现象

活动开始的时候页面都很卡。其他接口很慢，或者超时。整个应用都受影响。差不多2分钟后就慢慢恢复了。

找原因

1、查访卡顿的现象的时机和规律。主要是了解用户使用的情况。

2、查看服务器（mysql、应用服务器、redis、消息中间件）的资源历史使用情况。

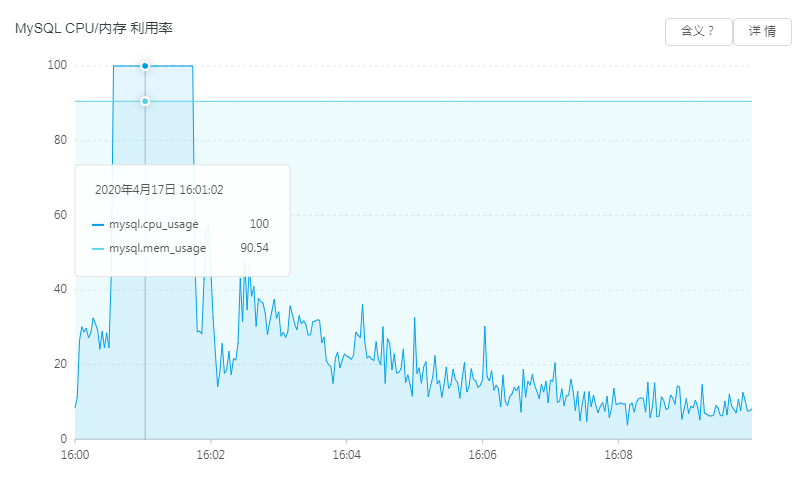
发现情况如下：

1、活动高峰时，qps是平时6倍左右。请求量最多的接口几分钟就有上万的请求量。其他请求量少一点的接口也有几千的访问量。

2、有一台应用服务器的负载达到了26，cpu、内存使用百分比都比较高。这台机器上部署了三个应用，和一个入口nginx。



3、数据库在页面卡的那一小段时间，cpu的使用率到了100%。内存使用率在90%左右。而且cpu使用率曲线和卡顿的曲线完全吻合。在卡顿的那段时间，出现了活动中用到的慢sql语句（语句很简单，数据量也只有3万）。



原因分析及解决方案：

1、活动时页面卡顿，应该是前端静态资源下载太慢导致的。

2、数据库cpu 100%的问题，应该是高并发下，数据库操作导致的。

3、活动时，整个app卡顿，很多页面打不开。应该就是活动时，数据库操作变慢，大并发量导致，大量请求堆积，进而导致负载太高，很多接口响应很慢。数据库cpu 100%，数据库操作也会变慢（由几十毫秒到几秒）

那针对这些问题做了如下优化：

1、修改数据高并发下变慢， 添加索引来解决。

2、数据库链接池增加，增大吞吐量，减少请求堆积

3、非交互数据，用mq做延迟消费。等待活动结束了，才慢慢处理

Map<String, Object> *argsMap* = new HashMap();

*argsMap*.put("x-message-ttl", 600000);  
*argsMap*.put("x-dead-letter-exchange", userpartiExchange);  
*argsMap*.put("x-dead-letter-routing-key", userpartiRoutingKey);

channel.queueDeclare(userpartiDelayQueue, true, false, false, argsMap);

4、mq缓慢消费，减轻mysql压力。

5、将预发数据库从线上数据库实例中分离出来。减轻线上数据库压力，避免影响线上业务。

6、将入口nginx和应用服务分开。分摊nginx这台服务器的压力。

7、添加更多的机器，来运行活动模块，来分摊活动服务的压力。

优化后的效果：

活动不卡了，参与的人数也多了，以前活动配置的红包7000个都抢不完，现在不到2分钟就发完25000个左右的红包。请求量最大的接口qps 大概200多。消息数据也没漏一个。这次又换算是完美。

如果再优化的话，有两点：

1、主要就是活动不要影响其他业务数据库操作了。现在数据库是一个数据库。

2、另一个就是redis序列化工具。以前的客户端用的value序列化工具是默认的序列化工具JdkSerializationRedisSerializer。效率低，数据可读性差。可以使用StringRedisRemplate，然后在用序列化工具处理。可以redis客户端做一层包装