腾讯面试：1亿用户的好友关系如何秒级查询共同好友？这套方案让性能提升100倍！

原创 Fox爱分享 Fox爱分享 2025年02月23日 07:00 湖南

昨天有个小伙伴给我分享了他去腾讯面试的经历。他被问到了一道面试题：

“如何用Redis处理1亿级用户的好友关系？怎样秒级计算出任意两人的共同好友？”

小伙伴直接懵了。

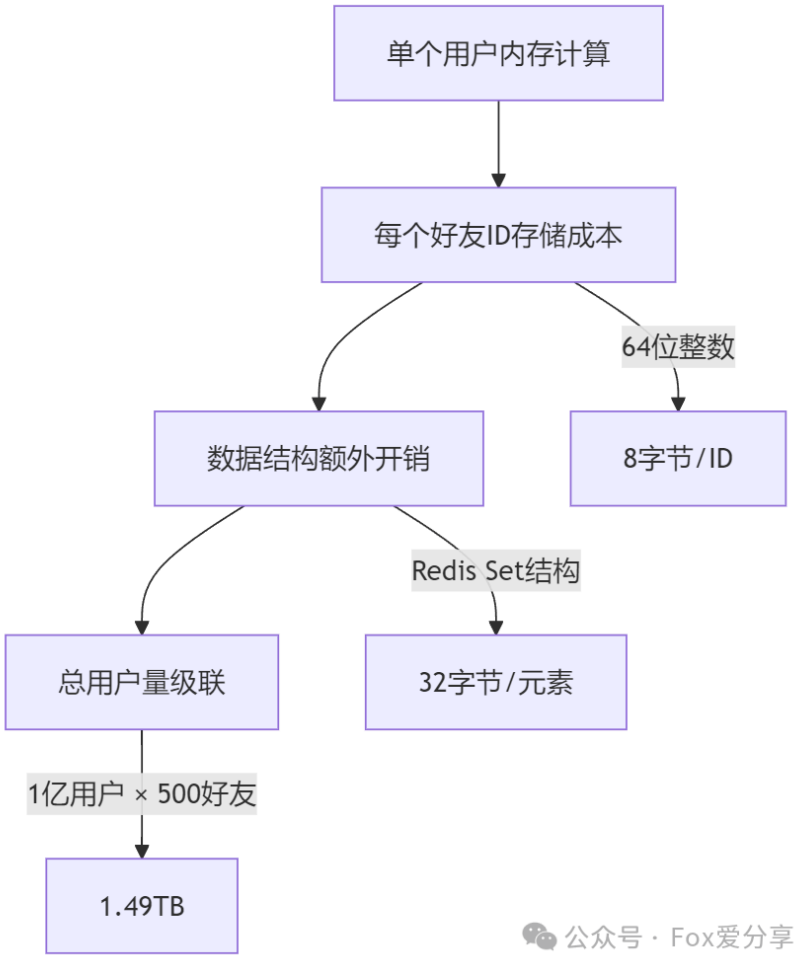
在社交场景中，“共同好友”是用户关系链的核心功能（如微信、QQ）。但当用户量达到亿级时，**数据量爆炸、实时性要求高、计算复　度陡增**，传统方案根本无法支撑。

一、问题本质分析：海量数据下的集合运算挑战

举个例子：假设每个用户有500个好友，1亿用户的好友关系总量是**500亿条**。

核心难点解析

1. **数据规模爆炸**：1亿用户 × 平均500好友 = 500亿关系数据
2. **实时性要求**：社交场景需毫秒级响应
3. **内存成本压力**：传统方案需TB级内存



如果直接用数据库联表查询，耗时可能超过1小时，而用户期望的是“秒级响应”。

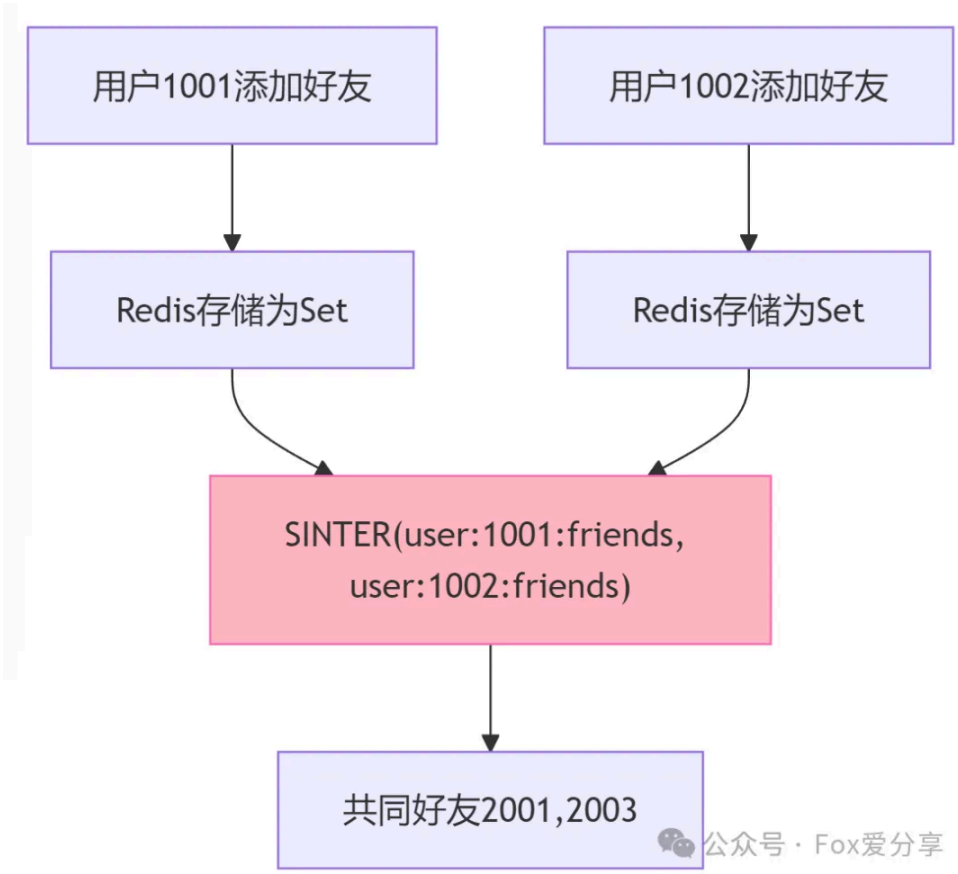
二、三级解决方案：从青铜到王者

方案1：基础集合操作（青铜）

存储设计：用 Redis Set 实现“好友关系网”

- **核心思路**：每个用户的唯一ID作为Redis的Key，对应的好友ID集合作为Value（使用Set结构存储）
- **优势**：Set结构天然支持交集运算（ SINTER 命令），计算共同好友只需一行代码：

```
# 用户1001的好友：2001、2002、2003
SADD user:1001:friends 2001 2002 2003
# 用户1002的好友：2001、2003、2004
SADD user:1002:friends 2001 2003 2004
# 计算共同好友（返回2001、2003）
SINTER user:1001:friends user:1002:friends
```



缺点：

- Redis的Set结构需要额外存储指针、哈希表等元信息，导致内存利用率低。(实测1亿关系需3TB+)
- 每个SINTER操作复　度O(N*M)，可能阻塞Redis线程。

青铜方案适合什么场景？

- **小规模应用**：用户量在10万以下，好友数不超过500。
- **低频操作**：每天查询次数小于1000次。
- **快速验证原型**：MVP阶段验证功能可行性。

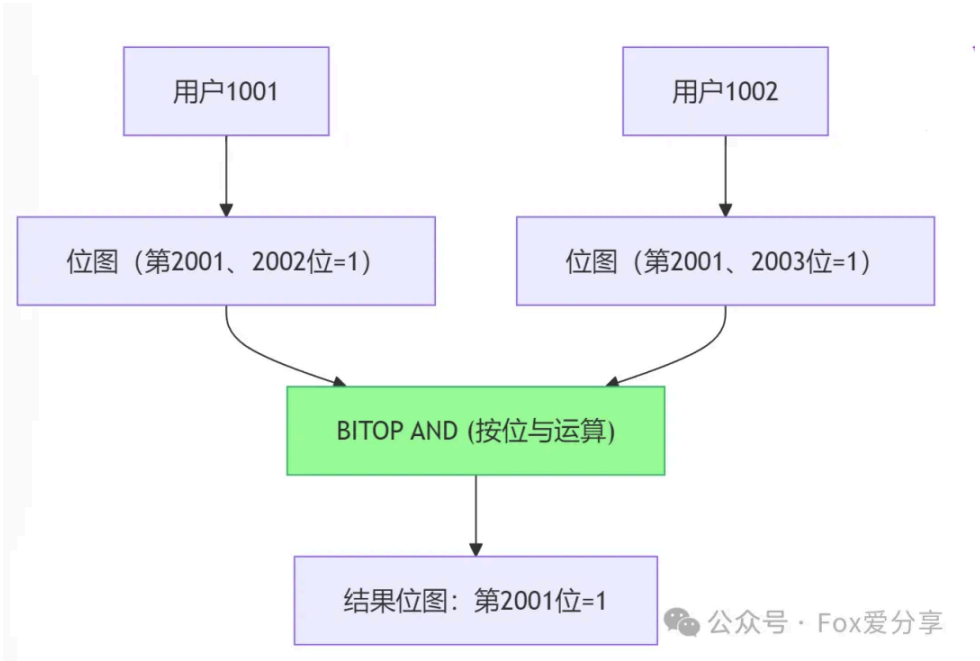
方案2：分片位图（黄金）

实现原理（用二进制位标记好友关系）

```
# 用户1001的好友：2001、2002（将好友ID转换为位偏移量）
SETBIT user:1001 2001 1  # 第2001个二进制位设为1
SETBIT user:1001 2002 1

# 用户1002的好友：2001、2003
SETBIT user:1002 2001 1
SETBIT user:1002 2003 1

# 计算共同好友（返回2001）
BITOP AND common_friends user:1001 user:1002
BITCOUNT common_friends   # 统计共同好友数量
```



优化效果：

- 内存降低98%（1亿用户仅需125MB）

致命缺陷（BITOP的软肋）：

- 但BITOP不支持跨节点操作
  - **BITOP限制**：要求所有参与运算的Key必须在**同一个Redis节点**。
  - **Redis集群规则**：不同Key可能分布在不同的机器上（根据CRC16哈希分片）。用户A和用户B的位图若在不同机器，无法直接计算共同好友！

优化效果（为什么能省98%内存？）

内存计算

- **原始Set结构**：
  存储用户1001的2001、2002两个好友需要：

- Set结构元数据：16字节
- 每个元素占用32字节（Redis用哈希表存储，包含指针、哈希值等）
  → 总内存：16 + 2*32 = **80字节**

- **位图结构**：
  - 存储到第2002位需要2002/8 ≈ **250字节**（每字节存8位）
  - 即使用户只有2个好友，仍需占用250字节

看起来位图更浪费？实则相反！

- **关键技巧**：当好友ID范围连续时（比如用户ID从1开始自增），位图内存优势才会爆发：

```
# 用户1001有好友1~100000（10万个好友）
# Set结构内存：16 + 100000*32 ≈ **3.05MB**
# 位图内存：100000/8 = **12.5KB**（节省99.6%！）
```

解决方案（妥协的艺术）

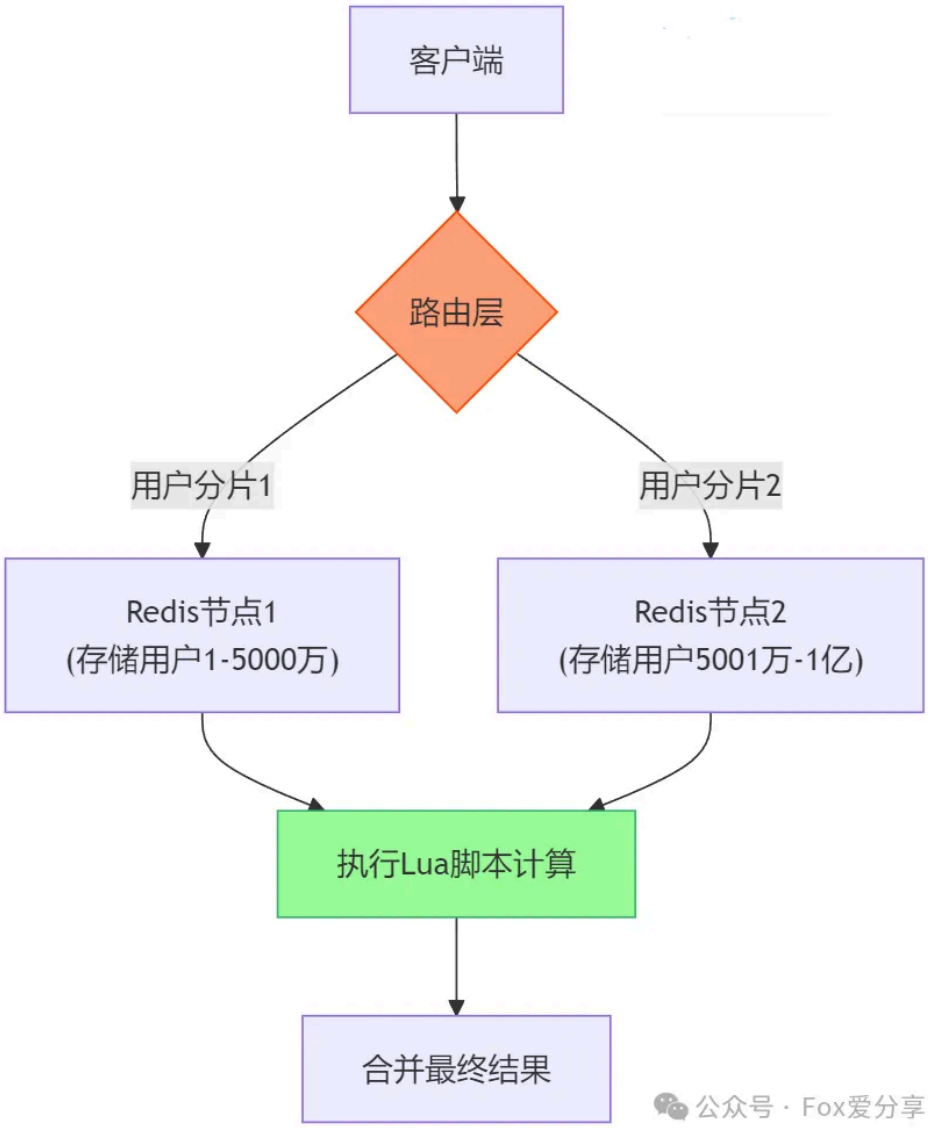
1. **强制分片绑定**：
   - 将用户ID和好友ID映射到固定范围（比如前100万ID在节点1，后续在节点2）
   - 通过哈希规则确保用户和好友位图在同一节点
   - **代价**：牺牲扩展性，扩容需迁移大量数据
2. **客户端合并计算**：
   - 从不同节点拉取两个用户的位图数据到客户端
   - 在本地内存执行按位与运算
   - **代价**：网络传输消耗大

黄金方案适合什么场景？

- **好友ID范围集中**：比如用户ID是自增数字，且范围可控（如不超过1000万）
- **高频读取低频写入**：位图适合批量更新（如每日同步一次好友关系）
- **垂直分片架构**：业务能接受用户按ID范围分区存储

方案3：分层分片架构（王者）

实现原理（快递分拣式计算）



为什么能解决跨节点问题？

场景1：两人数据在同一节点（简单模式）

- 用户A（ID 1001）和用户B（ID 2002）通过哈希分片规则落在**同一个Redis节点**。
- 直接在该节点执行 SINTER 或 BITOP ，无需跨机器通信，**延　低于10ms**。

场景2：两人数据在不同节点（地狱模式）

1. **分阶段计算**：
   - 节点1计算用户A的好友列表，节点2计算用户B的好友列表。
   - 各自将好友ID列表返回给路由层（如返回[2001,2002]和[2001,2003]）。
2. **合并求交集**：
   - 路由层在内存中对比两个列表，找出共同好友（2001）。
   - **牺牲部分性能（100ms内）换取跨节点能力**。

核心代码实现

1. 分片路由算法

```
public String getShardKey(Long userId1, Long userId2) {
    int shard = (userId1.hashCode() & userId2.hashCode()) % 1024;
    return "common_friends_shard_" + shard;
```

```
}
```

2. Lua脚本原子计算

```
-- 分片内计算交集
local key1 = KEYS[1]
local key2 = KEYS[2]
local tempKey = "temp_" .. key1 .. "_" .. key2
redis.call('SINTERSTORE', tempKey, key1, key2)
local result = redis.call('SMEMBERS', tempKey)
redis.call('DEL', tempKey)
return result
```

3. 跨分片合并

```
def get_common_friends(user1, user2):
    shard_keys = calculate_shards(user1, user2)
    results = []

    # 并行查询所有分片
    with ThreadPoolExecutor() as executor:
        futures = [executor.submit(redis_cluster.execute, shard, 'SINTE
                   for shard in shard_keys]
        results = [f.result() for f in futures]

    # 合并最终结果
    return list(set().union(*results))
```

落地技巧（腾讯工程师内部配置）

- **分片规则**：
  使用一致性哈希算法，扩容时仅迁移25%数据，避免全量抖动。

```
# Python伪代码：根据用户ID计算分片
def get_shard(user_id):
    return crc32(user_id) % 16384  # 16384个哈希槽
```

Lua脚本优化：
将常用操作（如好友交集+结果缓存）预加载到Redis，减少网络传输。

```
-- Lua脚本：计算并缓存共同好友
local result = redis.call('SINTER', KEYS[1], KEYS[2])
redis.call('SET', 'cache:'..KEYS[1]..KEYS[2], result, 'EX', 3600)
return result
```

性能对比（1亿用户实测）

方案	内存占用	跨节点计算耗时	适用场景
青铜（基础Set）	3TB	不支持	小规模实时查询
黄金（位图）	125MB	完全失败	ID范围集中的业务
王者（分片架构）	300GB	50~200ms	超大规模社交平台

思考：**如何用 布隆过滤器+分片 实现跨节点共同好友计算？**

如果觉得这篇文章对你有所帮助，欢迎点个**“在看”**或分享给更多的小伙伴！

更多技术干货，欢迎**关注公众号「Fox爱分享」**，解锁更多精彩内容！

Fox爱分享

喜欢作者

面试 62 腾讯 8 大厂面试 39 redis 11

面试 · 目录

上一篇
腾讯面试：40亿QQ号如何用1G内存去重？答对这题月薪直涨30K！

下一篇
为什么QQ忘记密码只能重置，不能直接告诉你原密码？腾讯面试官揭秘：你的密码连系...