Kafka高性能解密及优化

闫燕飞
(flyyfyan)

基础架构部-高级工程师

SPEAKER

闫燕飞

基础架构部-高级工程师

Apache Kafka得益于其分布式、高性能、低延迟、数据有序等等特性，被越来越多的流式计算组件选作核心数据管道及存储。当前腾讯云上部署的kafka日处理消息已经达数万亿，在对kafka的长期持续运营中，腾讯中间件团队结合业务实际情况对kafka部分技术细节进行优化，将资源利用率发挥到了极致。在次我们重点对kafka高性能的关键点以及腾讯云kafka团队对其进行的优化点进行分享。

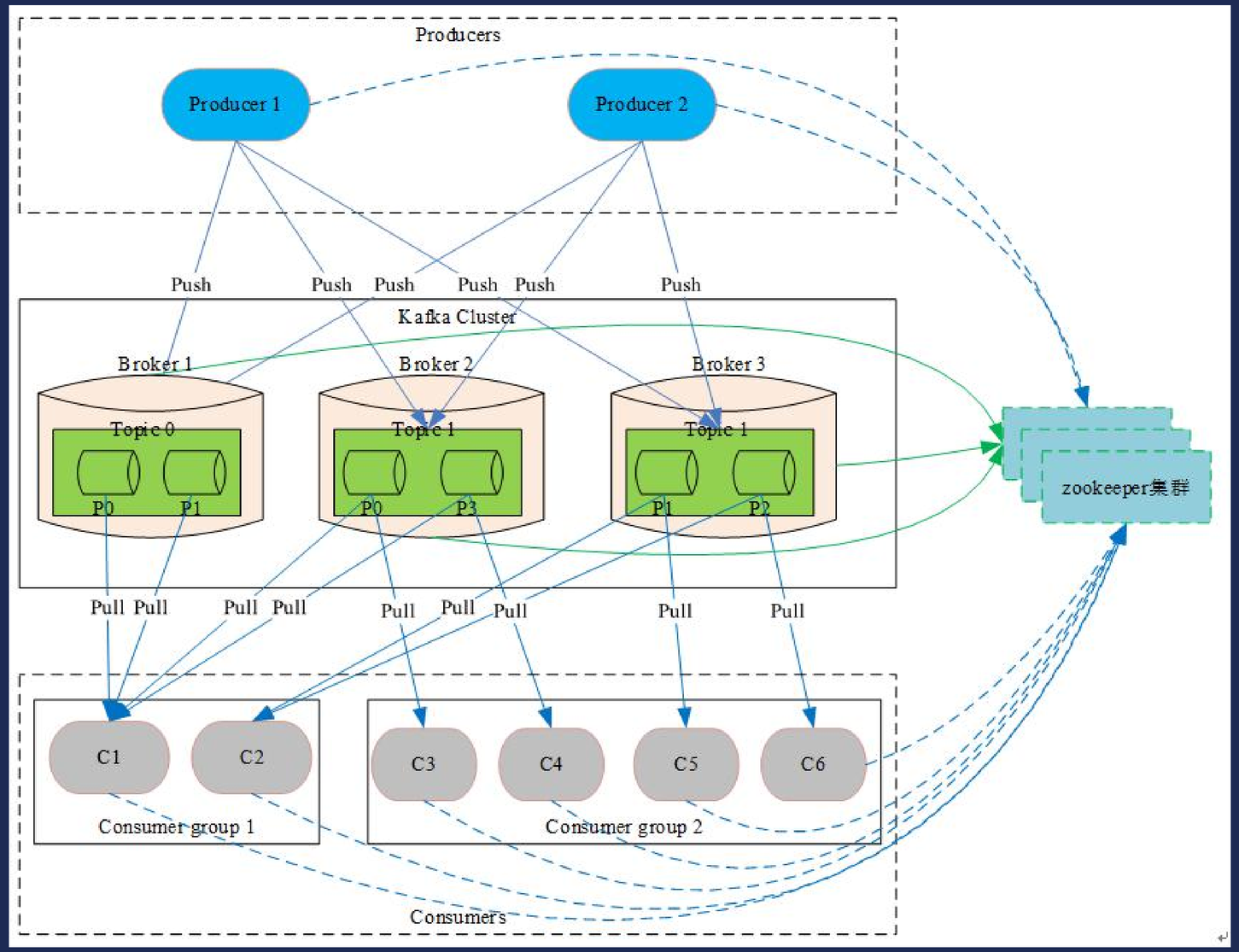
内容提纲

- Kafka-高性能揭秘
  - 架构
  - 存储
  - 消息格式
  - 关键点小结
  - 性能数据
- Kafka-性能优化
  - 当前架构剖析
  - 锁优化
  - 文件刷盘优化
  - gc优化
- 未来展望

Kafka高性能揭秘

- 架构
- 存储
- 消息格式
- 关键点小结
- 性能数据

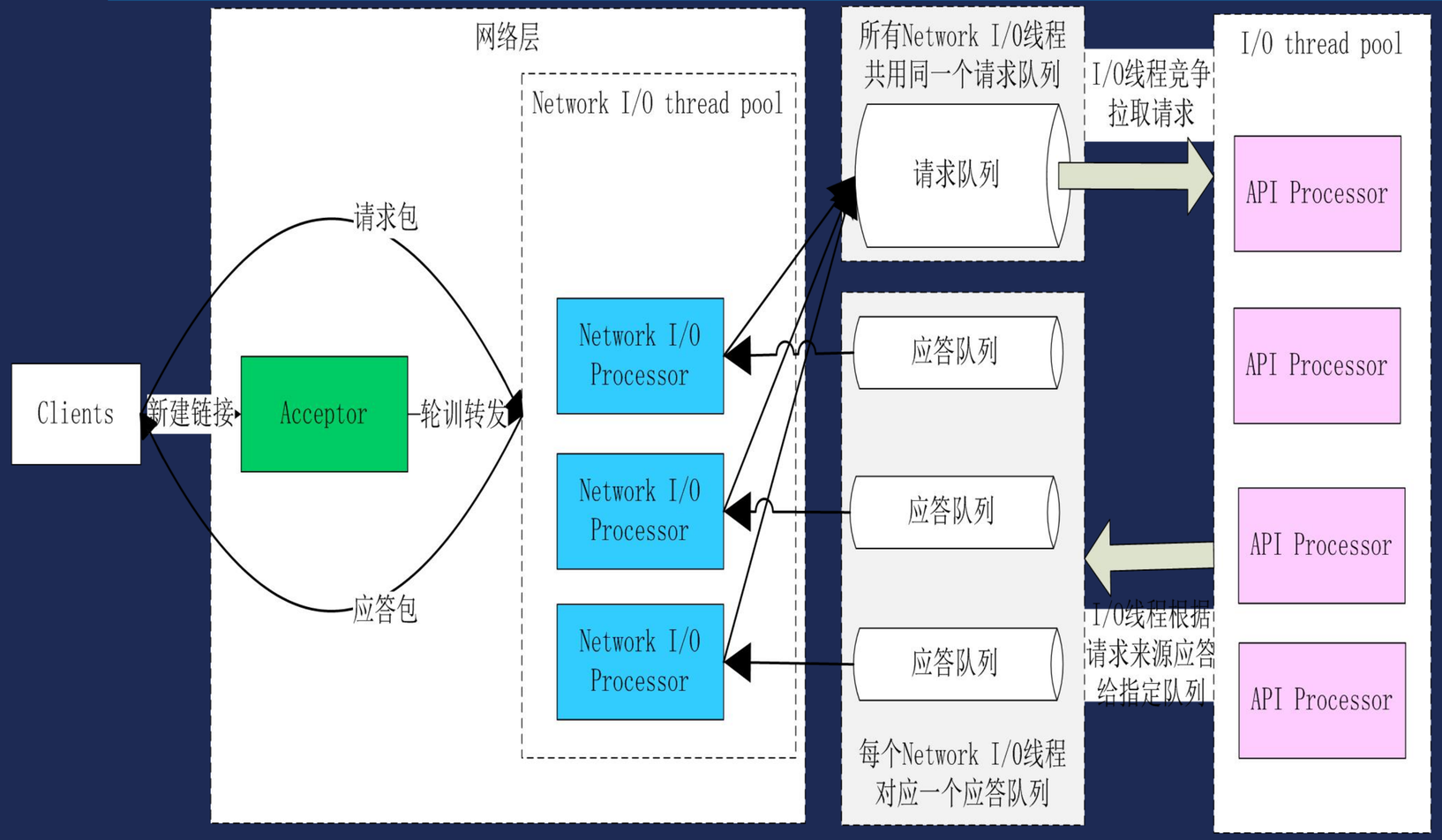
Kafka高性能揭秘 之 总体架构



名词解释

- Broker: kafka服务器
- Topic: 消息类别
- Partition: 物理上的概念，一个Topic可以包含多个partition
- Offset:消息在partition的唯一序号
- Producer:生产者,负责发布消息
- Consumer:消费者，负责消费消息
- Consumer group:消费者分组,消费者标签，用于将消费者分类
- Zookeeper集群:存储meta数据、leader选举、故障容错等

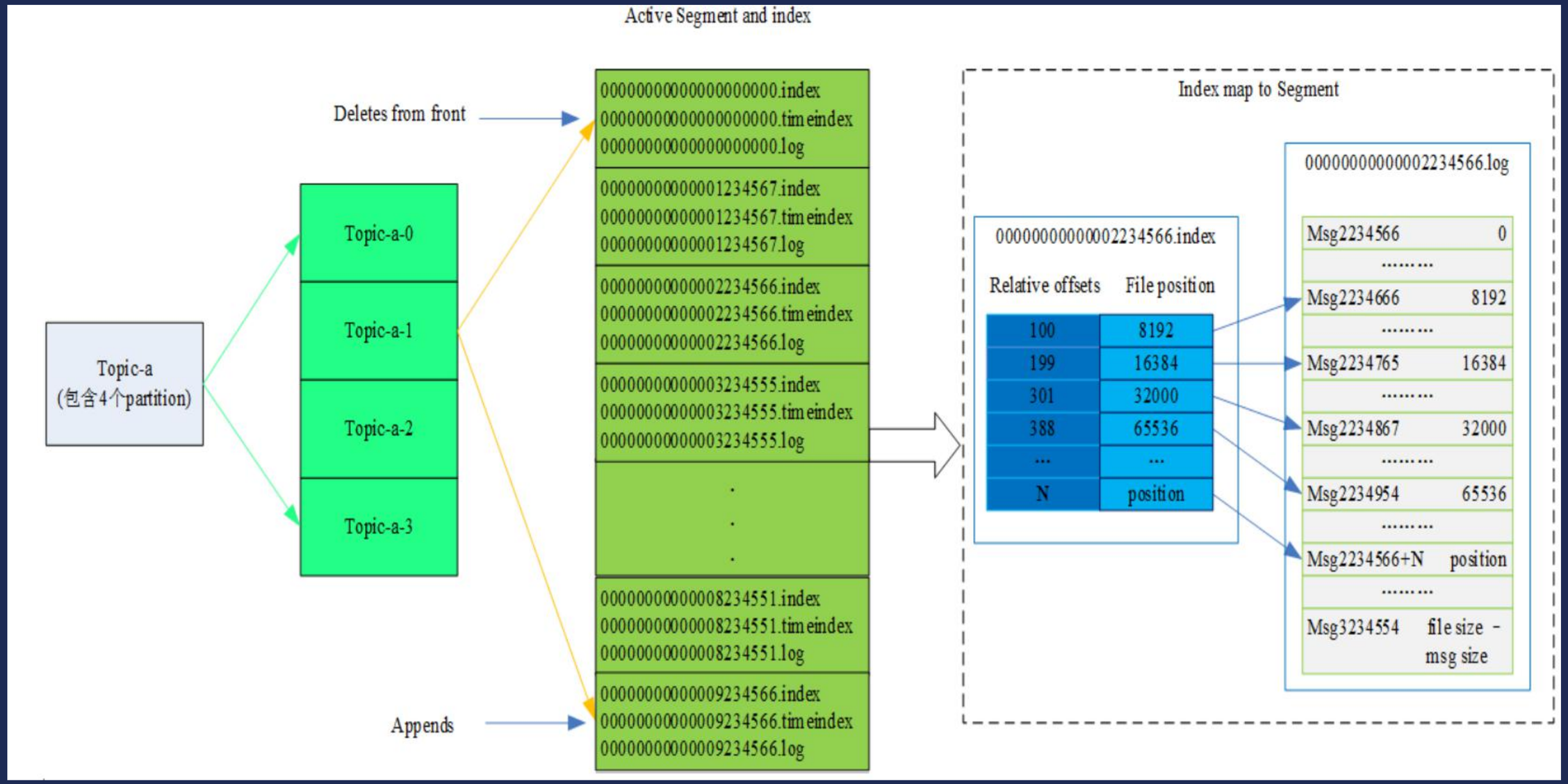
Kafka高性能揭秘 之 Broker架构



Kafka-Broker架构

- 采用多线程，
  可充分利用多核优势
- 采用队列，
  解耦网络处理及I/O处理线程

Kafka高性能揭秘 之 存储



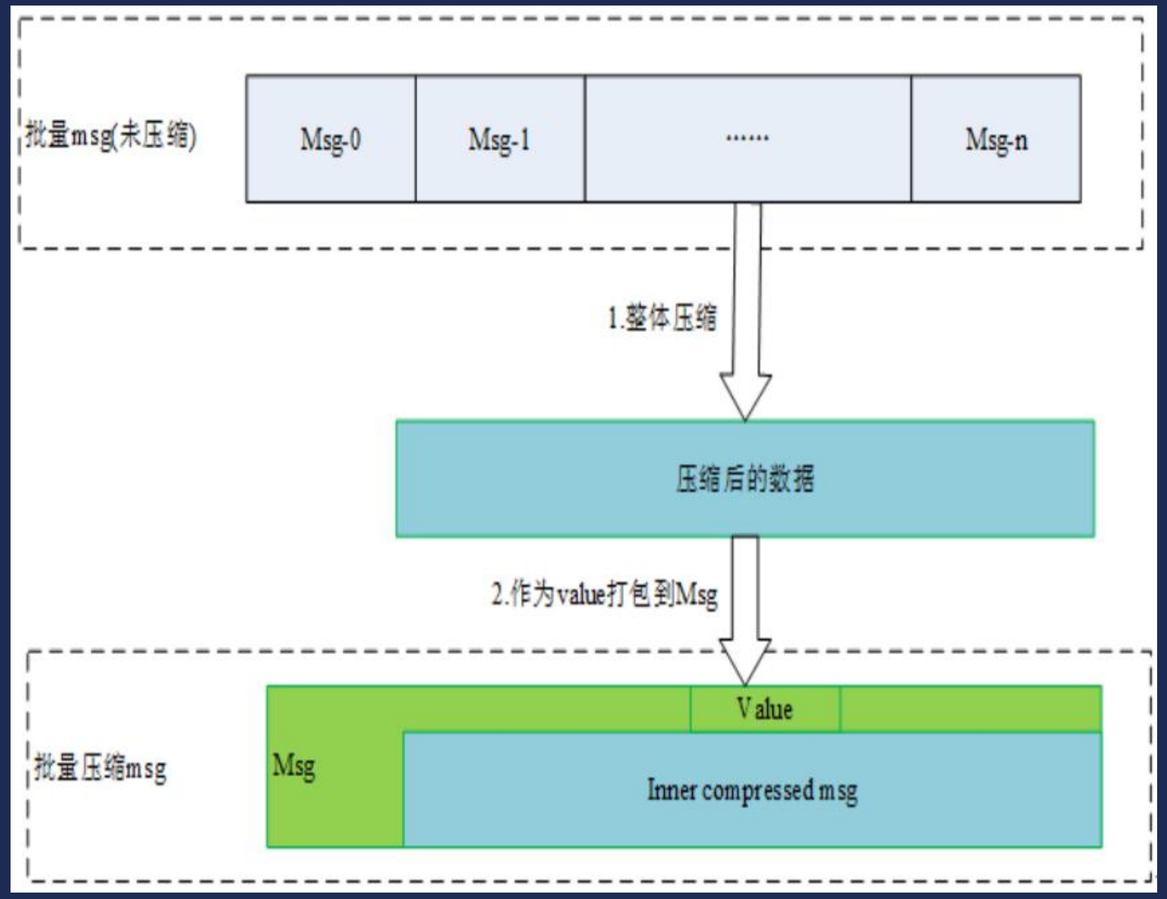
Kafka高性能揭秘 之 消息格式

```
struct kafka_bytes
{
    int32_t        len;
    char           data[0];
};
typedef struct kafka_bytes* kafka_bytes_t;
```

```
/*supported since 0.10.0*/
struct kafka_msghdr_v1
{
    int64_t              offset;
    int32_t              size;
    int32_t              crc;
    int8_t               magic;
    int8_t               attributes;
    int64_t              timestamp;

    struct kafka_bytes   key[0];

    /*varabile len can't have two*/
#if 0
    struct kafka_bytes   value[0];
#endif
};
typedef struct kafka_msghdr_v1* kafka_msghdr_v1_t;
```



- 二进制编码, 且客户端,broker,包括文件存储中格式保持一致
- 通过消息嵌套的方式，实现了消息的批量压缩提高了整体压缩率的同时，保证了格式的一致

Kafka高性能揭秘 之 关键点小结

- Topic进行分区(partition)，从而实现水平扩展
- 批量处理
- 基于磁盘大文件的顺序读写。
- Message采用简单二进制编码, 且客户端,broker,包括文件存储中消息格式一致，省去了转码开销
- 端到端的数据压缩，并且支持批量压缩。
- 简单高效的传输协议及序列化方式，采用了单字节对齐，网络字节序的二进制格式进行通信，协议编解码效率极高，资源消耗少。
- 充分利用系统的Page Cache，而不采用应用内存缓存
- 支持多Disk Drive

Kafka高性能揭秘 之 性能数据

- 测试场景：1个topic 多个partition 单机测试
- 机型信息：32核CPU 64G内存 12*2T SATA盘(软Raid10) 万兆网卡
- kafka版本0.10.2.0 每10w条消息或2秒，进行刷盘。140个客户端模拟一定的并发

partition个数	大小(byte)	消息qps	吞吐	cpu使用率	I/O使用率
2	64	96.2w/s	90MB/s	240%	27%
	128	90.6w/s	140MB/s	209%	33%
	256	68.3w/s	189MB/s	234%	36%
	512	47.4w/s	247MB/s	240%	42%
	1024	27.1w/s	274MB/s	316%	39%
4	64	102.7w/s	96MB/s	298%	29%
	128	103.5w/s	160MB/s	212%	33%
	256	76.6w/s	212MB/s	236%	37%
	512	58.0w/s	302MB/s	250%	45%
	1024	40.2w/s	406MB/s	325%	45%
8	64	113.4w/s	106MB/s	435%	33%
	128	111.9w/s	173MB/s	272%	37%
	256	79.5w/s	220MB/s	330%	41%
	512	65.4w/s	341MB/s	461%	49%
	1024	41.1w/s	415MB/s	504%	54%
16	64	123.0w/s	115MB/s	378%	39%
	128	97.0w/s	150MB/s	292%	41%
	256	86.7w/s	240MB/s	427%	52%
	512	58.5w/s	305MB/s	406%	57%
	1024	41.0w/s	414MB/s	512%	62%

- Kafka展现了较高的性能数据，可以达到较高的qps及磁盘吞吐
- 随着partition数目的增多(在一定范围内)，性能会有所提高
- 随着数据包的增大(一定范围内)，性能提高
- CPU及磁盘使用率较低，还有一定的优化空间

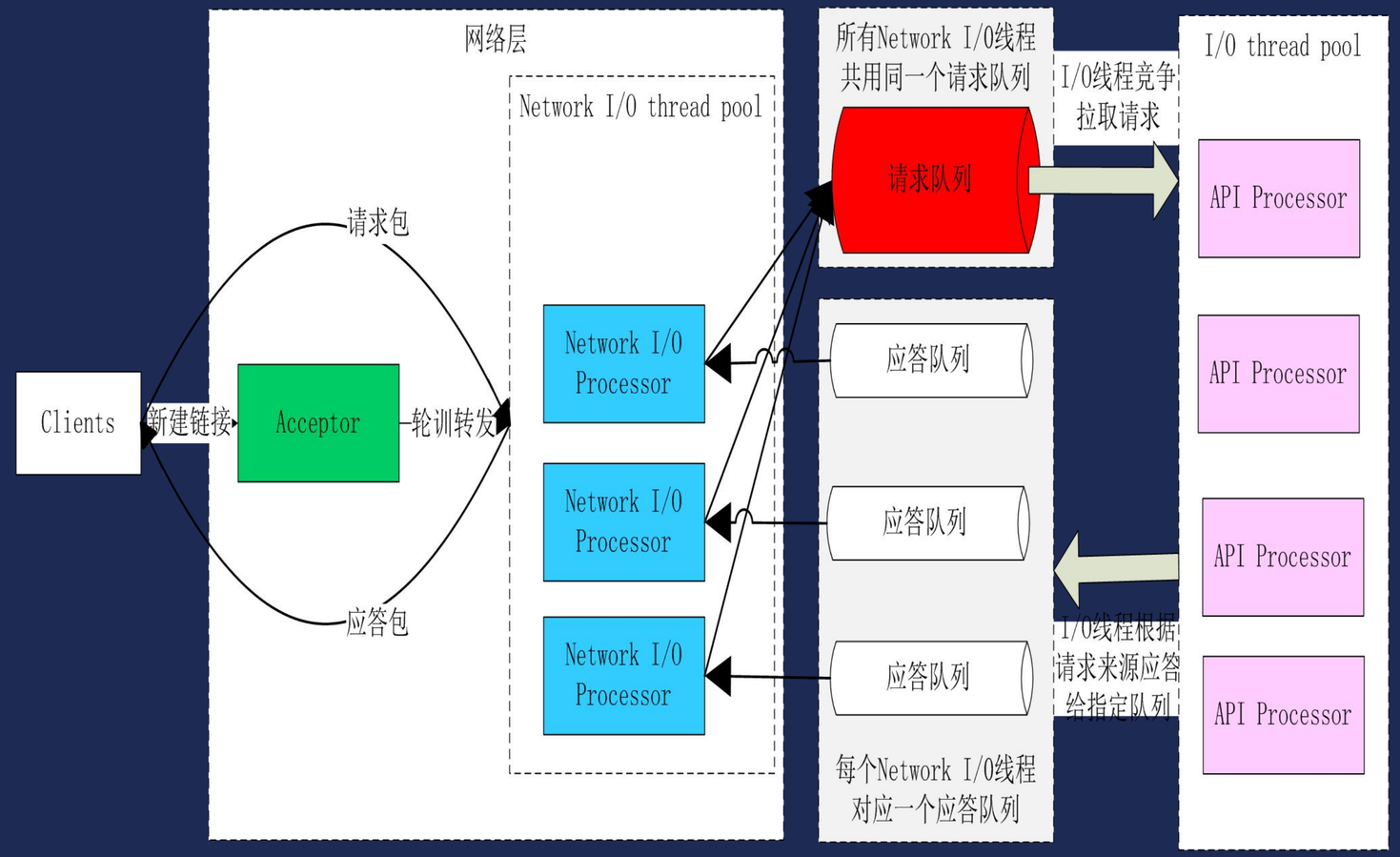
内容提纲

- Kafka-高性能揭秘
  - 架构
  - 存储
  - 消息格式
  - 关键点小结
  - 性能数据
- Kafka-性能优化
  - 当前架构剖析
  - 锁优化
  - 文件刷盘优化
  - gc优化
- 未来展望

Kafka性能优化

- 当前架构剖析
- 锁优化
- 文件刷盘优化
- gc优化

Kafka性能优化 之 当前架构剖析

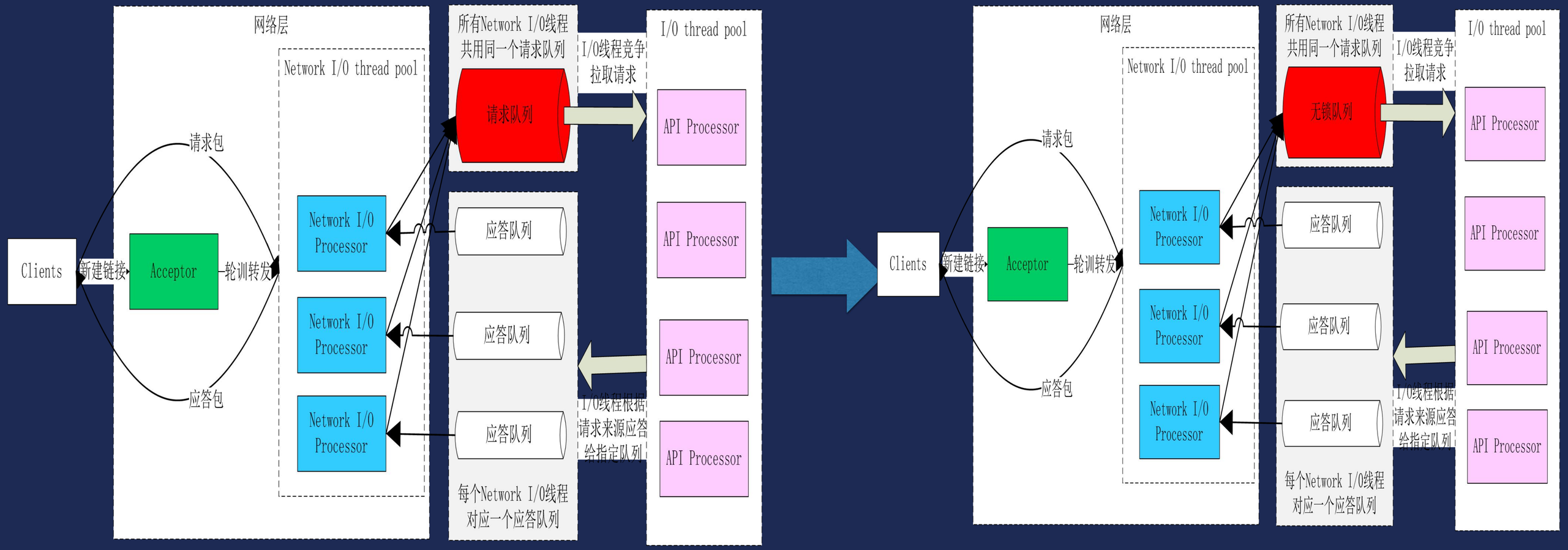


Kafka-Broker架构

瓶颈点猜想及分析

- 只有一个全局请求队列，且未做无锁处理，可能导致竞争过于激烈导致无法充分利用多线程优势
- 直接在I/O处理线程中进行磁盘flush容易堵塞整个系统
- 处理生产请求时会产生大量的消息对象，可能会导致gc成为性能瓶颈

Kafka性能优化 之 锁优化



Kafka-Broker架构

无锁队列-Broker架构

Kafka性能优化 之 锁优化结果

partition个数	大小(byte)	原生			lock-free Queue		
		吞吐	cpu	I/O	吞吐	cpu	I/O
2	64	90MB/s	240%	27%	93MB/s	150%	32%
	128	140MB/s	209%	33%	141MB/s	174%	35%
	256	189MB/s	234%	36%	206MB/s	192%	40%
	512	247MB/s	240%	42%	250MB/s	227%	44%
	1024	274MB/s	316%	39%	290MB/s	258%	47%
4	64	96MB/s	298%	29%	118MB/s	244%	34%
	128	160MB/s	212%	33%	165MB/s	279%	37%
	256	212MB/s	236%	37%	220MB/s	243%	40%
	512	302MB/s	250%	45%	305MB/s	277%	47%
	1024	406MB/s	325%	45%	405MB/s	356%	50%
8	64	106MB/s	435%	33%	120MB/s	213%	34%
	128	173MB/s	272%	37%	170MB/s	241%	39%
	256	220MB/s	330%	41%	222MB/s	468%	46%
	512	341MB/s	461%	49%	350MB/s	258%	53%
	1024	415MB/s	504%	54%	414MB/s	450%	55%
16	64	115MB/s	378%	39%	117MB/s	274%	34%
	128	150MB/s	292%	41%	151MB/s	310%	39%
	256	240MB/s	427%	52%	236MB/s	647%	46%
	512	305MB/s	406%	57%	310MB/s	445%	56%
	1024	414MB/s	512%	62%	414MB/s	482%	60%

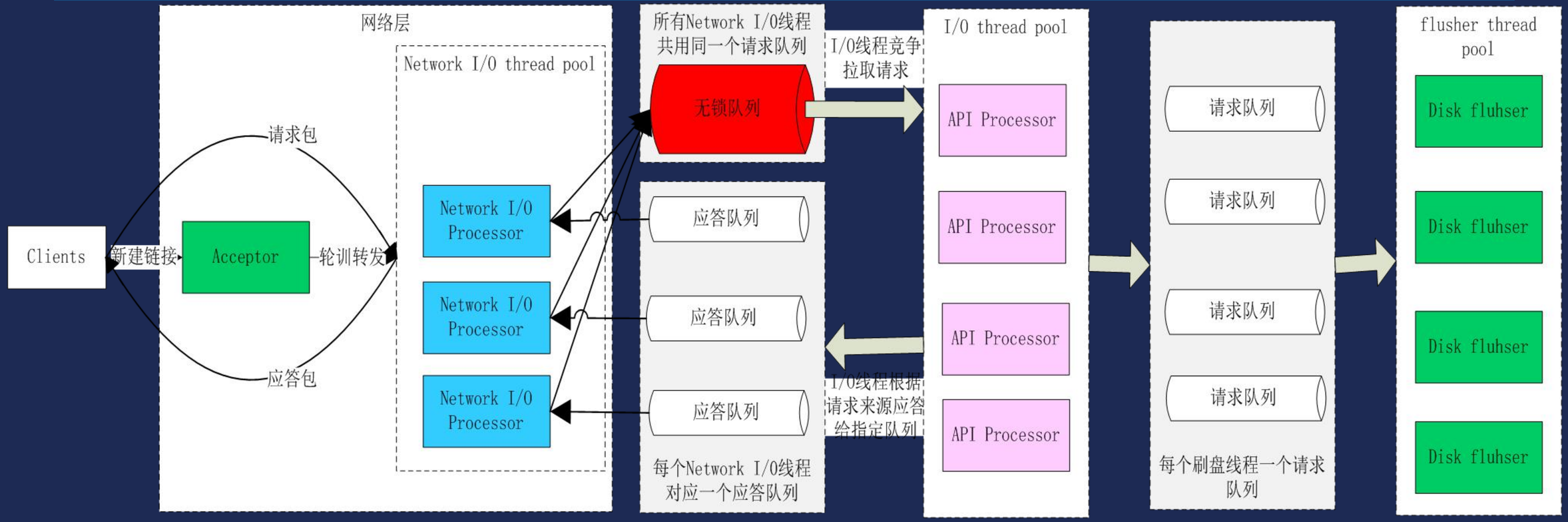
性能基本一致，优化结果不明显!!! Why？？？

Kafka性能优化 之 锁优化-结果分析

partition个数	大小(byte)	原生				lock-free Queue			
		吞吐	cpu	I/O	请求qps	吞吐	cpu	请求qps	I/O
2	64	90MB/s	240%	27%	1.01w/s	93MB/s	150%	1.36w/s	32%
	128	140MB/s	209%	33%	2.02w/s	141MB/s	174%	2.01w/s	35%
	256	189MB/s	234%	36%	2.74w/s	206MB/s	192%	2.76w/s	40%
	512	247MB/s	240%	42%	3.12w/s	250MB/s	227%	3.72w/s	44%
	1024	274MB/s	316%	39%	4.84w/s	290MB/s	258%	4.80w/s	47%
4	64	96MB/s	298%	29%	1.18ws/	118MB/s	244%	1.65w/s	34%
	128	160MB/s	212%	33%	2.40w/s	165MB/s	279%	2.32w/s	37%
	256	212MB/s	236%	37%	2.76w/s	220MB/s	243%	2.78w/s	40%
	512	302MB/s	250%	45%	3.01w/s	305MB/s	277%	3.36w/s	47%
	1024	406MB/s	325%	45%	3.11w/s	405MB/s	356%	3.30w/s	50%
8	64	106MB/s	435%	33%	1.75w/s	120MB/s	213%	1.83w/s	34%
	128	173MB/s	272%	37%	2.84w/s	170MB/s	241%	2.82w/s	39%
	256	220MB/s	330%	41%	2.93w/s	222MB/s	468%	2.92w/s	46%
	512	341MB/s	461%	49%	3.40w/s	350MB/s	258%	3.86w/s	53%
	1024	415MB/s	504%	54%	3.96w/s	414MB/s	450%	4.00w/s	55%
16	64	115MB/s	378%	39%	1.78w/s	117MB/s	274%	1.68w/s	34%
	128	150MB/s	292%	41%	2.07w/s	151MB/s	310%	2.07w/s	39%
	256	240MB/s	427%	52%	2.34w/s	236MB/s	647%	2.60w/s	46%
	512	305MB/s	406%	57%	3.78w/s	310MB/s	445%	3.92w/s	56%
	1024	414MB/s	512%	62%	4.08w/s	414MB/s	482%	4.16w/s	60%

- kafka采用批量的方式，使得请求量急剧减少，最大qps也不超过10w/s，故即使在仅有一个有锁的请求队列的情况下，对怎个系统的性能影响较小

Kafka性能优化 之 异步刷盘1



异步刷盘-Broker架构

- 增加一组刷盘线程，专门负责磁盘flusher，防止堵塞核心流程

Kafka性能优化 之 异步刷盘2

partition个数	大小(byte)	原生			lock-free Queue			async		
		吞吐	cpu	I/O	吞吐	cpu	I/O	吞吐	cpu	I/O
2	64	90MB/s	240%	27%	95MB/s	150%	32%	530MB/s	650%	76%
	128	140MB/s	209%	33%	141MB/s	174%	35%	635MB/s	686%	75%
	256	189MB/s	234%	36%	206MB/s	192%	40%	758MB/s	744%	81%
	512	247MB/s	240%	42%	250MB/s	227%	44%	763MB/s	774%	89%
	1024	274MB/s	316%	39%	290MB/s	258%	47%	750MB/s	807%	80%
4	64	96MB/s	298%	29%	118MB/s	244%	34%	690MB/s	1321%	90%
	128	160MB/s	212%	33%	165MB/s	279%	37%	786MB/s	1421%	94%
	256	212MB/s	236%	37%	220MB/s	243%	40%	766MB/s	1217%	92%
	512	302MB/s	250%	45%	305MB/s	277%	47%	770MB/s	1457%	93%
	1024	406MB/s	325%	45%	405MB/s	356%	50%	722MB/s	1251%	90%
8	64	106MB/s	435%	33%	120MB/s	213%	34%	720MB/s	1821%	96%
	128	173MB/s	272%	37%	170MB/s	241%	39%	768MB/s	2198%	93%
	256	220MB/s	330%	41%	222MB/s	468%	46%	776MB/s	2224%	92%
	512	341MB/s	461%	49%	350MB/s	258%	53%	715MB/s	1961%	92%
	1024	415MB/s	504%	54%	414MB/s	450%	55%	750MB/s	1194%	95%
16	64	115MB/s	378%	39%	117MB/s	274%	38%	800MB/s	1949%	96%
	128	150MB/s	292%	41%	151MB/s	310%	42%	701MB/s	2517%	92%
	256	240MB/s	427%	52%	236MB/s	647%	50%	703MB/s	2359%	95%
	512	305MB/s	406%	57%	310MB/s	445%	56%	705MB/s	2421%	92%
	1024	414MB/s	512%	62%	414MB/s	482%	60%	696MB/s	2380%	93%

- 小包情况下有 4 ~ 5倍的性能提升，大包情况下也有1倍左右的性能提升
- 随着包大小的增加，异步刷盘方式性能的提升度呈下降趋势
- 随着partition的增多，异步刷盘方式性能提升度有一定的下降趋势
- 磁盘I/O已达较高的使用率，基本维持在90%以上(注:一个partition的情况会导致无法并行写入,故磁盘I/O有所降低)。

Kafka性能优化 之 gc优化

partition个数	大小(byte)	async				gc			
		吞吐	cpu	I/O	gc	吞吐	cpu	I/O	gc
2	64	530MB/s	650%	76%	2.39%	620MB/s	851%	84%	1.01%
	128	635MB/s	686%	75%	2.10%	770MB/s	859%	81%	1.08%
	256	758MB/s	744%	81%	1.76%	770MB/s	861%	90%	1.01%
	512	763MB/s	774%	89%	1.38%	770MB/s	864%	90%	0.98%
	1024	750MB/s	807%	80%	1.22%	758MB/s	855%	85%	0.97%
4	64	690MB/s	1321%	90%	3.29%	700MB/s	939%	93%	1.18%
	128	786MB/s	1421%	94%	2.35%	800MB/s	1320%	98%	1.48%
	256	766MB/s	1217%	92%	2.08%	766MB/s	1100%	93%	0.95%
	512	770MB/s	1457%	93%	1.39%	770MB/s	1335%	86%	1.00%
	1024	722MB/s	1251%	89%	1.55%	722MB/s	1000%	92%	0.99%
8	64	720MB/s	1821%	96%	4.73%	778MB/s	2029%	93%	1.88%
	128	768MB/s	1898%	88%	3.35%	788MB/s	2032%	93%	1.86%
	256	776MB/s	1924%	96%	2.31%	782MB/s	2050%	92%	1.86%
	512	715MB/s	1961%	92%	2.20%	726MB/s	2048%	90%	1.85%
	1024	750MB/s	1194%	95%	1.57%	809MB/s	1187%	92%	1.17%
16	64	800MB/s	1949%	96%	9.10%	800MB/s	1468%	97%	2.41%
	128	701MB/s	2517%	92%	3.89%	705MB/s	2454%	95%	2.24%
	256	703MB/s	2359%	95%	3.24%	710MB/s	2132%	96%	2.14%
	512	705MB/s	2421%	92%	2.85%	710MB/s	2359%	94%	2.18%
	1024	696MB/s	2380%	93%	2.30%	695MB/s	1754%	97%	1.71%

- gc优化达到一定效果，可以看到优化后整个gc耗时占比已经低于2.5%，较未优化的情况有1.5% ~ 7.0%的提升
- 在吞吐量差不多的情况下，优化gc的版本有更小的cpu使用率，平均cpu使用率低5% ~ 10%
- 受限于磁盘I/O的瓶颈，可以看到优化gc相对于异步刷盘版本，总体来说未有较大的吞吐性能提升

Kafka性能优化 最终结果对比

partition个数	大小(byte)	原生			gc		
		吞吐	cpu	I/O	吞吐	cpu	I/O
2	64	90MB/s	240%	27%	620MB/s	851%	84%
	128	140MB/s	209%	33%	770MB/s	859%	81%
	256	189MB/s	234%	36%	770MB/s	861%	90%
	512	247MB/s	240%	42%	770MB/s	864%	90%
	1024	274MB/s	316%	39%	758MB/s	855%	85%
4	64	96MB/s	298%	29%	700MB/s	939%	93%
	128	160MB/s	212%	33%	800MB/s	1320%	98%
	256	212MB/s	236%	37%	766MB/s	1100%	93%
	512	302MB/s	250%	45%	770MB/s	1335%	86%
	1024	406MB/s	325%	45%	722MB/s	1000%	92%
8	64	106MB/s	435%	33%	778MB/s	2029%	93%
	128	173MB/s	272%	37%	788MB/s	2032%	93%
	256	220MB/s	330%	41%	782MB/s	2050%	92%
	512	341MB/s	461%	49%	726MB/s	2048%	90%
	1024	415MB/s	504%	54%	809MB/s	1187%	92%
16	64	115MB/s	378%	39%	800MB/s	1468%	97%
	128	150MB/s	292%	41%	705MB/s	2454%	95%
	256	240MB/s	427%	52%	710MB/s	2132%	96%
	512	305MB/s	406%	57%	710MB/s	2359%	94%
	1024	414MB/s	512%	62%	695MB/s	1754%	97%

- 小包情况下有 4 ~ 5倍的性能提升，大包情况下也有1倍左右的性能提升
- 随着包大小的增加，异步刷盘方式性能的提升度呈下降趋势
- 随着partition的增多，异步刷盘方式性能提升度有一定的下降趋势
- 磁盘I/O已达较高的使用率，基本维持在90%以上(注:一个partition的情况会导致无法并行写入,故磁盘I/O有所降低)。

未来展望

- Kafka-当前消费还无法使用pipe方式消费，导致出现以下几个问题：
  - 消费者性能非常依赖于与broker之间的网络延迟，跨城区调用性能异常低下，限制了kafka使用的场景
  - 副本复制同样走的是消费的逻辑，同样无法Pipe导致不太可能做跨区域部署，并且在压力较大时，比较容易导致副本拉取无法跟上生产速度

- Kafka-当前为性能等方面考虑，不支持低版本消费者消费保存的高版本消息，而整个kafka发展到现在已经形成了三个版本的消息格式，导致整个系统的兼容性不是那么的友好，是否可以考虑放开相关的支持？

Thanks!

联合主办方：腾讯云 | 开源中国 oschina.net | APACHE kafka A distributed streaming platform

直播支持：腾讯课堂 KE.QQ.COM 学习成就梦想