# 一． 项目设计中的优劣

**优点：**

1. 整个系统松耦合，便于进行扩展和二次开发。

2. 子系统间通过消息队列进行通信，可以屏蔽异构平台的细节，并且消息队列较为可靠，发送方成功发送消息后，消息队列会一直保留消息直到接收方成功接收。

3. 系统可以进行水平扩展并引入了负载均衡器保证了一定的高可用。

4. 在系统中引入了性能监控，并且在负载过高时能向管理员报警。

5. 编码系统中设计了任务队列，较为合理的并发任务数可以使得每个任务被分配足够的系统资源，也保证了每个任务的顺序处理。

**缺点：**

1. 没有缓存机制，重复提交同一请求时会重新加载。

   改进手段：可以使用 redis 进行缓存。

2. 系统中没有索引机制，对于大量视频文件进行读写查找速度较慢。

   改进手段：建立索引机制，使用数据库管理文件。

3. 整个系统过于依赖消息队列，如果消息队列崩溃将导致系统不可用。

   改进手段：可以开启消息持久化，并且多主机部署消息队列服务。

4. 在高并发的情况下下 NFS 性能有限，并且 NFS 对于数据完整性不做校验。

   改进手段：GlusterFS 对于高并发比较友好。

# 二． Bilibili 高可用架构分析

## 一、负载均衡

负载均衡具体分成两个方向，一个是前端负载均衡，另一个是数据中心内部的负载均衡。

前端负载均衡方面，一般而言用户流量访问层面主要依据 DNS，希望做到最小化用户请求延迟。将用户流量最优地分布在多个网络链路上、多个数据中心、多台服务器上，通过动态 CDN 的方案达到最小延迟。

前端服务器的负载均衡主要考虑几个逻辑：

1. 尽量选择最近节点；
2. 基于带宽策略调度选择 API 进入机房；
3. 基于可用服务容量平衡流量。

数据中心内部的负载均衡方面，最忙和最不忙的节点所消耗的 CPU 相差幅度较小。但如果负载均衡没做好，情况可能就像上图左边一样相差甚远。由此可能导致资源调度、编排的困难，无法合理分配容器资源。

数据中心内部负载均衡主要考虑：

1. 均衡流量分发；
2. 可靠识别异常节点；
3. scale-out，增加同质节点以扩容；
4. 减少错误，提高可用性。

负载均衡算法：

1. 选择 backend：CPU，client：health、inflight、latency 作为指标，使用一个简单的线性方程进行打分；

2. 对新启动的节点使用常量惩罚值（penalty），以及使用探针方式最小化放量，进行预热；

3. 打分比较低的节点，避免进入“永久黑名单”而无法恢复，使用统计衰减的方式，让节点指标逐渐恢复到初始状态（即默认值）。

**二、限流**

避免过载，是负载均衡的一个重要目标。随着压力增加，无论负载均衡策略如何高效，系统某个部分总会过载。优先考虑优雅降级，返回低质量的结果，提供有损服务。在最差的情况，妥善的限流来保证服务本身稳定。

限流主要关注以下几点：

1. 针对 qps 的限制，带来请求成本不同、静态阈值难以配置的问题；

2. 根据 API 的重要性，按照优先级丢弃；

3. 给每个用户设置限制，全局过载发生时候，针对某些“异常”进行控制非常关键；

4. 拒绝请求也需要成本；

5. 每个服务都配置限流带来的运维成本。

过载保护方面，核心思路就是在服务过载时，丢弃一定的流量，保证系统临近过载时的峰值流量，以求自保护。常见的做法有基于 CPU、内存使用量来进行流量丢弃；使用队列进行管理；可控延迟算法：CoDel 等。

简单来说，当 CPU 达到 80%的时候，可以认为它接近过载，如果这个时候的吞吐达到 100，瞬时值的请求是 110，就可以丢掉这 10 个流量，这种情况下服务就可以进行过载保护。

**三、重试**

当请求返回错误(例:配额不足、超时、内部错误等)，对于 backend 部分节点过载的情况下，倾向于立刻重试，但是需要留意重试带来的流量放大。

1. 限制重试次数和基于重试分布的策略(重试比率:10%);

2. 只应该在失败的这层进行重试，当重试仍然失败，全局约定错误码“过载，无须重试”，避免级联重试；

3. 随机化、指数型递增的重试周期：ExponentialBackoff + Jitter；

4. 重试速率指标，用于诊断故障。

而在客户端侧，则需要做限速。因为用户总是会频繁尝试去访问一个不可达的服务，因此客户端需要限制请求频次，可以通过接口级别的 error_details，挂载到每个 API 返回的响应里。

**四、超时**

大部分的故障都是因为超时控制不合理导致的。首当其冲的是高并发下的高延迟服务，导致 client 堆积，引发线程阻塞，此时上游流量不断涌入，最终引发故障。

进程内的超时控制，关键要看一个请求在每个阶段（网络请求）开始前，检查是否还有足够的剩余来处理请求。