参考现代IM消息系统的架构设计做以下改动：

**1.数据类型划分**

原本不做区分，客户端服务端交互的数据走差不多的流程。

现在将数据按功能划分为消息数据和业务管理数据，后续区分处理。

**2.服务器架构**

原本的设计中使用master+worker组成集群，master需要根据事务id进行转发。

现在改为每次用户登录时，master就根据负载均衡算法给用户分配好worker，后续client端不再直接和master交互

master需要监听worker的状态，当worker崩溃，master还需要通知用户进行迁移至其他worker

**3.消息存储方式**

原本消息是以会话为单位进行组织，并且全部存放到redis集群，这样很浪费空间，最终会导致服务器无法负载庞大的内存开销。

消息现在按待接收状态划分为两类：离线消息和历史消息，对应存储位置称为消息同步库（redis）和消息存储库（mysql），采用先存储再同步的策略。

**离线消息：**用户离线时来自其他用户、群组的消息。

扩散写的方式，不单独存放

拉取离线消息属于高频操作，在登录、重连、收到消息序号不连续时都需要重新拉取离线消息，因此消息同步库使用redis集群实现，数据以接收用户为单位进行组织，群聊消息采用需要对过期时间、离线消息最大阈值做出限制，只保留新的数据，其他的直接丢弃当做历史消息处理

**历史消息：**服务器上接收到的来自其他用户、群组的全部消息。

拉取历史消息需要用户端手动发出请求，相对频率较低但历史消息数据量很大，而历史消息的写入效率更加需要重点关注，考虑到后续可能需要支持各种检索匹配，选择使用数据库集群（mysql）存储

这里给出一个存储思路：每个发送者的消息和群聊消息都用单独的表存放，从用户视角来看一次只需要一个会话的历史记录，群聊记录可以直接拉取，一对一聊天的历史记录只需要两张表的检索与合并，两张表需要自动记录插入时间（CURRENT\_TIMESTAMP）或者用全局的消息id做标志（不建议，可能严重影响性能），不然无法还原会话，如果后续要做水平分库还需要数据库服务器时间同步

**4.流程上确保消息有序**

原本的设计消息从服务端到客户端带有旁路，可以免去redis直接发送，但这也可能导致消息乱序，现在统一采用先存储再发送的消息发送方式。

潜在要考虑的问题：多发送方，多接收方，多线程，网络乱序等

由于很难做到各端时间同步或者使用全局消息id标志先后，这里只做两个限制：服务端接收有序、客户端接收有序，管理请求乱序暂时不考虑，直接通知用户失败重试

**服务端接收有序：**为了避免网络等原因导致的发送乱序（这很常见），客户端发送消息时都会携带一个递增的消息id，每次用户登录时客户端发送id和服务端期望id都设置为1

服务端要为每个用户维护一个发送消息列表，按发送消息id排序好待发送消息再处理，实现的细节后面会提到

**客户端接收有序：**redis的用户信息里额外维护一个单调递增的字段，代表该用户下一个接收消息的id，当消息从发送消息列表取出存入redis同步库时，会顺带存入该id

客户端为用户维护一个接收消息列表，负责严格按照id编号接收来自服务端的消息

**5.数据流**

消息数据流：1.client封装消息，通过udp发送给负责该用户的worker

2.worker将消息放到发送消息列表并排序好，如果发送消息id是期望的id，或者发送列表内消息总量已达到阈值，直接跳到第4步

3.worker定期检查所有发送消息列表，离上一次从列表取出消息如果已经过了指定时间间隔，则跳到第4步

4.如果发送消息列表中首个id不是所期望的，则通知发送者，该消息已发送失败，期望id更新为列表里下一个空缺id，上次取出消息时间戳更新为当前时间

5.接着依次取出消息发送列表中的连续消息进行处理，解析目标用户等信息，先存放到消息存储库（mysql），再存放到消息同步库（redis）

6.目标用户如果在线，则同步离线记录给目标用户，用户确认收到后删除消息同步库的记录

管理数据流：1.client封装管理请求，通过udp发送给负责该用户的worker

2.worker识别到是管理类的请求，按照请求内容，更新redis的用户信息、组信息、好友关系等

3.worker将管理请求执行结果返回给client，后续通过gfs日志文件或者数据库对操作和结果进行记录和同步