app直播消息群性能优化-阶段性总结

一，直播群框架与目前性能开销

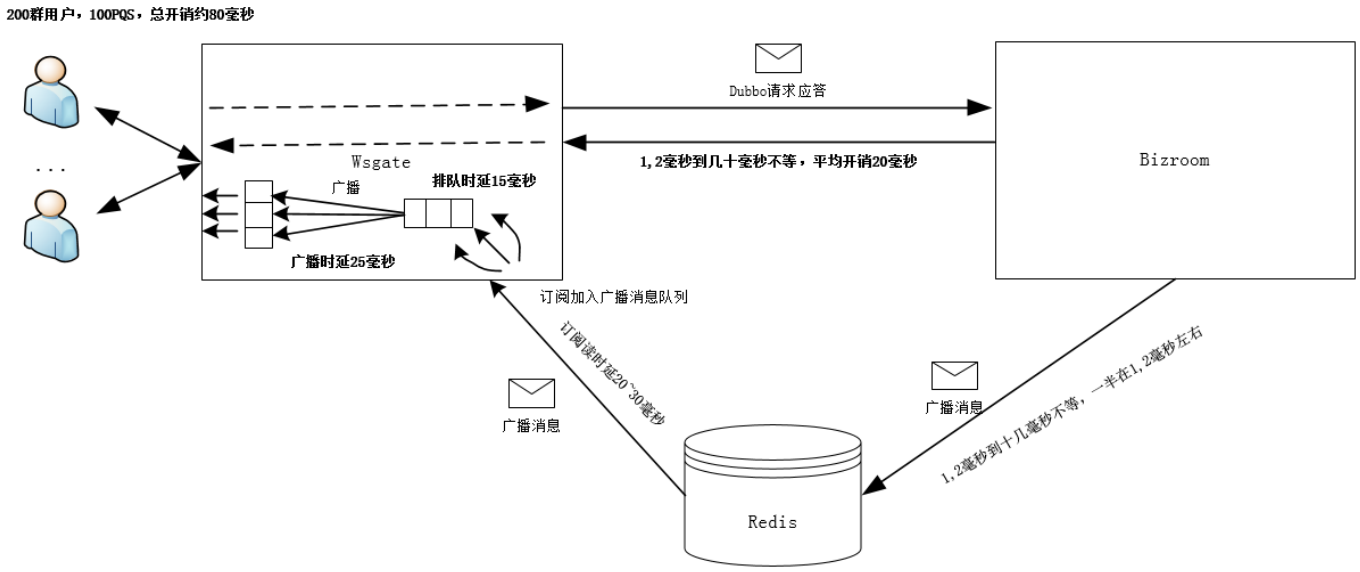

图1

二，在200用户的直播群，100QPS 聊天场景压测，优化结果表

	A	B	C	D	E	F	G
200用户直播群	100QPS 聊天场景	场景	总时延(ms)	业务处理时延	排队时延	广播时延	小结
		有内容检测压测	>500	500	13	22	
		没有内容检测压测	400	400	12	20	内容检测一次http请求大概在100到200毫秒
		优化订阅生产者消费者，异步读写	300	300	13	23	redis消息队列的生成者消费者用异步线程写入，优化100毫秒
		去掉room业务逻辑压测	120	85	13	22	之前的业务逻辑有点复杂，需要近180毫秒
		没有room业务逻辑，优化订阅线程池压测	80	40	15	25	之前订阅线程数量很多，不受控制的增长，而压测时，上下文切换频繁，故而设置了线程池，优化了近40毫秒
200用户直播群	1000QPS 聊天场景	场景	总时延(ms)	业务处理时延	排队时延	广播时延	小结
		没有room业务逻辑，优化订阅线程池（8,20）压测	15k	15K	14	16	和100QPS的比较，CPU占用了从50%到260%，4核接近饱和，4核8G单机可能支持不了1000QPS， 其中压测中dubbo请求开销在几十毫秒，而redis订阅写几毫秒左右，在高并发下，redis订阅读开销是一个最大的影响因素
200用户直播群	1000QPS 聊天场景	场景	总时延(ms)	业务处理时延	排队时延	广播时延	
		没有room业务逻辑，优化订阅线程池（增加线程（16,32））压测	1.5k	1.5K	14	16	

图2

三，对消息广播时延和排队时延压测数据

A	B	C	D	E	F	G	H	I
广播时延压测			100人群	500人群	1000人群	2000人群		
一条消息大概在220B左右	大约5个聚合度，聚合之后的消息体大约1KB	10QPS	13(5)	40(9)	50(5)	180(7)		
	大约15个聚合度，聚合之后的消息体大约3KB	20QPS	15(6)	60(15)	120(16)	260(18)		
	备注：括号后面数字是表示聚合的消息条数，括号前数值单位是毫秒							
排队时延压测			100人群	500人群	1000人群	2000人群		
		10QPS	7(5)	25(9)	40(5)	80(7)		
		20QPS	8(6)	35(15)	60(16)	220(18)		
结论	1，广播时延，因为是遍历把消息拷贝到各自的websocket的发送队列中，主要因素是群人数和消息体大小							
	2，排队时延，和在线人数，和并发数有关系，并发越大，消息队列的抢占需要等待时间，以及在线人数多，并发大，消息广播就慢，就排队时延俞大							
	3，目前有丢包策略，所以在高并发下，广播时延和排队时延不会过分的突激，在平常使用场景下，基本稳定							

图3

1000用户直播群 10QPS 总时延=room业务处理逻辑（11ms）+排队时延（35ms）+广播时延（56ms，聚合8条消息）=102ms，其中dubbo调用平均在5毫秒

1000用户直播群 20QPS 总时延=room业务处理逻辑（15ms）+排队时延（33ms）+广播时延（86ms，聚合18条消息）=133ms，其中dubbo调用平均在9毫秒

消息少，room业务中dubbo调用开销小，redis订阅读写开销小，使聚合的消息体反而大，加上群人数多，使广播时延增大，反促使排队时延增大，和之前单独压测广播时延和排队时延的数据，图3，相当

四，优化过程

4.1 性能开销组成：总时延=room处理时延+wsgate本地消息队列排队时延+群广播时延=dubbo调用时延+业务逻辑时延+redis订阅写时延+redis订阅读时延+排队时延+广播时延

4.2

项目编码统计每个步骤的开销，并配置influxdb+grafana，用来可视化监测统计每一条消息的开销，如下图：是包含http请求，去文本检测逻辑的 ，总时延>500毫秒

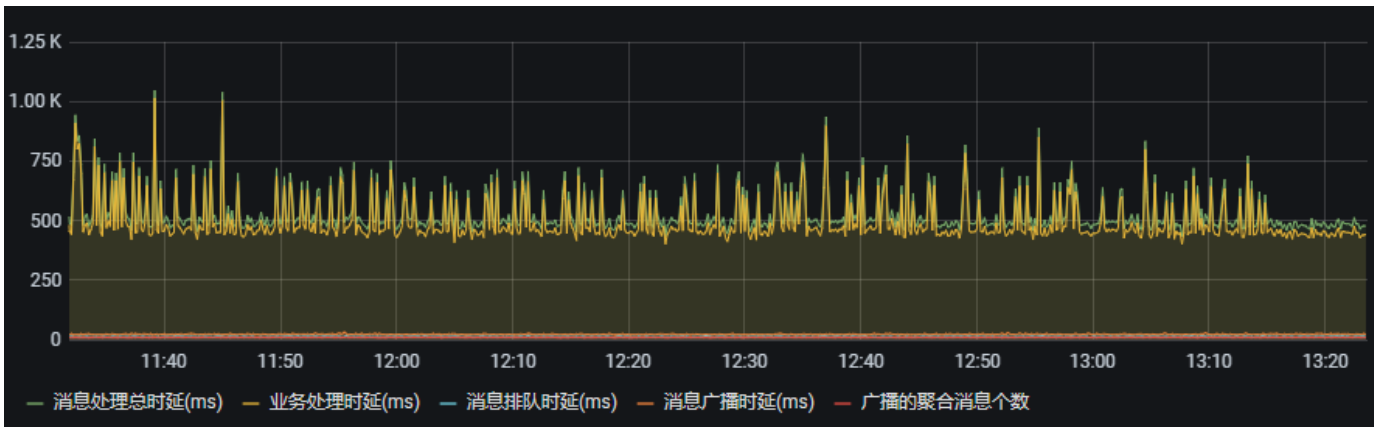

图4

4.3

去掉文本检测逻辑：总时延400左右，但是　room处理时延=近400毫秒，排队时延=12毫秒左右，广播时延=20毫秒左右，可以看到占比较大是room处理时延（=room dubbo调用时延（约270）+redis订阅读写逻辑时延）

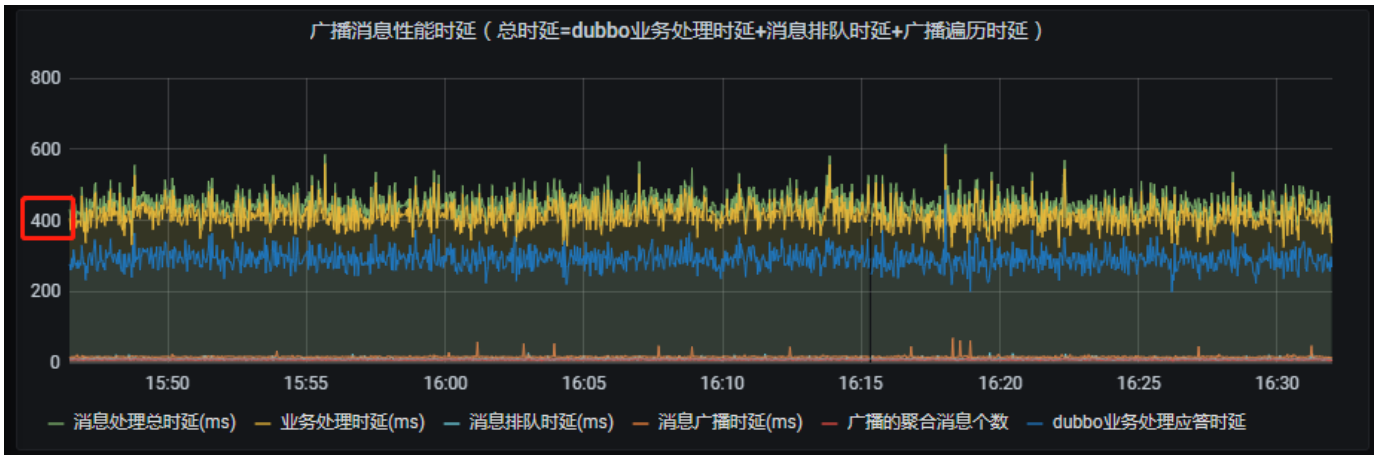

图5

4.4　因为redis订阅读写在两个不同的服务器，不好计算时间开销，由图5中的统计可以看到，redis订阅时延=room处理时延-room dubbo调用时延=约130毫秒，故而优化redis订阅逻辑

1）ntpdate cn.pool.ntp.org 同步两服务器时间，就算多次同步，总会有毫秒级偏差，导致计算redis读时间-redis写时间会出现负数

优化逻辑：1，设置redis订阅写和读线程池，2，redis订阅读写，使用异步操作，3，优化核心代码（主要指redis订阅写的代码，为了精准把消息发送到相应的wsgate上，需要多次读redis+1次写redis）

优化结果：图5和图6比较，大概优化近100毫秒

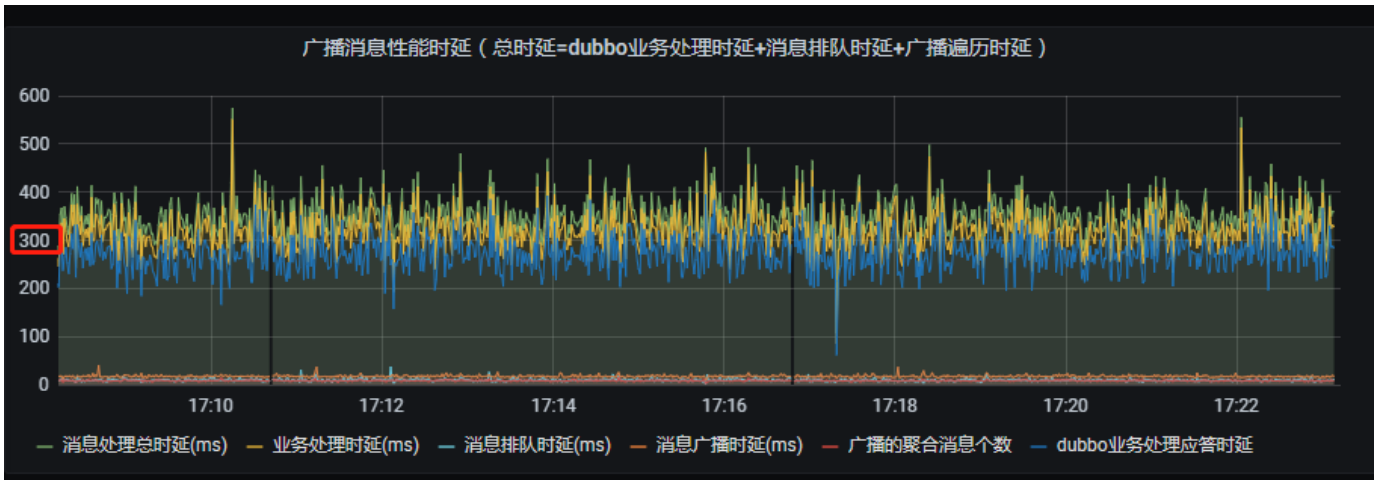

图6

4.5 room dubbo调用现在是开销最大，统计room 业务逻辑的开销，发现room业务逻辑开销就需要近200毫秒：room
dubbo调用=rpc请求+rpc应答+room业务逻辑，故而暂时去掉room业务逻辑压测，并建议优化room业务逻辑，使以后接入room业务也不会太慢

因为没有保留图片，去掉room业务逻辑，目前总开销大概在120毫秒左右，见图2中的数据

4.6 优化wsgate redis订阅读的线程，之前wsgate
redis订阅读的线程是不受控制的增长，设置线程池之后，优化近40毫秒，目前中时延80毫秒左右

4.6.1 分析java线程相关信息

1）jstack pid >stack_pid.log

不完全统计线程数：

wsgate ：

1，http-nio-8081-exec-： 200个左右
用户数200，猜测应该是每个websocket一个线程，需要待确定，优化思路，就是看是否可以较简单的设置线程池，这个是tomcat的默认配置200线程，10000链接
2，WsClientSendThreadPool：20
3，DubboClientHandler：一般20个，变化的值，根据dubbo线程模型，消费者线程池是cachedThreadPool，会自动视情况生成新的线程，在并发高的时候，可能需要注意
4，NettyClientWorker：5个，dubbo默认是 核数+1

5，redis相关线程：lettuce-eventExecutorLoop 4个 lettuce-epollEventLoop 2个
6，async-broadcast-：5个，将订阅得到的数据，异步添加到群广播中
7，container：40个，60个之类的不等，看栈调用，猜测用于订阅
8，GC ：4个
9，C2 CompilerThread ：2个 这俩有事会突然cpu占比到30%

bizroom：

1，DubboServerHandler ：200个左右，大多数waiting状态
2，NettyServerWorker 5个
3，redis相关线程：lettuce-eventExecutorLoop 4个 lettuce-epollEventLoo 4个
4，async-redis-publish：5个，用来异步向redis写
5，GC ：4个
6，C2 CompilerThread ：2个

并且绝大多数是wait或time_wait状态

2） 看函数开销，看主要在什么地方有开销：strace -o out.log -f -t -c -e trace=all -p 或者perf top

a) wsgate:主要是系统调用，上下文切换，epoll

.1 ，没有测试的时候：

```
% time seconds usecs/call calls errors syscall
------ ----------- ----------- --------- --------- ----------------
89.59 222.036224 762 291265 139270 futex
8.78 21.750498 16756 1298 epoll_wait
1.43 3.538377 153842 23 21 restart_syscall
```

.2，测试的时候：

```
% time seconds usecs/call calls errors syscall
------ ----------- ----------- --------- --------- ----------------
96.88 28560.702591 155482 183691 49742 futex
1.48 436.672515 1850307 236 29 restart_syscall
0.74 219.429350 87006 2522 epoll_wait
```

b)bizroom：主要是系统调用，上下文切换，epoll

1，没有测试：

```
% time seconds usecs/call calls errors syscall
------ ----------- ----------- --------- --------- ----------------
76.93 327.758214 1471 222838 108638 futex
22.23 94.726276 26980 3511 epoll_wait
```

2，有测试：

```
% time seconds usecs/call calls errors syscall
------ ----------- ----------- --------- --------- ----------------
86.36 3559.934047 4061 876603 197934 futex
10.84 446.722457 15898 28099 epoll_wait
1.42 58.486081 2123 27546 recvfrom
1.14 46.934455 1066692 44 23 restart_syscall
```

概要设置：

```
//配置核心线程数
executor.setCorePoolSize(16);
//配置最大线程数
```

```
executor.setMaxPoolSize(32);
//配置队列大小
executor.setQueueCapacity(1000);

redisMessageListenerContainer.setTaskExecutor(executor);
```

3）看上下文切换：pidstat -w 1 -p 或者 cat /proc/stat | grep ctxt && sleep 30 && cat /proc/stat | grep ctxt 每30秒种的上下文切换次数，dstat（推荐），vmstat 1 （推荐）

wsgate：一压测，从一两千到1万多，并且多空闲

备注：cs: 每秒上下文切换数。id: 空闲时间百分比

测试时：

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
2 0 0 202768 44604 1015052 0 0 0 184 5912 8320 6 10 83 1 0
0 0 0 199780 44604 1015212 0 0 0 300 7549 10285 10 13 76 1 0
5 0 0 197932 44616 1015196 0 0 0 328 8319 10859 10 15 74 1 0
```

没有测试时：

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
4 0 0 356264 44260 990848 0 0 1 6 0 0 1 1 97 0 0
0 0 0 356124 44260 990880 0 0 0 6 881 1879 1 1 99 0 0
0 0 0 355860 44260 990892 0 0 0 50 1064 2109 1 1 98 1 0
0 0 0 355860 44260 990920 0 0 0 128 790 1716 1 1 90 9 0
0 0 0 355860 44260 990932 0 0 0 0 800 1772 1 0 99 0 0
0 0 0 355876 44260 990948 0 0 0 50 844 1872 1 1 98 0 0
```

所以设置redis订阅读的线程池，避免不受控制的增长，减少可能的线程切换

4.7 目前总时延=80ms ，但是 room dubbo调用=rpc请求+rpc应答=还是20毫秒左右，并且存在进程下线程0running，1，需要优化dubbo调用性能，2，需要继续优化redis订阅相关的性能

1）top ,uptime都可以看负载，压测时负载高

wsgate：

没测试：

20:34:59 up 1047 days, 10:07, 7 users, load average: 0.14, 0.70, 2.65

测试：

20:15:18 up 1047 days, 9:48, 7 users, load average: 3.76, 5.94, 5.03

2）查看硬件cpu利用率情况：mpstat -P ALL 1　CPU利用率都不高，每个核在压测时都只有百分之十几二十几 ，大多空闲，还可以　top命令下，按1，则可以展示出服务器有多少CPU，及每个CPU的使用情况

压测时：

07:31:40 PM CPU %usr %nice %sys %iowait %irq %soft %steal %guest %gnice %idle
07:31:42 PM all 8.99 0.00 12.71 0.39 0.00 0.90 0.00 0.00 0.00 77.02
07:31:42 PM 0 6.74 0.00 19.17 1.04 0.00 3.63 0.00 0.00 0.00 69.43
07:31:42 PM 1 5.18 0.00 9.84 0.00 0.00 0.00 0.00 0.00 0.00 84.97
07:31:42 PM 2 17.77 0.00 12.69 0.51 0.00 0.00 0.00 0.00 0.00 69.04
07:31:42 PM 3 5.61 0.00 9.18 0.00 0.00 0.00 0.00 0.00 0.00 85.20

3）看io：iostat -kx 1，把写日志调高，看着磁盘io都不高，wsgate主要是网络io型程序

4）top 看线程 ： top -Hp　pid：cpu，内存，磁盘io（iostat）都利用率不高，但是负载高，怀疑是网卡io不足，导致性能慢

测试：cpu利用率不高，内存利用率不高，负载高，但是有进程下有线程0running

top - 20:09:14 up 1047 days, 9:41, 6 users, load average: 3.39, 3.44, 3.89 负载高
Threads: 231 total, 0 running, 231 sleeping, 0 stopped, 0 zombie
%Cpu(s): 8.6 us, 10.4 sy, 0.0 ni, 78.6 id, 1.3 wa, 0.0 hi, 1.0 si, 0.1 st
KiB Mem : 8010968 total, 222676 free, 6805944 used, 982348 buff/cache
KiB Swap: 0 total, 0 free, 0 used. 592156 avail Mem

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
32501 root 20 0 6001396 1.1g 14196 S 6.6 14.6 1:03.04 java
498 root 20 0 6001396 1.1g 14196 S 4.0 14.6 1:04.96 java
32487 root 20 0 6001396 1.1g 14196 S 3.3 14.6 1:06.25 java
32027 root 20 0 6001396 1.1g 14196 S 3.0 14.6 0:13.08 java
31997 root 20 0 6001396 1.1g 14196 S 1.3 14.6 1:17.00 java
32023 root 20 0 6001396 1.1g 14196 S 1.3 14.6 1:35.66 java

没有测试：

procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
r b swpd free buff cache si so bi bo in cs us sy id wa st
4 0 0 356264 44260 990848 0 0 1 6 0 0 1 1 97 0 0
0 0 0 356124 44260 990880 0 0 0 6 881 1879 1 1 99 0 0
0 0 0 355860 44260 990892 0 0 0 50 1064 2109 1 1 98 1 0
0 0 0 355860 44260 990920 0 0 0 128 790 1716 1 1 90 9 0
0 0 0 355860 44260 990932 0 0 0 0 800 1772 1 0 99 0 0
0 0 0 355876 44260 990948 0 0 0 50 844 1872 1 1 98 0 0

5）看网络io：nicstat -M -i eth0 1 （推荐），iftop， nethogs

wsgate ：压测时网络io：写网卡带宽20到30多Mb每秒，但是%Util为0，不知道是软件不对，还是占比太小，这个有去查网卡能支持的带宽，无果，所以不好下结论

压测时：

```
Time Int rMbps wMbps rPk/s wPk/s rAvs wAvs %Util Sat
19:46:02 eth0 1.58 14.93 1535.6 1698.5 135.3 1151.8 0.00 0.00
19:46:03 eth0 1.65 21.40 1895.6 2498.5 114.4 1122.5 0.00 0.00
```

没有压测时

```
Time Int rMbps wMbps rPk/s wPk/s rAvs wAvs %Util Sat
11:23:25 eth0 0.04 0.07 44.23 40.75 132.5 226.9 0.00 0.00
11:23:26 eth0 0.13 0.07 103.9 88.88 161.8 103.3 0.00 0.00
```

bizroom：压测时网络io不大

压测时：没保存数据，印象中大概 rMbps wMbps 0点几，到1点几

没有压测时：

```
Time Int rMbps wMbps rPk/s wPk/s rAvs wAvs %Util Sat
11:19:05 eth0 0.02 0.02 23.00 20.00 137.8 132.2 0.00 0.00
11:19:06 eth0 0.02 0.04 28.00 27.00 88.64 217.7 0.00 0.00
```

4.8 想优化dubbo性能，设置TCP连接，1条改成8条，没有效果

因为看bizroom的网络io，读写带宽大约1.2Mb/s,一条TCP足够了，压测结果也确实显示，没有改善时延

4.9 想优化redis订阅读的链接，无果，可能方法不对

```
# Lettuce
# 连接池最大连接数
spring.redis.lettuce.pool.max-active=8
spring.redis.lettuce.pool.max-wait=-1
spring.redis.lettuce.pool.max-idle=8
spring.redis.lettuce.pool.min-idle=0
```

```
lsof -p pid -nP|grep TCP
```

和redis的TCP链接，总是只2条，而同样设置bizroom却有9条TCP链接

4.10 jvm优化，似有优化近10毫秒，并且开销监测图更平稳，没有大的波动

设置jvm参数： nohup java -server -Xms1200m -Xmx1200m -Xmn400m -XX:SurvivorRatio=8 -Xss512k -XX:+UseConcMarkSweepGC -XX:+UseParNewGC -XX:+UseCMSCompactAtFullCollection -XX:CMSFullGCsBeforeCompaction=4 -XX:+CMSScavengeBeforeRemark -XX:+ScavengeBeforeFullGC -XX:+CMSParallelInitialMarkEnabled -XX:+DisableExplicitGC -XX:+PrintGCDateStamps -XX:+PrintGCDetails -verbose:gc -Xloggc:./logs/gc_%p_%t.log -XX:ErrorFile=./logs/hs_err_%p.log -XX:HeapDumpPath=./logs/heap_dump_%p.hprof -XX:+HeapDumpOnOutOfMemoryError -Dcom.sun.management.jmxremote.authenticate=true -Dcom.sun.management.jmxremote.ssl=false -XX:+CMSIncrementalMode -jar $bin --spring.profiles.active=test

目前总时延大约=70ms=room处理时延30ms+排队时延18毫秒+广播时延22毫秒，且总体稳定，波动抖动幅度小

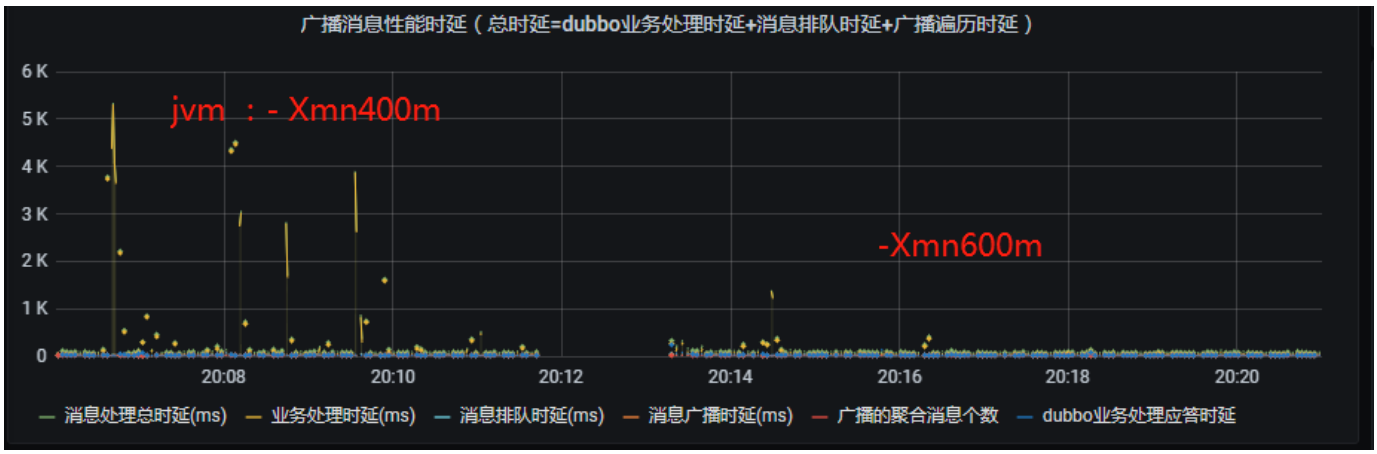

图7

1）jstat -gcutil 2566 1000

400 大概每8秒一次YGC ，频繁YGC，开销 大约16毫秒左右
600 大概每14秒一次YGC，适当YGC，开销 25毫秒左右
800 大概每18秒一次YGC，相对不频繁YGC，开销 35毫秒左右

2）看gc日志

-Xmn400 配置时：
2020-09-29T19:46:13.987+0800: 538.028: [GC (Allocation Failure) 2020-09-29T19:46:13.987+0800: 538.029: [ParNew: 334531K->7370K(368640K), 0.0169297 secs] 399033K->72235K(1187840K) icms_dc=0 , 0.0173285 secs] [Times: user=0.05 sys=0.00, real=0.02 secs]
2020-09-29T19:46:22.057+0800: 546.098: [GC (Allocation Failure) 2020-09-29T19:46:22.057+0800: 546.098: [ParNew: 335050K->9079K(368640K), 0.0176832 secs] 399915K->74223K(1187840K) icms_dc=0 , 0.0180419 secs] [Times: user=0.06 sys=0.00, real=0.02 secs]

（334531K-7370K）-（399033K-72235K）=363KB 需要拷贝到老生代，需要增加新生代空间，故而改为600m

还可以用http://gceasy.io/ 去分析gc日志

4.11 目前总体现象是，cpu利用率不高，内存利用率不高，磁盘io不高，存在线程0 running，dubbo调用一次平均在20毫秒左右，redis订阅读性能是一个较大影响因素（当并发1000的时候，订阅读可以到1秒2秒 ），有时间，依然需要继续优化dubbo和redis订阅逻辑

4,12　可能需要改进或注意的地方

1，redis订阅模式，存在固有的问题，订阅key，默认容量大约8MB（需要看源码或者官方资料核实，或调整），当出现生成的快，消费的慢的时候，可能redis会丢数据，这样不太能支持大并发，不过目前咱们的当量应该可以满足

2，对群广播模式需要再做一些了解，比如之前公司分享的reactor模式，目前咱们的notify模式，存在排队开销 约15毫秒，随着并发越大，这个时间也会越大

3，dubbo的高性能使用，以及redis订阅逻辑需要优化，不过dubbo性能还算相对稳定，redis订阅的性能会随着并发的增大而下降