大容易引发系列问题

- Valkey的内存过大导致fork的阻塞时间过长，几十GB的fork就可能阻塞几百毫秒。
- Valkey的内存过大导致全量同步耗时过长，增量请求积压过多容易超出output buffer限制。

从三分片到五分片？

水平伸缩 → slot迁移



slot迁移方案需要解决的核心问题

迁移 = 搬迁 + 转移

（数据） （归属权）

- slot数据如何搬迁？
- slot归属权如何转移？
- 迁移中业务请求如何处理？

02

老版本的slot迁移方案

老版本slot迁移方案的核心思路

slot数据如何搬迁？

- 客户端逐key搬迁

→ MIGRATE命令

slot归属权如何转移？

- 客户端直接设置

→ CLUSTER SETSLOT命令

迁移中业务请求如何处理？

- 不确定key在哪边，需要分别尝试

→ ASK重定向

老版本slot迁移方案的流程

标记状态阶段(slot粒度)

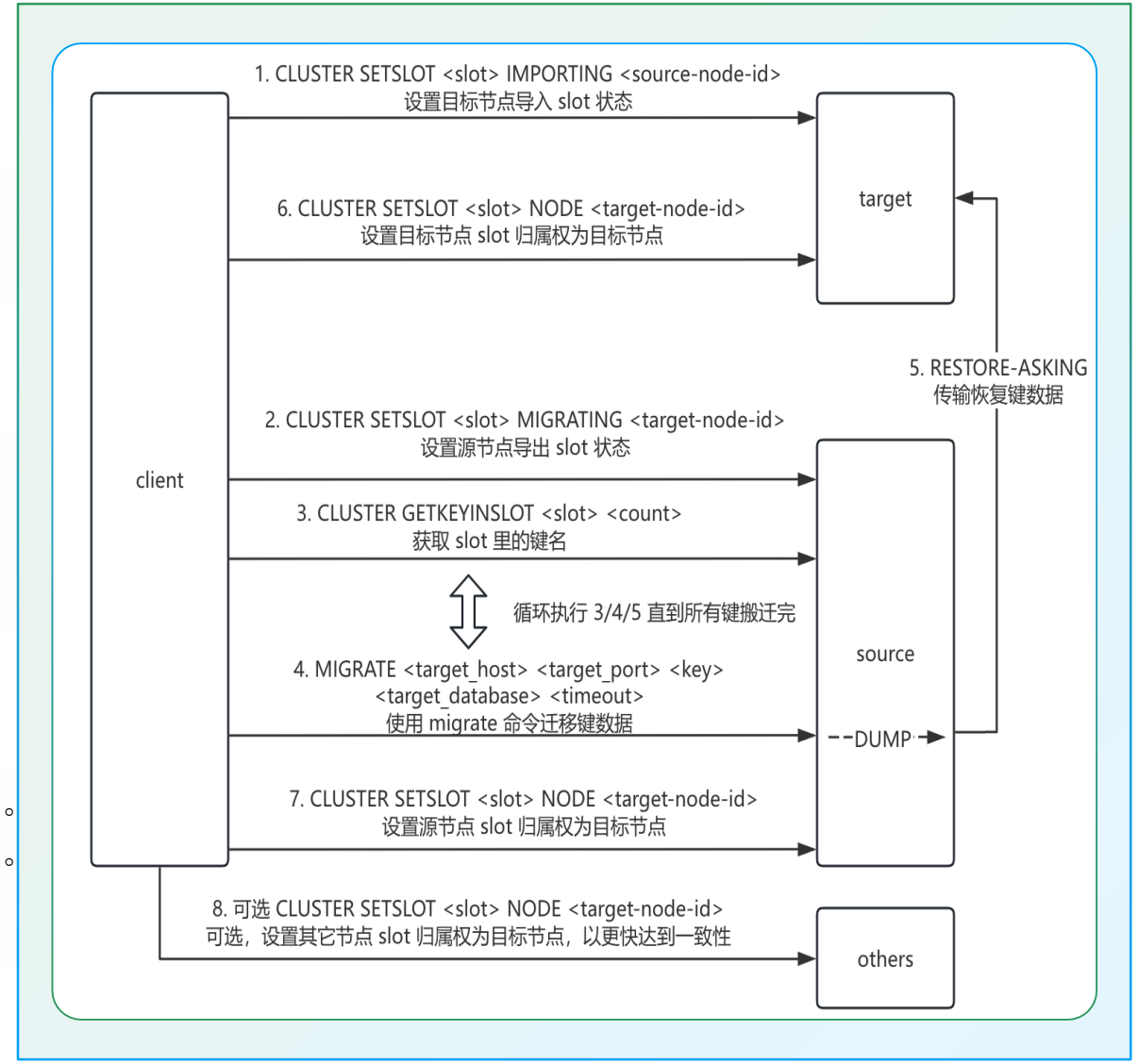
1. 在目标节点标记slot的 IMPORTING（导入）状态。
2. 在源节点标记slot的 MIGRATING（导出）状态。

数据搬迁阶段(key粒度)

1. 在源节点获取一批属于该slot的key 。
2. 在源节点通过 MIGRATE 命令将这一批key同步搬到目标节点。
3. 循环执行上述步骤，直至该slot中所有都搬到了目标节点。

归属转移阶段(slot粒度)

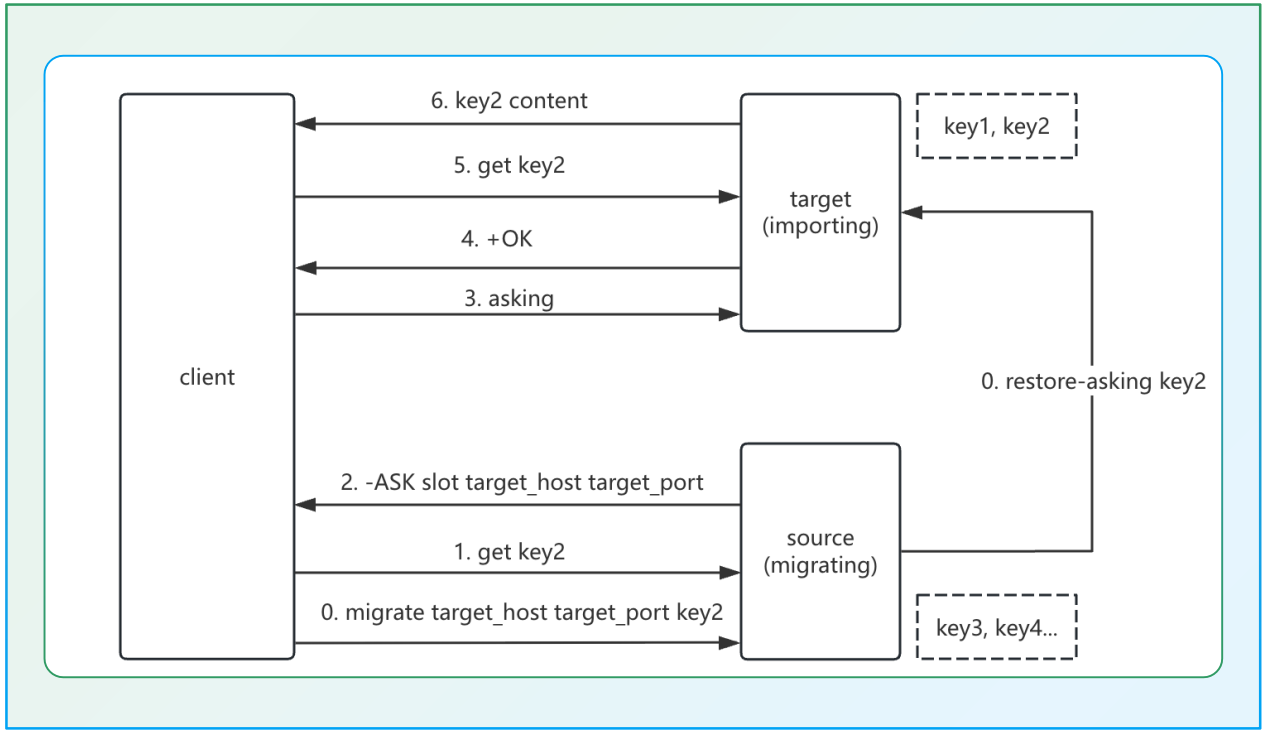
1. 清理目标节点的 IMPORTING 状态，并转移slot归属权到此节点。
2. 清理源节点的 MIGRATING 状态，并更新slot归属权到目标节点。
3. [可选]向集群中其它节点更新slot归属权。



老版本方案迁移中对业务请求的处理

ASK重定向

1. 客户端优先将请求发送到源节点，源节点有这个key就直接返回；
2. 源节点没有这个key，且对应slot处于MIGRATING状态，返回ASK重定向错误；
3. 客户端提取目标节点地址，先发送ASKING命令，标记此连接处于特殊状态，再将原始请求发给目标节点；
4. 目标节点检测到对应slot处于IMPORTING状态，且连接处于特殊状态，临时允许处理此请求，并返回结果。



老版本方案面临的挑战 – 业务影响

多key请求异常

根因：slot数据搬迁不是原子的

- 迁移中，源节点和目标节点都只有一部分key
- 多key命令要求所有访问的key都在同一节点
  - 部分key已搬走，反复重试直到所有key到目标节点，带来时延抖动
  - 部分key不存在，持续报错

大key阻塞问题

根因：MIGRATE是同步阻塞命令

- 搬迁时，源节点的序列化/网络传输/目标节点的反序列化，每一步耗时都和key的大小正相关
- Valkey支持多种复杂结构的key
  - 几十万上百万的元素，搬迁带来严重的时延抖动/超时报错
  - 几百万上千万的元素，搬迁导致节点被判死，集群不可用

业务性能受损

根因：单线程 + ASK重定向

- 搬key的命令占用主线程执行时间，用户请求QPS受损
- ASK重定向导致用户请求网络多一跳，请求时延增加

老版本方案面临的挑战 – 健壮性

流程中断处理困难

根因：数据搬迁不是原子的

- 发起后，源节点和目标节点都只有一部分key
- 中断后，继续或回滚都很困难
  - 流程继续，源 -> 目的
  - 流程回滚，目的 -> 源

迁移状态丢失问题

根因：迁移状态未持久化

- 源端主节点宕，从节点提主，不知道slot在迁移中
- 源端对于已搬到目标节点的key当不存在处理，引起数据一致性问题

迁移状态顺序问题

根因：迁移状态设置不是原子的

- 标记状态，若是先源后目标，反复ASK和MOVED重定向
- 清理状态，若是先源后目标，两边来回MOVED重定向

老版本方案面临的挑战 – 迁移速度

迁移速度过慢

根因：客户端驱动 + 单线程模型

- 每一批key的搬迁，都需要客户端和源节点交互两次，增加额外2个RTT。
- 源节点主线程既要执行业务请求，又要执行搬迁命令，互相影响。

03

新版本的slot迁移方案

Valkey9.0的新方案：Atomic Slot Migration

Introduce atomic slot migration #1949

New issue

Merged **madolson** merged 125 commits into `valkey-io:unstable` from `enjoy-binbin:slot_migration_murphyjacob4` on Aug 12

Conversation 519 | Commits 125 | Checks 62 | Files changed 40 | +5,147 −376

murphyjacob4 commented on Apr 13 • edited by enjoy-binbin — Contributor

Introduces a new family of commands for migrating slots via replication. The procedure is driven by the source node which pushes an AOF formatted snapshot of the slots to the target, followed by a replication stream of changes on that slot (a la manual failover).

This solution is an adaptation of the solution provided by **@enjoy-binbin**, combined with the solution I previously posted at #1591, modified to meet the designs we had outlined in #23.

New commands

- `CLUSTER MIGRATESLOTS SLOTSRANGE start end [start end]... NODE node-id` : Begin sending the slot via replication to the target. Multiple targets can be specified by repeating `SLOTSRANGE ... NODE ...`
- `CLUSTER CANCELMIGRATION ALL` : Cancel all slot migrations
- `CLUSTER GETSLOTMIGRATIONS` : See a recent log of migrations

This PR only implements "one shot" semantics with an asynchronous model. Later, "two phase" (e.g. slot level replicate/failover commands) can be added with the same core.

Reviewers
- zuiderkwast
- enjoy-binbin
- madolson ✓
- hwware
- Copilot
- PingXie ✓

+1 more reviewer
- gmbnomis

Assignees
- enjoy-binbin

Labels

https://github.com/valkey-io/valkey/pull/1949

Atomic Slot Migration的核心思路

slot数据如何搬迁？
- 借鉴主备Replication，但需限定slot

→ Slot Replication

slot归属权如何转移？
- 借鉴Manual Failover，但需限定slot

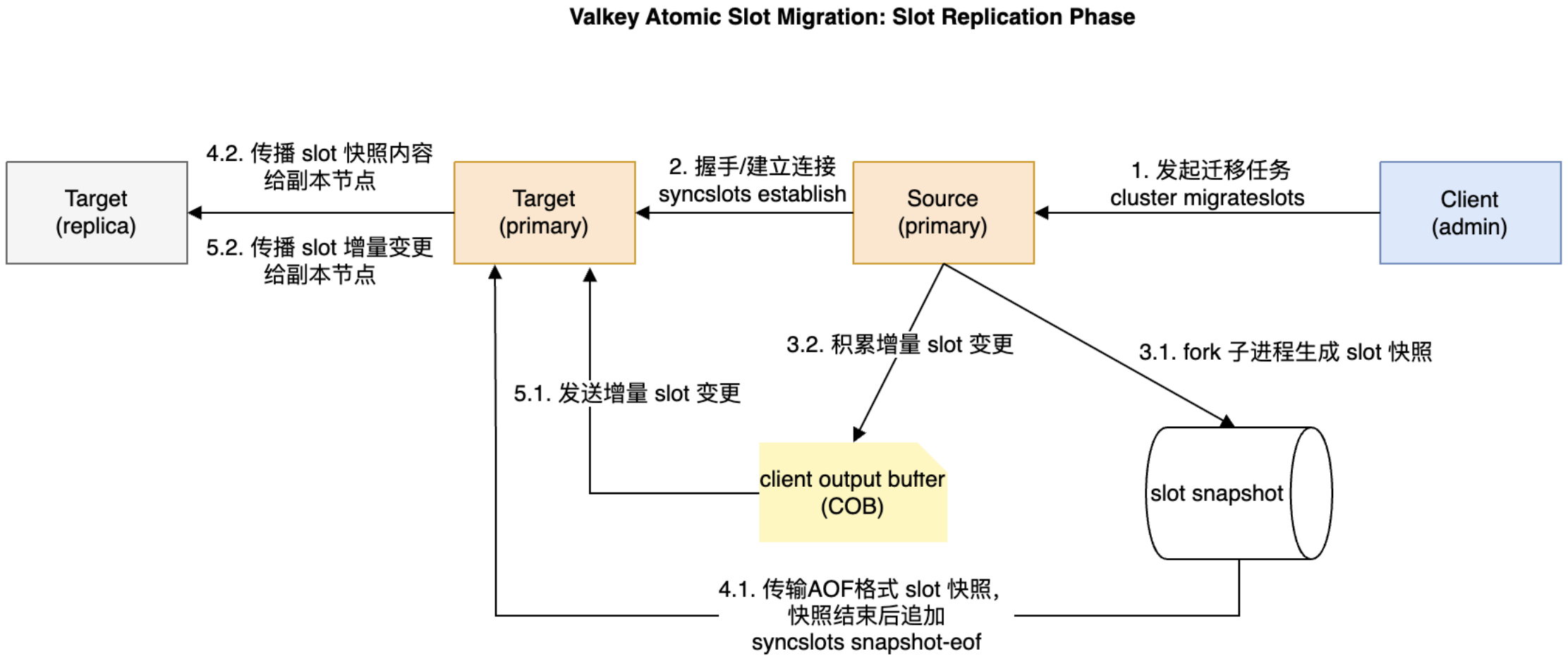
→ Slot Failover

迁移中业务请求如何处理？
- 搬迁中源节点不删数据，直接访问

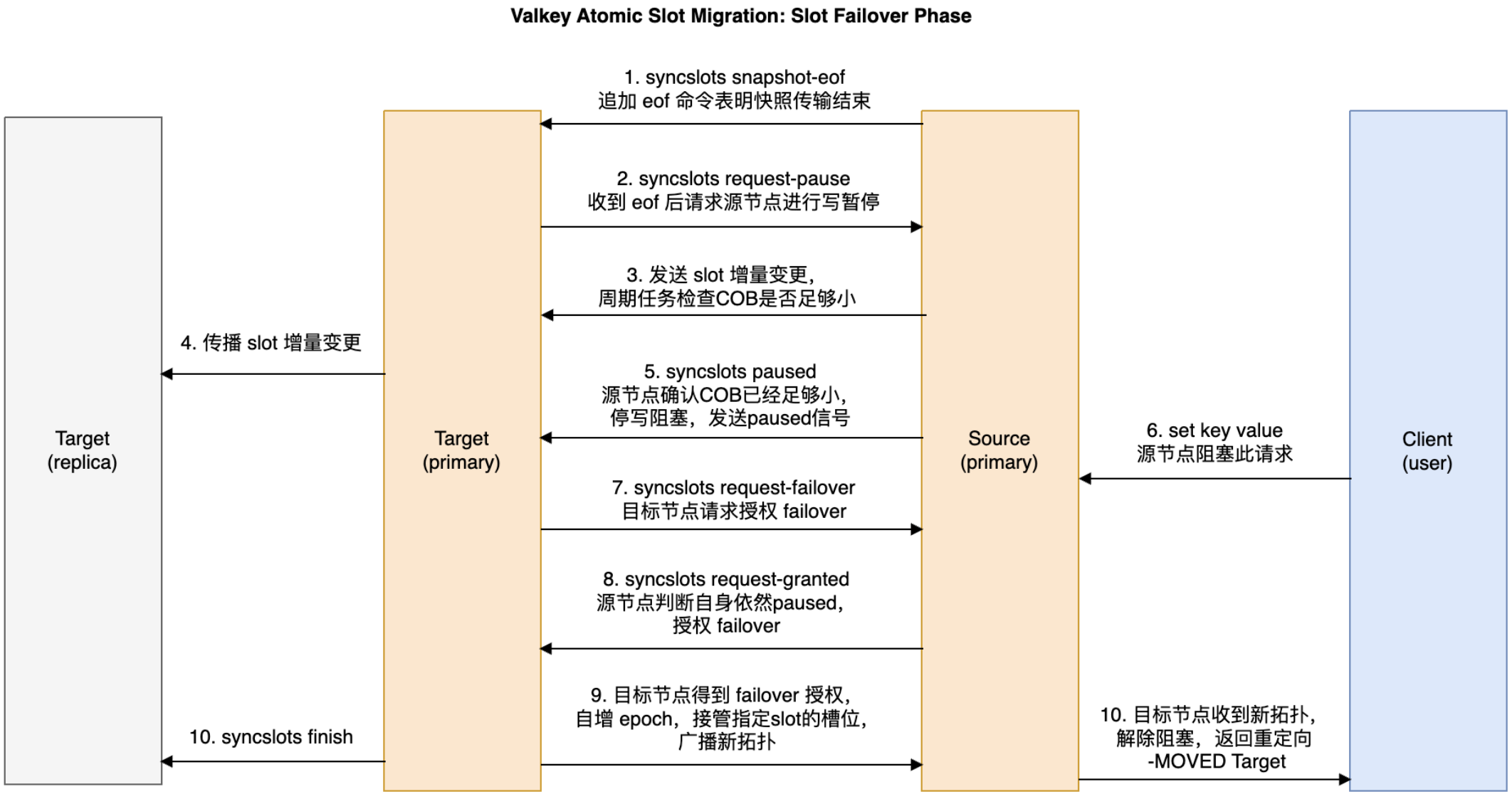
→ 源节点正常处理

Migration via Replication and Failover

Atomic Slot Migration: Slot Replication



Atomic Slot Migration: Slot Failover



新老迁移方案的对比：对业务请求的影响

Legacy Slot Migration(老方案)

- 迁移大key会阻塞主线程
- 多key请求会报错
- 需ASK重定向，时延上升
- 源节点在主线程dump，挤占业务请求的CPU使用

Atomic Slot Migration(新方案)

- 迁移大key不阻塞主线程
- 多key请求源节点正常处理
- 无需重定向，时延不变
- 源节点在子进程传快照，不会影响主进程处理业务请求

新老迁移方案的对比：健壮性与易用性

Legacy Slot Migration(老方案)

- slot迁移状态可能丢失
- 源和目标的迁移状态可能乱序
- 流程中断/回滚困难
- 客户端驱动，需持续交互

Atomic Slot Migration(新方案)

- 没有迁移状态丢失问题
- 没有迁移状态乱序问题
- 流程中断/回滚简单
- 源节点驱动，一键发起

新老迁移方案的对比：迁移速度

测试场景：

1. 两个集群都使用基于Valkey9.0.0编译的同一个二进制搭建，且部署在同一个region；
2. 发起迁移的客户端单独部署在同region的另一台机器上；
3. 老方案使用valkey-cli --cluster rebalance 命令完成迁移，参数采用默认值；
4. 发起迁移前，两个都提前填充了40GB数据(string类型，长度16KB)；
5. Heavy Load场景的背景压力，使用memtier-benchmark工具，set/get 1:10。

Test Case	Legacy Slot Migration	Atomic Slot Migration	Speedup
No Load: 3 to 4 shards	1m42.089s	0m10.723s	9.52x
No Load: 4 to 3 shards	1m20.270s	0m9.507s	8.44x
Heavy Load: 3 to 4 shards	2m27.276s	0m30.995s	4.75x
Heavy Load: 4 to 3 shards	2m5.328s	0m27.105s	4.62x

注：此处的测试数据引用自valkey官网的博客，https://valkey.io/blog/atomic-slot-migration/

如何使用Atomic Slot Migration

- CLUSTER MIGRATESLOTS SLOTSRANGE start-slot end-slot NODE node-id
  - 给源节点发送，向目标节点发起指定slot范围的迁移任务。

- CLUSTER GETSLOTMIGRATIONS
  - 迁移任务运行期间，循环给源节点发送，获取迁移任务的运行状态，一直到任务结束或失败。

- CLUSTER CANCELSLOTMIGRATIONS
  - 迁移任务运行期间，在必要情况下，给源节点发送，安全地中断并回滚迁移任务。

THANK YOU!

Q&A