链滴

Kafka 异步消息也会阻塞？记一次 Dubbo 频繁超时排查过程

作者：9526xu

原文链接：https://ld246.com/article/1570536844924

来源网站：链滴

线上某服务 A 调用服务 B 接口完成一次交易，一次晚上的生产变更之后，系统监控发现服务 B 接口繁超时，后续甚至返回线程池耗尽错误 Thread pool is EXHAUSTED。因为服务 B 依赖外部接口，开始误以为外部接口延时导致，所以临时增加服务 B dubbo 线程池线程数量。配置变更之后，重启务，服务恢复正常。一段时间之后，服务 B 再次返回线程池耗尽错误。这次深入排查问题之后，才发 Kafka 异步发送消息阻塞了 dubbo 线程，从而导致调用超时。

一、问题分析

Dubbo 2.6.5，Kafak maven 0.8.0-beta1

服务 A 调用服务 B，收到如下错误：

2019-08-30 09:14:52,311 WARN method [%f [DUBBO] Thread pool is EXHAUSTED! Thread Name: DubboServerHandler-xxxx, Pool Size: 1000 (active: 1000, core: 1000, max: 1000, largest: 100), Task: 6491 (completed: 5491), Executor status:(isShutdown:false, isTerminated:false, isTerinating:false), in dubbo://xxxx!, dubbo version: 2.6.0, current host: 127.0.0.1

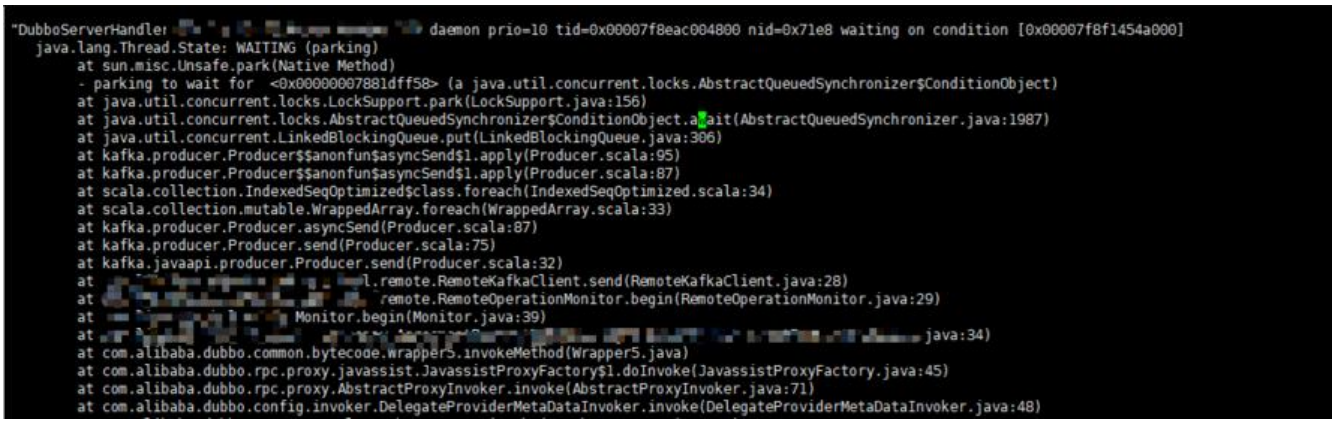
可以看到当前 dubbo 线程池已经满载运行，不能再接受新的调用。正常情况下 dubbo 线程可以很完成任务，然后归还到线程池中。由于线程执行的任务发生阻塞，消费者端调用超时。而服务提供者由于已有线程被阻塞，线程池必须不断创建新线程处理任务，直到线程数量达到最大数量，系统返回 hread pool is EXHAUSTED。

线程任务长时间被阻塞可能原因有：

- 频繁的 fullgc，导致系统暂停。
- 调用某些阻塞 API，如 socket 连接未设置超时时间导致阻塞。
- 系统内部死锁

通过分析系统堆栈 dump 情况，果然发现所有 dubbo 线程都处于 WATTING 状态。

下图为应用堆栈 dump 日志：



从堆栈日志可以看到 dubbo 线程最后阻塞在 LinkedBlockingQueue#put ，而该阻塞发生在 Kafka 送消息方法内。

这里服务 B 需要使用 Kafka 发送监控消息，为了消息发送不影响主业务，这里使用 Kafka 异步发送息。由于 Kafka 服务端最近更换了对外的端口，而服务 B Kafka 配置未及时变更。最后服务 B 修改置，服务重新启动，该问题得以解决。

二、Kafka 异步模式

下面分析 Kafka 异步发送消息阻塞的实际原因。

0.8.0 Kafka 默认使用同步模式发送消息，异步发送消息需要设置producer.type=async属性。同步式需要等待 Kafka 将消息发送到消息队列，这个过程当然会阻塞主线程。而异步模式最大的优点在于需要等待 Kafka 这个发送过程。

原本认为这里的异步是使用子线程去运行任务，但是 Kafka 异步模式并非这样。查看 Kafka 官方文档roducer,可以看到对异步模式描述。

Batching is one of the big drivers of efficiency, and to enable batching the Kafka producer has an asynchronous mode that accumulates data in memory and sends out larger batches in a si gle request. The batching can be configured to accumulate no more than a fixed number of essages and to wait no longer than some fixed latency bound (say 100 messages or 5 seconds . This allows the accumulation of more bytes to send, and few larger I/O operations on the se vers. Since this buffering happens in the client it obviously reduces the durability as any data uffered in memory and not yet sent will be lost in the event of a producer crash.

从上我们可以看到，Kafka 异步模式将会把多条消息打包一块批量发送到服务端。这种模式将会先把息放到内存队列中，直到消息到达一定数量（默认为 200）或者等待时间超限（默认为 5000ms）。

这么做最大好处在于提高消息发送的吞吐量，减少网络 I/O。当然这么做也存在明显劣势，如果生产宕机，在内存中还未发送消息可能就会丢失。

下面从 kafka 源码分析这个阻塞过程。

三、Kafka 源码解析

Kafka 消息发送端采用如下配置:

```
Properties props = new Properties();
```

```
        props.put("metadata.broker.list", "localhost:9092");
    // 选择异步发送
        props.put("producer.type", "async");
        props.put("serializer.class", "kafka.serializer.StringEncoder");
        props.put("queue.buffering.max.messages","1");
        props.put("batch.num.messages","1");
        Producer<Integer, String> producer= new Producer(new ProducerConfig(props));
        producer.send(new KeyedMessage("test", "hello world"));
```

这里设置 producer.type=async,从而使 Kafka 异步发送消息。

send 方法源码如下：

ps: 这个版本 Kafka 源码采用 Scala 编写，不过源码还是比较简单，比较容易阅读。

```
  def send(messages: KeyedMessage[K,V]*) {
    if (hasShutdown.get)
      throw new ProducerClosedException
    recordStats(messages)
    sync match {
      case true => eventHandler.handle(messages)
    // 由于 producer.type=async 异步发送
      case false => asyncSend(messages)
    }
  }
```

由于我们上面设置 producer.type=async，这里将会使用 asyncSend 异步发送模式。

asyncSend 源码如下：

```
  private def asyncSend(messages: Seq[KeyedMessage[K,V]]) {
    for (message <- messages) {
      val added = config.queueEnqueueTimeoutMs match {
        case 0  =>
          queue.offer(message)
        case _  =>
          try {
            config.queueEnqueueTimeoutMs < 0 match {

            case true =>
              queue.put(message)
              true
            case _ =>
              queue.offer(message, config.queueEnqueueTimeoutMs, TimeUnit.MILLISECONDS)
            }
          }
          catch {
            case e: InterruptedException =>
              false
          }
      }
      if(!added) {
        producerTopicStats.getProducerTopicStats(message.topic).droppedMessageRate.mark()
        producerTopicStats.getProducerAllTopicsStats.droppedMessageRate.mark()
```

```
        throw new QueueFullException("Event queue is full of unsent messages, could not send
vent: " + message.toString)
      }else {
        trace("Added to send queue an event: " + message.toString)
        trace("Remaining queue size: " + queue.remainingCapacity)
      }
    }
  }
```

asyncSend 将会把消息加入到 LinkedBlockingQueue 阻塞队列中。这里根据 config.queueEnqueu
TimeoutMs参数使用不同方法。

当 config.queueEnqueueTimeoutMs=0，将会调用 LinkedBlockingQueue#offer，如果该队列未
，将会把元素插入队列队尾。如果队列未满，直接返回 false。所以如果此时队列已满，消息不再会
入队列中，然后 asyncSend 将会抛出 QueueFullException 异常。

当 config.queueEnqueueTimeoutMs < 0,将会调用 LinkedBlockingQueue#put 加入元素，如果
队列已满，该方法将会一直被阻塞直到队列存在可用空间。

当 config.queueEnqueueTimeoutMs > 0,将会调用 LinkedBlockingQueue#offer，这里与上面不
之处在于设置超时时间，如果队列已满将会阻塞知道超时。

config.queueEnqueueTimeoutMs参数通过 queue.enqueue.timeout.ms 配置生效，默认为 -1。
认情况下 LinkedBlockingQueue 最大数量为 10000，可以通过设置 queue.buffering.max.messages
改变队列最大值。

消息放到队列中后，Kafka 将会使用一个异步线程不断从队列中获取消息，批量发送消息。

异步处理消息代码如下：

```
  private def processEvents() {
    var lastSend = SystemTime.milliseconds
    var events = new ArrayBuffer[KeyedMessage[K,V]]
    var full: Boolean = false

    // drain the queue until you get a shutdown command
    Stream.continually(queue.poll(scala.math.max(0, (lastSend + queueTime) - SystemTime.mill
seconds), TimeUnit.MILLISECONDS))
                      .takeWhile(item => if(item != null) item ne shutdownCommand else true).forea
h {
      currentQueueItem =>
        val elapsed = (SystemTime.milliseconds - lastSend)
        // check if the queue time is reached. This happens when the poll method above returns
fter a timeout and
        // returns a null object
        val expired = currentQueueItem == null
        if(currentQueueItem != null) {
          trace("Dequeued item for topic %s, partition key: %s, data: %s"
              .format(currentQueueItem.topic, currentQueueItem.key, currentQueueItem.message))
          events += currentQueueItem
        }

        // check if the batch size is reached
        full = events.size >= batchSize
```

```
        if(full || expired) {
          if(expired)
            debug(elapsed + " ms elapsed. Queue time reached. Sending..")
          if(full)
            debug("Batch full. Sending..")
          // if either queue time has reached or batch size has reached, dispatch to event handler
          tryToHandle(events)
          lastSend = SystemTime.milliseconds
          events = new ArrayBuffer[KeyedMessage[K,V]]
        }
      }
      // send the last batch of events
      tryToHandle(events)
      if(queue.size > 0)
        throw new IllegalQueueStateException("Invalid queue state! After queue shutdown, %d r
maining items in the queue"
          .format(queue.size))
    }
```

这里异步线程将会不断从队列中获取任务，一旦条件满足，就会批量发送任务。该条件为：

1. 批量消息数量达到 200，可以设置 batch.num.messages 参数改变配置。

2. 等待时间到达最大的超时时间，默认为 5000ms，可以设置 queue.buffering.max.ms 改变改置。

四、问题解决办法

上面问题虽然通过更换 Kafka 正确地址解决，但是为了预防下次该问题再发生，可以采用如下方案：

1. 改变 config.queueEnqueueTimeoutMs 默认配置，像这种系统监控日志允许丢失，所以可以置 config.queueEnqueueTimeoutMs=0。

2. 升级 Kafka 版本，最新版本 Kafka 使用 Java 重写发送端逻辑，不再使用阻塞队列存储消息。