ActiveMQ 测试报告

1、 安装和配置

1、下载 amq5.2

http://people.apache.org/~gtully/staging-repos/activemq-5.2.0/org/apache/activemq/apache-activemq/5.2.0/apache-activemq-5.2.0-bin.tar.gz

tar zxvf

amq 的目录结构：

conf：配置文件 activemq.xm，log 配置文件 log4j.properties，运行日志文件 activemq.log,安全 keystore 文件，密码文件 credentials.properties 等

bin：command line tools,包括 activemq 和 activemq-admin,主要使用工具为 activemq-admin，使用命令行工具需要 jmx 支持。

data：kaha 的缺省数据存储存储目录，可以在 activemq.xml 中配置。

example：一些 demo

webapps：activemq 提供的一个 web 管理界面应用程序

lib：一些 jar 包

我们主要使用文件和命令集中在 conf 和 bin 中，考虑到测试中需要经常修改配置，应该将 conf 和 data 等用户数据配置在 activemq 的安装目录外部，activemq 可以使用命令行使用指定的配置文件，例如：

bin/activemq xbean:file:cluster/conf/activemq.xml

kaha 存储目录是配置在 activemq.xml 文件中。

2、基本配置

activemq 每个实例称为一个 broker，broker 的配置包括包括 broker 属性，desitination，connector，存储配置和安全配置，这些配置对 server 的功能和性能和很大影响，也是我们重点测试和观察的对象，安全配置在下一节单独说明。

broker 属性配置：

broker 属性有以下三个：

property	default	description
brokerName	localhost	broker 名称
useJmx	false	是否开启 jmx 支持

persistent	true	是否存储累积消息

缺省情况下在 xml 配置文件中配置：

```
<broker xmlns="http://activemq.apache.org/schema/core" brokerName="localhost" userJmx="true" persisitent="true"/>
```

也可以直接写在 url 中，使用命令行直接启动：

```
activemq  broker:(tcp://localhost:61616,network:static:tcp://remotehost:61616)?persistent=false&useJmx=true
```

我们是配置在 xml 配置文件中，其中 jmx 配置后面还有专门描述。

desitination 配置：

我们目前只使用 queue，没有使用 topic，所以只有 queue policy 的设置。activemq 可以针对每个 queue 进行单独设置，这里“>”是通配符，表示所有 queue，具体到生产环境中，需要针对每个 queue 进行设置。

这里 memoryLimit="100mb"表示 queue 的内存限制为 100M，producerFlowControl="false"表示关闭流量控制，如果不关闭流量控制，在消息量发生累积时，amq 会主动控制流量，减少消息的生产。

其它属性如下：

property	default	description
producerFlowControl	true	the producer will slow down and eventually block if no resources(e.g. memory) are available on the broker. If this is off messages get off-lined to disk to prevent memory exhaustion
enableAudit	true	tracks duplicate messages (which can occur in failover for non-persistent messages
useCache	true	persistent messages are cached for fast retrieval from store
maxPageSize	200	maximum number of persistent messages to page from store at a time
maxBrowsePageSize	400	maximum number of persistent messages to page from store for a browser
minimumMessageSize	1024	for non-serialized messages (embedded broker) - the assumed size of the message used for memory usage calculation. Serialized messages used the serialized size as the basis for the memory calculation
advisoryForConsumed	false	send an advisory message when a message is consumed by a client
advisoryForDelivered	false	send an advisory message when a message is sent to a client
advisoryForDiscardedMessages	false	send an advisory when a message is discarded
advisoryForSlowConsumers	false	send an advisory message if a consumer is deemed slow
advsioryForFastProducers	false	send an advisory message if a producer is deemed fast
advisoryWhenFull	false	send an advisory message when a limit (memory,store,temp disk) is full
useConsumerPriority	true	use the priority of a consumer when dispatching messages from a Queue
strictOrderDispatch	false	ignore least loaded and always round robin dispatch
optimizedDispatch	false	don't use a separate thread for dispatching from a Queue
lazyDispatch	false	only page in from store the number of messages that can be dispatched at time

JMX 配置：

```
<managementContext>
        <managementContext createConnector="true"/>
</managementContext>
```

在生成环境中需要对 jmx 访问进行密码保护，具体设置如下：

1、broker 属性 useJmx 设置为“true”。

2、<managementContext createConnector="false"/>，阻止 amq 创建缺省 jmx 连接。

3、在配置目录下创建 jmx.access 和 jmx.password 文件，都是 key/value 对,前者指定角色的访问权限，有 read，write，readwrite 等，后者指定角色的密码，如 zgl=abc。这里注意 jmx.password 访问权限需要设置为用户只读，否则会有 Error。

4、修改 bin/activemq 脚本中 SUNJMX 选项，修改为 SUNJMX="-Dcom.sun.management.jmxremote.port=1616 -Dcom.sun.management.jmxremote.ssl=false -Dcom.sun.management.jmxremote.password.file=${ACTIVEMQ_BASE}/conf/jmx.password -Dcom.sun.management.jmxremote.access.file=${ACTIVEMQ_BASE}/conf/jmx.access"

具体参考 http://activemq.apache.org/jmx.html 。

NetworkConnectors 配置：

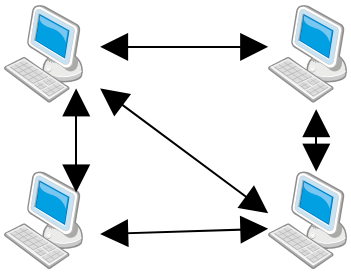
NetworkConnectors 是 amq 用来配置集群，是集群之间的节点相互发现的一种策略，在同一

```
<networkConnectors>
        <networkConnector name="default-nc" uri="multicast://239.255.2.9"/>
        <networkConnector name="amq-cluster" uri=" static://(tcp://core2:61616, tcp://core3:61616, tcp://core4:61616)"/>
</networkConnectors>
```

集群中的消息可以进行负载均衡，有就是说一个节点在没有消费者时会将累积消息 forward 到另外一台较为空闲的节点上（看起来很美，但是据我们测试发现效果差强人意啊）.

Network 连接有两种方法，一种是使用 multicast，同一集群的节点都配置同一个多播地址，节点之间直接通过多播消息互相发现并建立连接。

另外一种是使用静态 IP 地址指定需要连接的机器，通过相互指定 ip 来达到进来相互连接的目的，如果有 4 个节点，如下图，这时候每个节点需要在 uri 中指定另外 3 个节点的 ip 地址，如 core1 的 uri 可能是 static://(tcp://core2:61616, tcp://core3:61616, tcp://core4:61616)，其余类推。



因为我们网络配置的问题，多播可能会有问题（具体见论坛中校长的一个 jgroup 的故障分析），所以我们测试的集群采用静态 ip 的方法。

使用静态 ip 时一些重要的属性：

property	default	description
initialReconnectDelay	1000	time(ms) to wait before attempting a reconnect (if useExponentialBackOff is false)

maxReconnectDelay	30000	time(ms) to wait before attempting to re-connect
useExponentialBackOff	true	increases time between reconnect for every failure in a reconnect sequence
backOffMultiplier	2	multipler used to increase the wait time if using exponential back off

其它属性设置：

property	default	description
name	bridge	name of the network - for more than one network connector between the same two brokers - use different names
dynamicOnly	false	if true, only forward messages if a consumer is active on the connected broker
decreaseNetworkConsumerPriority	false	decrease the priority for dispatching to a Queue consumer the further away it is (in network hops) from the producer
networkTTL	1	the number of brokers in the network that messages and subscriptions can pass through
conduitSubscriptions	true	multiple consumers subscribing to the same destination are treated as one consumer by the network
excludedDestinations	empty	destinations matching this list won't be forwarded across the network
dynamicallyIncludedDestinations	empty	destinations that match this list **will** be forwarded across the network **n.b.** an empty list means all destinations not in the exluded list will be forwarded
staticallyIncludedDestinations	empty	destinations that match will always be passed across the network - even if no consumers have ever registered an interest
duplex	false	if true, a network connection will be used to both produce ***AND*** Consume messages. This is useful for hub and spoke scenarios when the hub is behind a firewall etc.

一个复杂的例子：

具体参考 http://activemq.apache.org/networks-of-brokers.html

```
<networkConnectors>
      <networkConnector uri="static://(tcp://localhost:61617)"
         name="bridge"
         dynamicOnly="false"
         conduitSubscriptions="true"
         decreaseNetworkConsumerPriority="false">
            <excludedDestinations>
                        <queue physicalName="exclude.test.foo"/>
                        <topic physicalName="exclude.test.bar"/>
            </excludedDestinations>
            <dynamicallyIncludedDestinations>
                        <queue physicalName="include.test.foo"/>
                        <topic physicalName="include.test.bar"/>
            </dynamicallyIncludedDestinations>
            <staticallyIncludedDestinations>
                        <queue physicalName="always.include.queue"/>
                        <topic physicalName="always.include.topic"/>
            </staticallyIncludedDestinations>
      </networkConnector>
    </networkConnectors>
```

存储配置：

amq 可以使用文件系统或者数据库存储累积的消息，文件系统存储使用的是其自有的 kaha 存储系统，数据库可以使用内嵌的 Derby，也可以使用 mysql、oracle 或者 pg，只要在 xml 配置文件中配置正确的数据库就可以了。

kaha 文件系统实际上上是一个文件索引系统，有两部分组成，一个是数据文件系统，由一个个独立的文件组成，缺省文件大小是 32M 大（可配置），另外一个是索引文件系统，记录消息在数据文件中的位置信息以及数据文件中的空闲块信息 。数据文件是存储到硬盘上的，索引文件是缓存在内存中的。所以这个存储系统对大消息存储有利，象我们的 memberId 之类的文本消息，实际上是浪费，索引比消息还大，哈。

具体参考：http://activemq.apache.org/amq-message-store.html

数据库存储系统是使用 jdbc 连接数据库存储消息，amq 只要配置数据源就可以了 。数据库存储配置有两种，一种是使用 amq 自身的高性能的 journey 日志来记录消息日志，另外一种是完全由数据库系统保存消息包括日志，但是性能较低。

```
<persistenceAdapter>
            <amqPersistenceAdapter syncOnWrite="false" directory="${activemq.base}/data"
indexBinSize="8192" cleanupInterval="30000" maxFileLength="32 mb" forceRecoverReferenceStore="false"
recoverReferenceStore="false"/>
        </persistenceAdapter>
        <!-- Use the following if you wish to configure the journal with JDBC -->
        <persistenceAdapter syncOnWrite="false" directory="${activemq.base}/data"  maxFileLength="50 mb"
forceRecoverReferenceStore="false" recoverReferenceStore="false">
            <journaledJDBC dataDirectory="${activemq.base}/data" dataSource="#mysql-ds"/>
        </persistenceAdapter>
        <!-- Or if you want to use pure JDBC without a journal -->
        <persistenceAdapter>
            <jdbcPersistenceAdapter dataSource="#postgres-ds"/>
        </persistenceAdapter>
```

如果使用 jdbc 存储，数据源配置:

```
<!-- Postgres DataSource Sample Setup -->
    <bean id="postgres-ds" class="org.postgresql.ds.PGPoolingDataSource">
      <property name="serverName" value="localhost"/>
      <property name="databaseName" value="activemq"/>
      <property name="portNumber" value="0"/>
      <property name="user" value="activemq"/>
      <property name="password" value="activemq"/>
      <property name="dataSourceName" value="postgres"/>
      <property name="initialConnections" value="1"/>
      <property name="maxConnections" value="10"/>
    </bean>

    <!-- MySql DataSource Sample Setup -->
    <bean id="mysql-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
      <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
      <property name="url" value="jdbc:mysql://core6/activemq?relaxAutoCommit=true"/>
      <property name="username" value="root"/>
      <property name="password" value=""/>
      <property name="maxActive" value="200"/>
      <property name="poolPreparedStatements" value="true"/>
    </bean>


    <!-- Oracle DataSource Sample Setup -->
    <bean id="oracle-ds" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
      <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
      <property name="url" value="jdbc:oracle:thin:@localhost:1521:AMQDB"/>
      <property name="username" value="scott"/>
      <property name="password" value="tiger"/>
      <property name="maxActive" value="200"/>
      <property name="poolPreparedStatements" value="true"/>
    </bean>

    <!-- Embedded Derby DataSource Sample Setup -->
    <bean id="derby-ds" class="org.apache.derby.jdbc.EmbeddedDataSource">
      <property name="databaseName" value="derbydb"/>
      <property name="createDatabase" value="create"/>
    </bean>
```

因为 jdbc 性能较差，大概和 kaha 相比吞吐量相差一个数量级，所以我们使用 kaha 存储，实际配置：

```
<persistenceAdapter>
           <amqPersistenceAdapter syncOnWrite="false" directory="${activemq.base}/data"
indexBinSize="8192" cleanupInterval="30000" maxFileLength="32 mb" forceRecoverReferenceStore="false"
recoverReferenceStore="false"/>
</persistenceAdapter>
```

各属性含义如下：

property	default	description
syncOnWrite	false	消息是否同步写到硬盘，这个选项对性能影响非常大
directory	null	数据存储目录，缺省为安装目录下的 data 目录，生成环境上需要配置到外部，以方便 amq 升级
indexBinSize	1024	索引文件缓存页面数，缺省为 1024，当 amq 扩充或者缩减存储时，会锁定整个 broker，导致一定时间的阻塞，所以这个值应该调整到比较大，但是代码中实现会动态伸缩，调整效果并不理想。
cleanupInterval	30000	定时清理存储文件，将那些完全为空的文件删除
maxFileLength	32M	数据文件大小，设置大一点会提高性能（？）

forceRecoverReferenceStore	true	这两个选项是 amq 未公开选项，缺省设置会到每次恢复强制从日志一条一条依次 redo 恢复，那是相当的慢啊，100w 数据多要进半个小时，帅哥和校长这两个黑客从源代码扣出来的，生产环境必须设置，这也是我们不得不使用 5.2 的一个原因。
recoverReferenceStore	true	

全部属性：

property name	default value	Comments
directory	activemq-data	the path to the directory to use to store the message store data and log files
useNIO	true	use NIO to write messages to the data logs
syncOnWrite	false	sync every write to disk
maxFileLength	32mb	a hint to set the maximum size of the message data logs
persistentIndex	true	use a persistent index for the message logs. If this is false, an in-memory structure is maintained
maxCheckpointMessageAddSize	4kb	the maximum number of messages to keep in a transaction before automatically committing
cleanupInterval	30000	time (ms) before checking for a discarding/moving message data logs that are no longer used
indexBinSize	1024	default number of bins used by the index. The bigger the bin size - the better the relative performance of the index
indexKeySize	96	the size of the index key - the key is the message id
indexPageSize	16kb	the size of the index page - the bigger the page - the better the write performance of the index
directoryArchive	archive	the path to the directory to use to store discarded data logs
archiveDataLogs	false	if true data logs are moved to the archive directory instead of being deleted

具体参考：http://activemq.apache.org/amq-message-store.html

SystemUsage 配置：

SystemUsage 配置设置了一些系统内存和硬盘容量，当系统消耗超过这些容量设置时，amq 会“slow down producer”，还是很重要的。

```
<systemUsage>
            <systemUsage>
                <memoryUsage>
                    <memoryUsage limit="512 mb"/>
                </memoryUsage>
                <storeUsage>
                    <storeUsage limit="8 gb" name="foo"/>
                </storeUsage>
                <tempUsage>
                    <tempUsage limit="256 mb"/>
                </tempUsage>
            </systemUsage>
        </systemUsage>
```

各属性含义如下：

property	default	description
memoryUsage	20M	amq 使用内存大小，照 amq 论坛上说，这个值应该大于所有 durable desitination 设置的 memoryUsage 之和，否则会导致硬盘 swap，影响性能。
storeUsage	1G	kaha 数据存储大小，如果设置不足，性能会下降到 1 个 1 个发的地步哦

tempUsage	100M	非 persistent 的消息存储在 temp 区域，我们实际上没有这个需要，设置为 5M 就可以了。

TransportConnector 配置：

这里配置 broker 的监听端口，客户端和其它的 broker 都通过这个端口连接到这个 broker 上。这里设置 broker 的监听地址和端口，需要注意的是，如果使用 static uri 配置 cluster，不需

```
<transportConnectors>
          <transportConnector name="openwire" uri="tcp://localhost:61616"
discoveryUri="multicast://239.255.2.9"/>
       </transportConnectors>
```

要配置 discoveryUri，另外也可以使用 nio 协议。

具体参考：http://activemq.apache.org/configuring-transports.html

Jetty 配置：

配置一个 jetty 服务器，提供 web 形式的管理界面，生产环境中需要禁止。

安全配置：

具体参考：http://activemq.apache.org/security.html

常用命令：

启动：bin/activemq xbean:file:cluster/conf/amq1.xml

停止：bin/activemq-admin stop --jmxurl service:jmx:rmi:///jndi/rmi://core3:1616/jmxrmi --jmxuser monitor --jmxpassword 1234

查询：bin/activemq-admin query –QQueue=”test*” --jmxurl service:jmx:rmi:///jndi/rmi://core3:1616/jmxrmi --jmxuser monitor --jmxpassword 1234

bin/activemq-admin query --objname Type=Connect,BrokerName=amq* --jmxurl service:jmx:rmi:///jndi/rmi://core3:1616/jmxrmi --jmxuser monitor --jmxpassword 1234

具体参考：http://activemq.apache.org/activemq-command-line-tools-reference.html

2、 单机测试

测试机器：core6，8 个 cpu，amq 启动内存为 1024M

- 启动和停机测试
  1) 正常启动在 1 分钟之内
  2) 消息累积时启动

  如果不设置 forceRecoverReferenceStore，recoverReferenceStore，即使是正常开关机器，100W 消息以上也需要将近半个小时，设置这两个选项后，20W 数据重启大概在 1 分钟以内，100W 在 3 分钟之内，200W 需要约 10 分钟。

  这里需要说明的是，恢复时间主要消耗在 kaha 上，如果使用 jdbc 存储，所有恢复时间都在 1 分钟之内。
  3) amq 停机速度很快，基本没有出现过需要强制 kill 的情况。

- 消息发送测试

消息大小为 2000 字符

Jvm 参数默认配置

一个 Producer 与一个 Broker 独立在不同的机器.

- **不同参数和消息数量对吞吐量的影响**

A. DeliveryMode=PERSISTENT

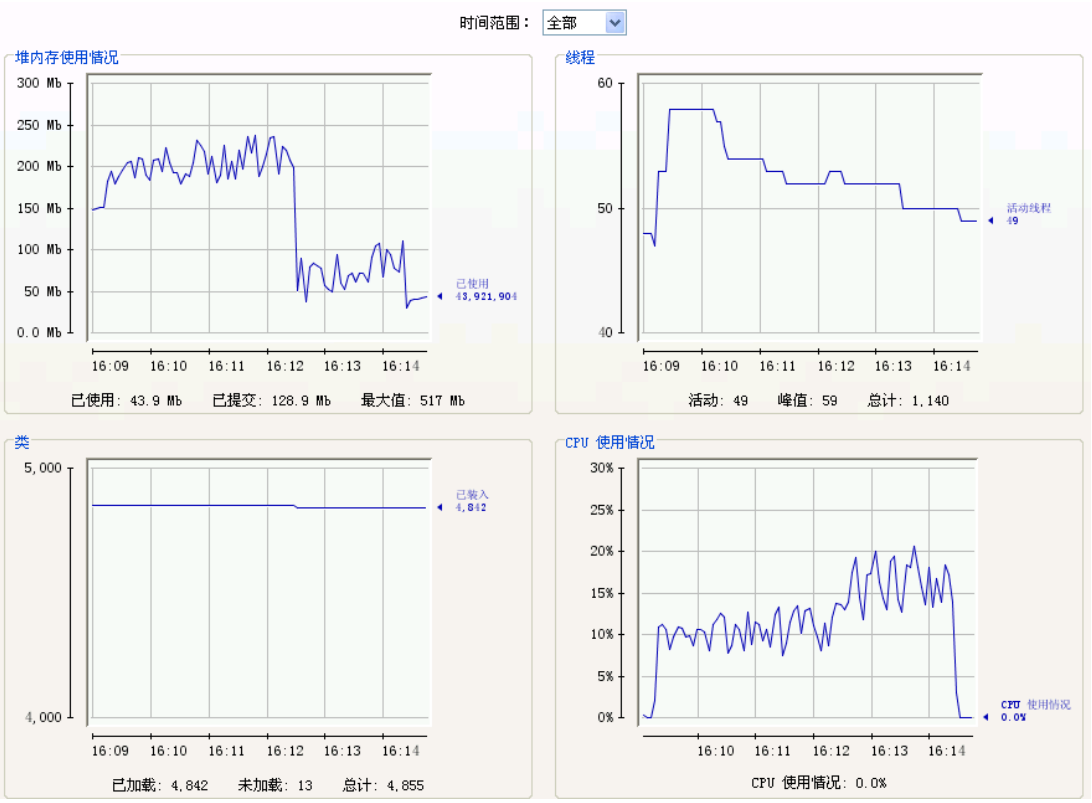
useAsyncSend= false

发送消息总量 100W

速度保持在 3200 条/s 左右，

发送速度比较平稳

Jconsole 过程图



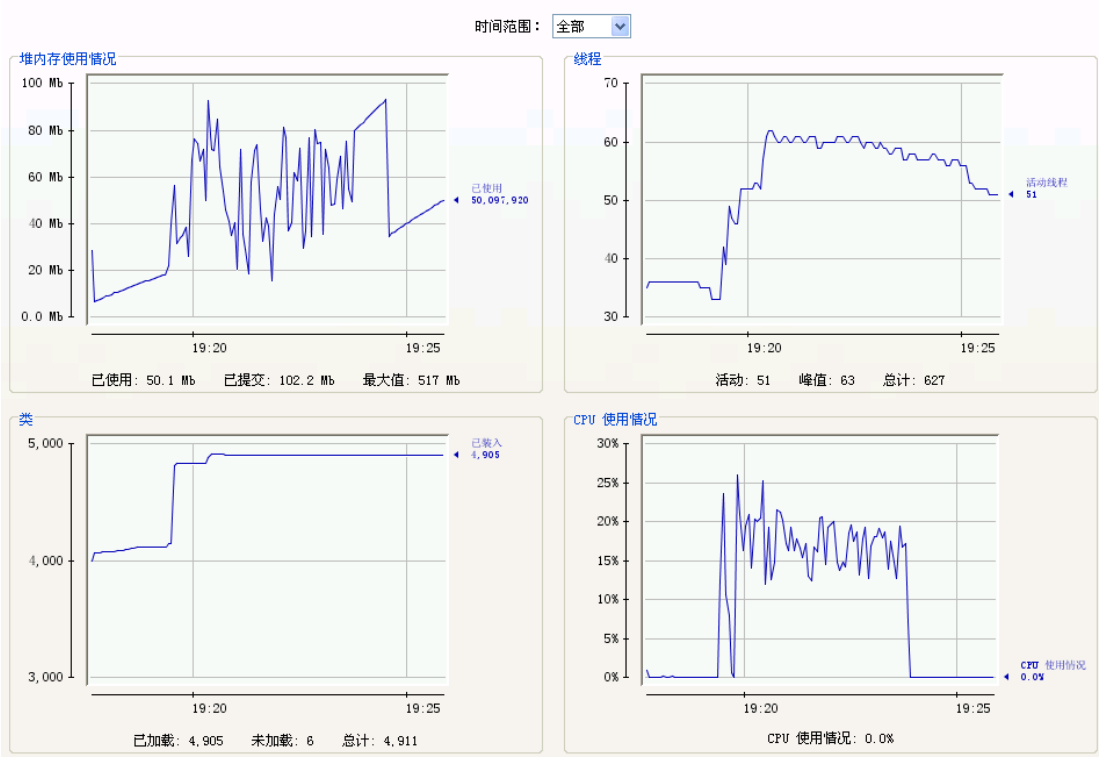
B. DeliveryMode=PERSISTENT

useAsyncSend= true 时（broker 挂机时会出现消息丢失），

速度保持在 4100 条/s 左右，

发送消息总量 100W，

发送速度比较平稳。



C. DeliveryMode=PERSISTENT

useAsyncSend= false 时，

发送消息总量 400W，

发送完 100W 时平均速度 3245 条/s，

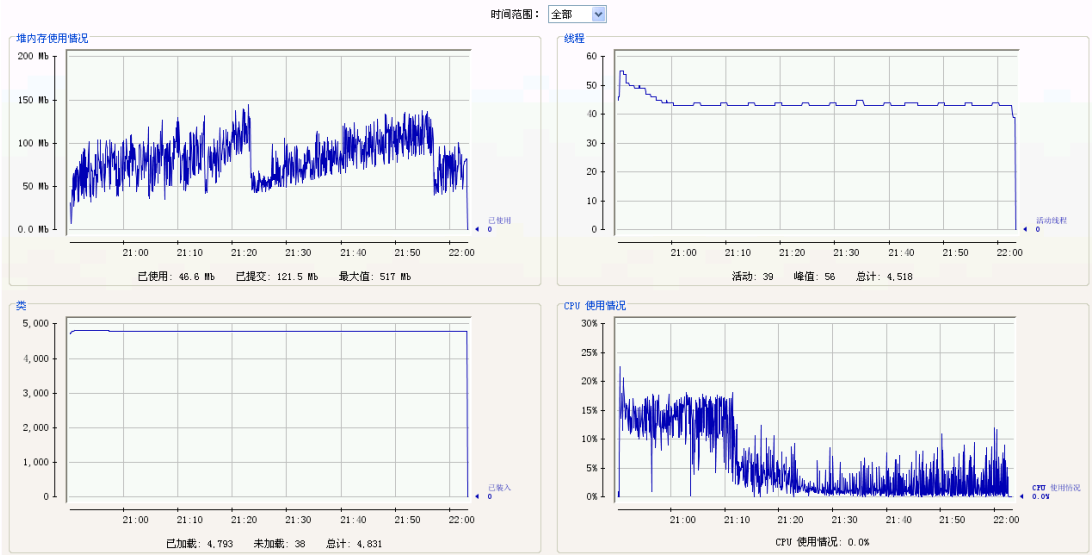
发送完 200W 时平均速度 2689 条/s，

发送完 300W 时平均速度 2512 条/s，

发送完 350W 时平均速度 2064 条/s，

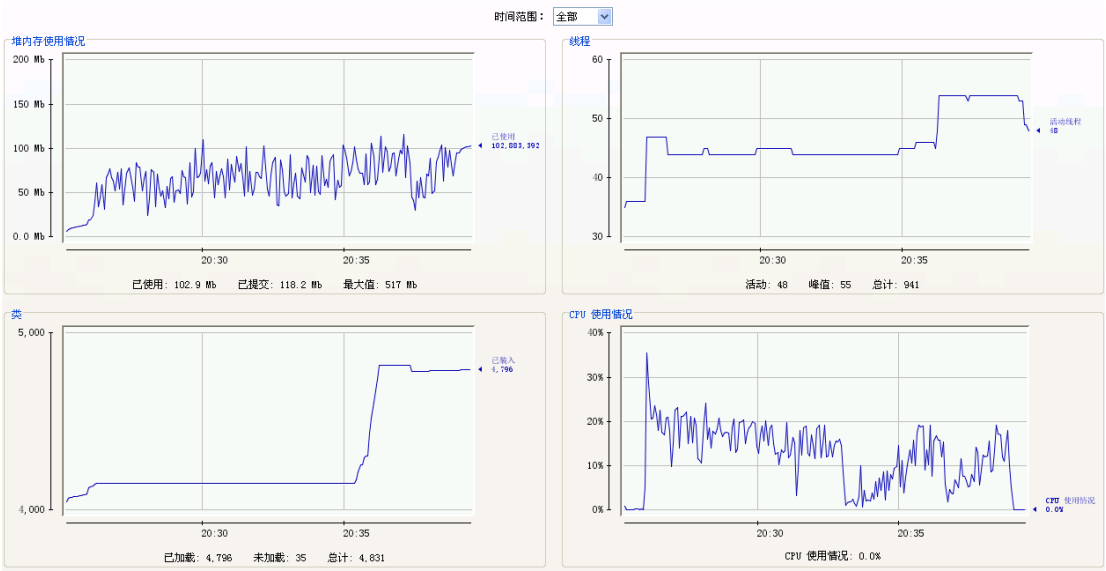
发送完 400W 时平均速度 1254 条/s，

可以看出在使用 AMQ Message Store 持久化消息时，消息堆积对 producer 对发送速度又很大影响。

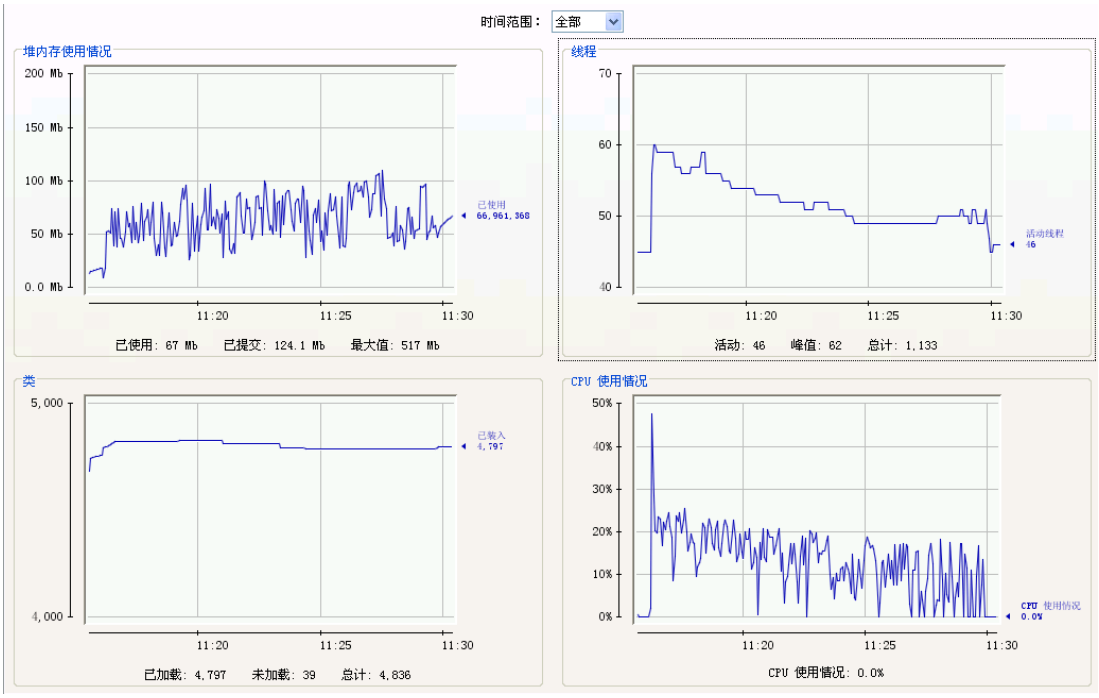


D. DeliveryMode= NON_PERSISTENT(不做消息持久化)

发送消息总量 100W，
发送速度比较平稳，
速度 10000 条/s 以上(client 和 broker 同一台机器可达 20000 以上)
DeliveryMode=PERSISTENT
useAsyncSend= false
2 个 producer，分别在不同的机器，发送消息量每个为 100W,总量 200W
Top 的 load average 在 5 左右,最高时上到 7.5。
2 个 producer 在同一时段，发送速度基本一致。
发送速度比较不平稳。
发送完总量 10W 时每个 producer 平均速度 1964 条/s，总的平均速度为 3928 条/s
发送完总量 50W 时每个 producer 平均速度 1852 条/s，总的平均速度为 3704 条/s
发送完总量 80W 时每个 producer 平均速度 1387 条/s, 总的平均速度为 2774 条/s
发送完总量 100W 时每个 producer 发送速度为 1288 条/s，总的平均速度为 2576 条/s



E. DeliveryMode=PERSISTENT
useAsyncSend= false
没有 consumer，
有 4 个 producer，在 2 台机器上，每台机器有 2 个 produer，每个 producer 同时发送 50W，消息总量有 200W。
发送速度不太稳定。
发送完总量 10W 时总的平均速度为 4000 条/s 左右，
发送完总量 50W 时总的平均速度为 3680 条/s 左右，
发送完总量 80W 时总的平均速度为 3552 条/s 左右，
发送完总量 100W 时总的平均速度为 3400 条/s 左右，
发送完总量 200W 时总的平均速度为 2500 条/s 左右，
load average 最高达到 10.07



F. DeliveryMode=PERSISTENT

useAsyncSend= false

没有 consumer，

有 8 个 producer，在 2 台机器上，每台机器有 4 个 produer，每个 producer 同时发送 25W，消息总量有 200W。

发送速度不太稳定。

发送完总量 10W 时总的平均速度为 4000 条/s 左右，

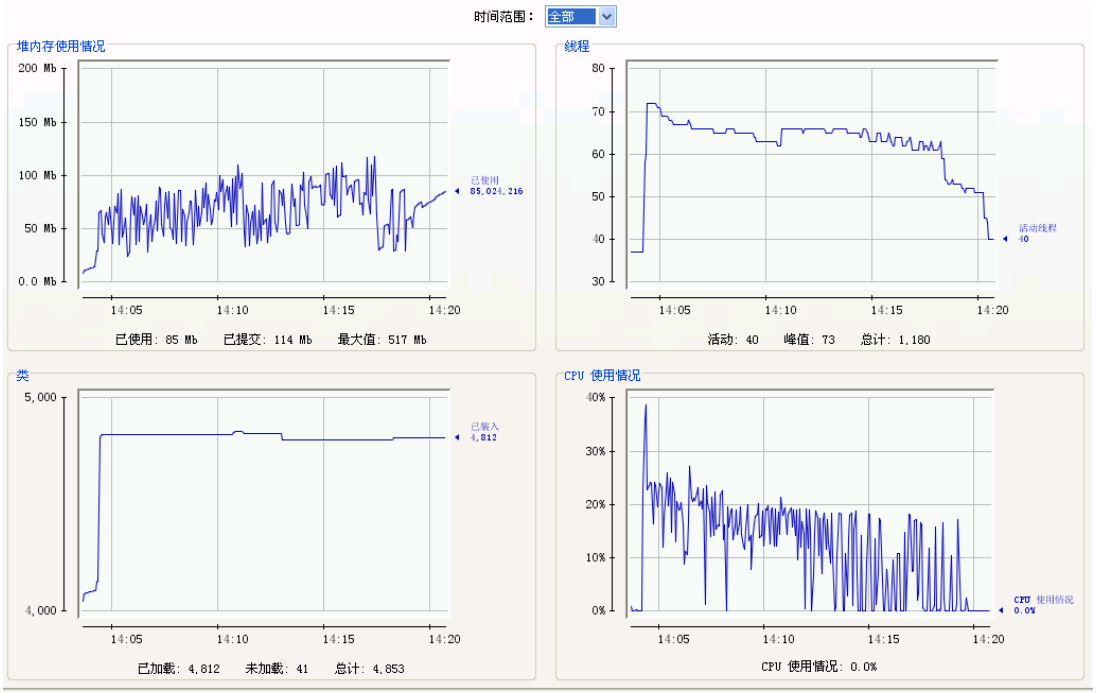
发送完总量 50W 时总的平均速度为 3450 条/s 左右，

发送完总量 80W 时总的平均速度为 3400 条/s 左右，

发送完总量 100W 时总的平均速度为 3300 条/s 左右，

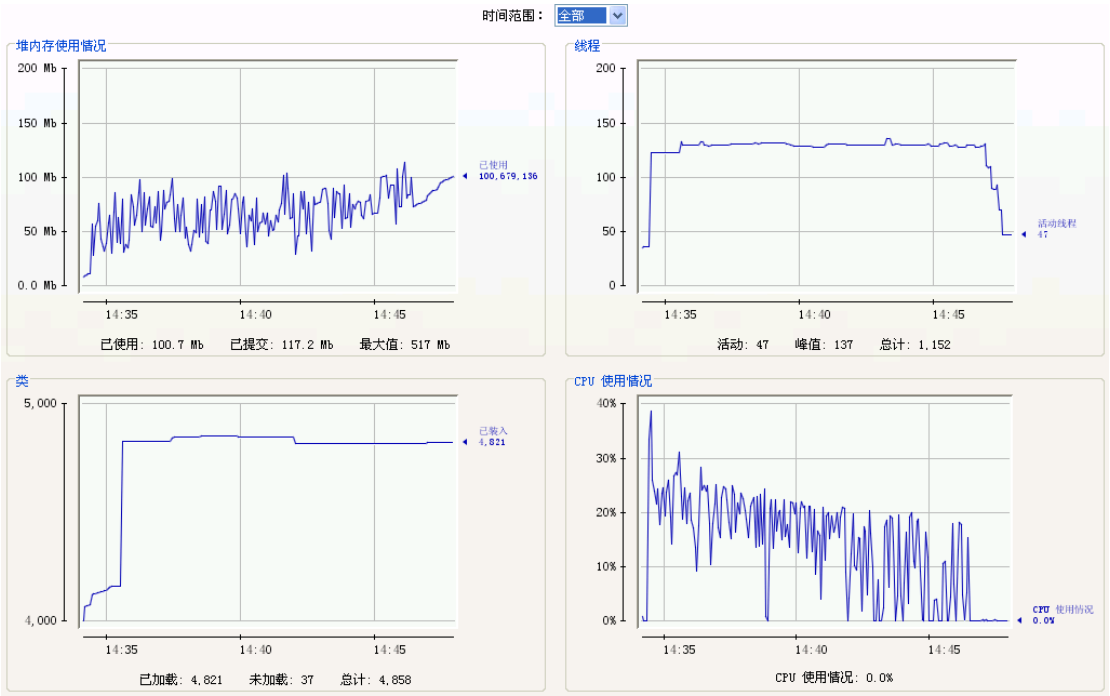
发送完总量 200W 时总的平均速度为 2200 条/s 左右.

load average 最高达到 7.5

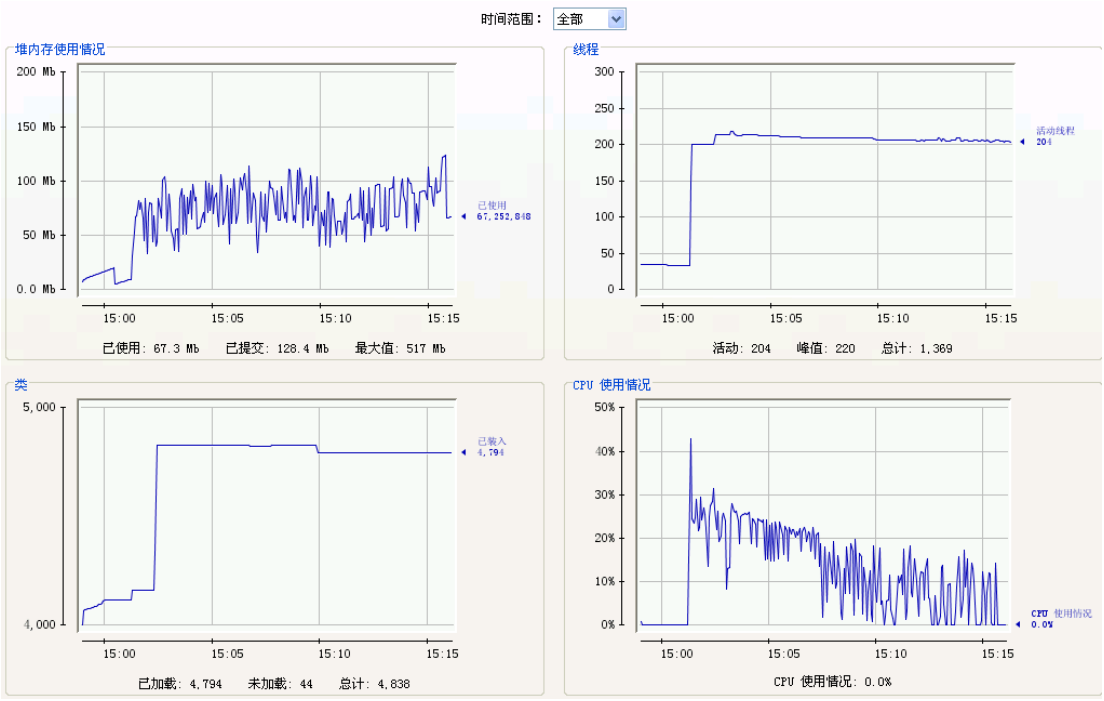


G. DeliveryMode=PERSISTENT
useAsyncSend= false
没有 consumer，
有 20 个 producer，在 2 台机器上，每台机器有 10 个 produer，每个 producer 同时发送 10W，消息总量有 200W。
发送速度相对稳定。
发送完总量 50W 时总的平均速度为 3400 条/s 左右，
发送完总量 80W 时总的平均速度为 3500 条/s 左右，
发送完总量 100W 时总的平均速度为 3400 条/s 左右，
发送完总量 160W 时总的平均速度为 3100 条/s 左右，
发送完总量 200W 时总的平均速度为 2200 条/s 左右.

H. DeliveryMode=PERSISTENT
useAsyncSend= false
没有 consumer，
有 40 个 producer，在 2 台机器上，每台机器有 20 个 produer，每个 producer 同时发送 5W，消息总量有 200W。
发送速度相对稳定。
发送完总量 10W 时总的平均速度为 4000 条/s 左右，
发送完总量 50W 时总的平均速度为 3800 条/s 左右，
发送完总量 80W 时总的平均速度为 3800 条/s 左右，
发送完总量 100W 时总的平均速度为 3800 条/s 左右，
发送完总量 160W 时总的平均速度为 3500 条/s 左右，
发送完总量 200W 时总的平均速度为 2700 条/s 左右.



I. DeliveryMode=PERSISTENT
useAsyncSend= false
没有 consumer，
有 40 个 producer，在 2 台机器上，每台机器有 40 个 produer，每个 producer 同时发送 2.5W，消息总量有 200W。
发送速度相对稳定。
发送完总量 50W 时总的平均速度为 3800 条/s 左右，
发送完总量 80W 时总的平均速度为 3950 条/s 左右，
发送完总量 100W 时总的平均速度为 3950 条/s 左右，
发送完总量 160W 时总的平均速度为 3300 条/s 左右，
发送完总量 200W 时总的平均速度为 2300 条/s 左右.
load average 最高值 9.28



分析：

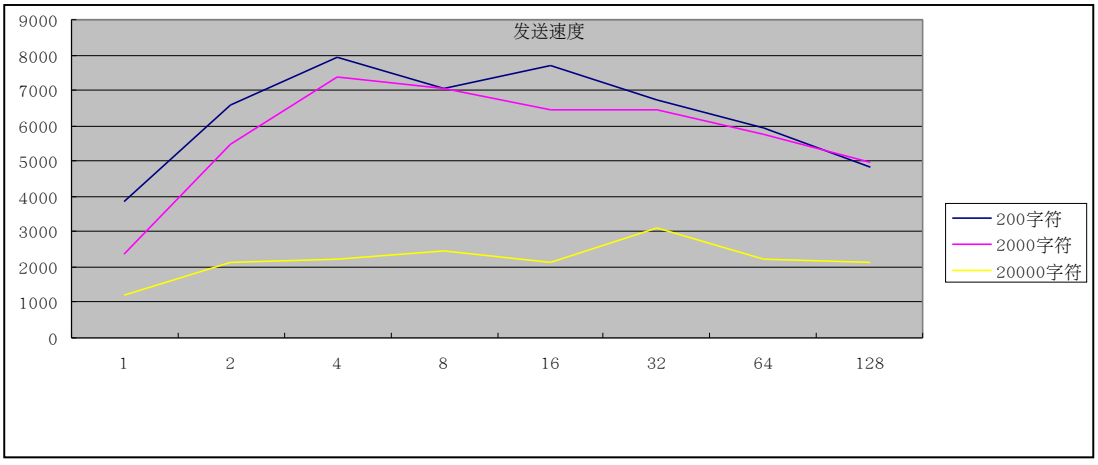
1) 消息是否 persist 对吞吐量影响很大。
2) 随着消息累积数的增加，吞吐量逐步下降，但是最终会稳定在 2000/s 左右。
3) 随着累积数量的增加，系统 load 会大幅度增加。

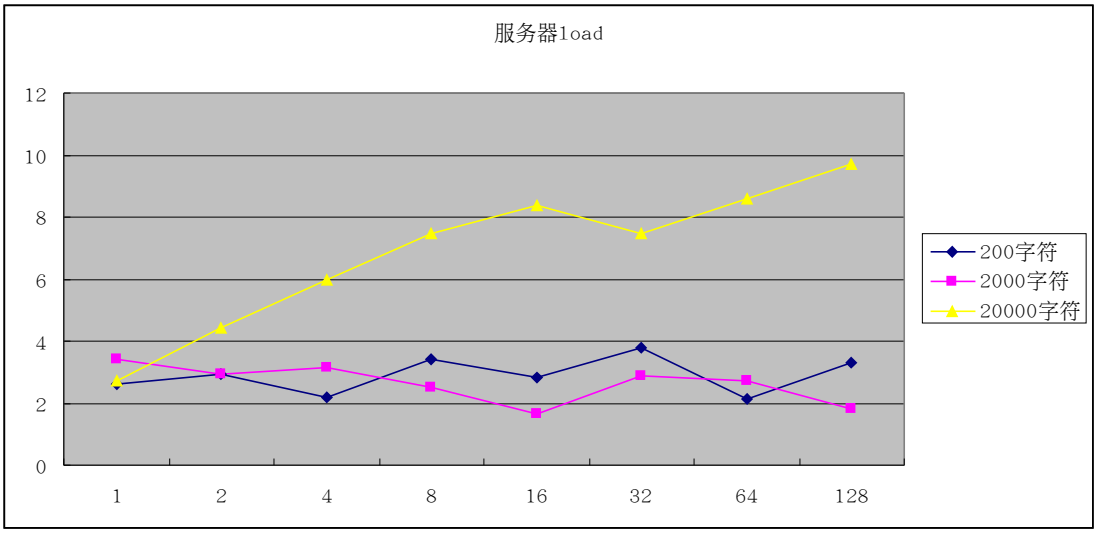
- **连接数不同对吞吐量的影响**

测试方法：

采用不同连接数发送 32w 消息的平均速度和服务器 load，每次发送前都清空队列，保证没有消息累积。

连接数		1	2	4	8	16	32	64	128
速度	200 字符	3829	6604	7913	7035	7706	6707	5934	4813
	2000 字符	2375	5475	7368	7034	6452	6433	5755	4957
	20000 字符	1216	2151	2220	2446	2135	3131	2240	2149
服务器 load	200 字符	2.62	2.95	2.18	3.41	2.83	3.77	2.12	3.29
	2000 字符	3.40	2.93	3.15	2.53	1.63	2.87	2.74	1.83
	20000 字符	2.71	4.44	5.96	7.47	8.35	7.45	8.61	9.70





分析：

4) 连接数并非越多越好，在 4~16 个连接时可以达到最大吞吐量，太大或者太小都会影响速度。
5) 大尺寸消息对速度影响非常大，在 2000B 以下区间，性能影响较小，我们目前的消息基本集中在这个区间。
6) 大尺寸消息会大大增加服务器 load，而连接数对服务器 load 影响较小，由此可以说明 amq 的主要性能瓶颈在 io 部分，cpu 操作并不密集。

测试中发现问题：

1) 发送 32w 消息，但是有事会发现队列中消息数量多于 32W，大概误差在 3~7 个之间，需要进一步确认是否是 amq 统计错误还是真正产生重复消息。
2) 每次 kaha 存储的 hash bin 调整会完全阻塞 amq，长达几秒不能响应，但是因为这个值是动态调整，单纯增长这个值并不能完全解决这个问题，需要进一步寻找解决办法。

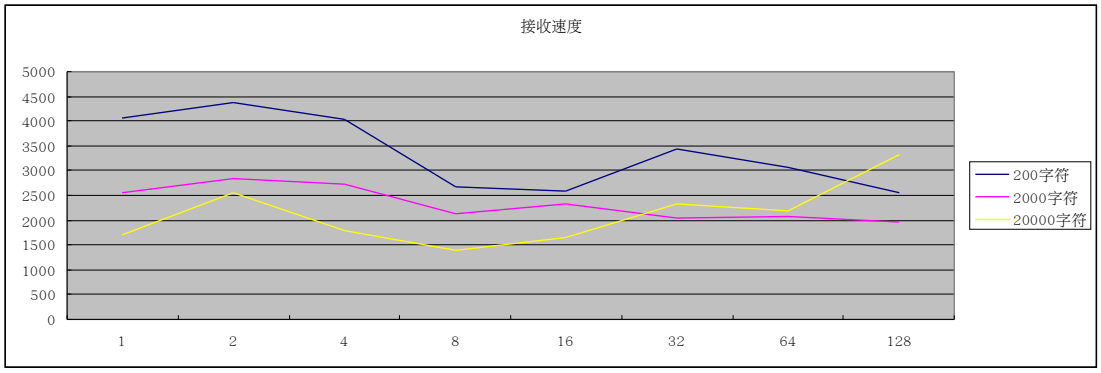
■ 消息接收测试

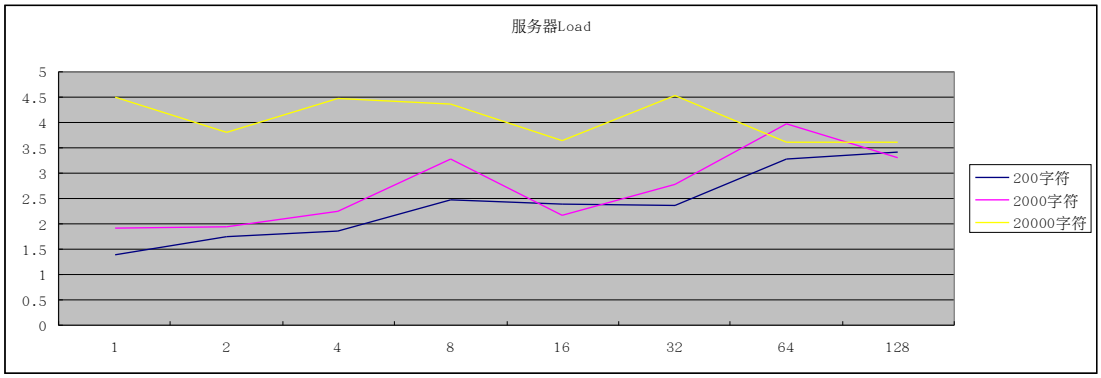
连接数对吞吐量的影响：

测试方法：

采用不同连接数发送 16w 消息的平均速度和服务器 load，每次发送前都清空队列，保证没有消息累积。

连接数		1	2	4	8	16	32	64	128
速度	200 字符	4059	4389	4048	2674	2591	3437	3055	2546
	2000 字符	2559	2832	2726	2137	2341	2056	2069	1954
	20000 字符	1709	2571	1800	1387	1640	2338	2177	3312
服务器 load	200 字符	1.40	1.74	1.85	2.47	2.39	2.35	3.29	3.43
	2000 字符	1.91	1.94	2.24	3.27	2.16	2.77	3.97	3.30
	20000 字符	4.50	3.80	4.46	4.37	3.64	4.53	3.60	3.60





分析：

1) 消费时连接数较小时速度较高。
2) 同样，消息 20k 大小时 load 很高，说明 kaha 性能还是整体瓶颈。
3) kaha 在清理文件存储时会锁定，导致短期的服务停止，可以设置 amqPersistenceAdapter 的 cleanupInterval 时间控制，缺省是 30s，可以设置为 10 分钟。

■ 同时发送和接收测试

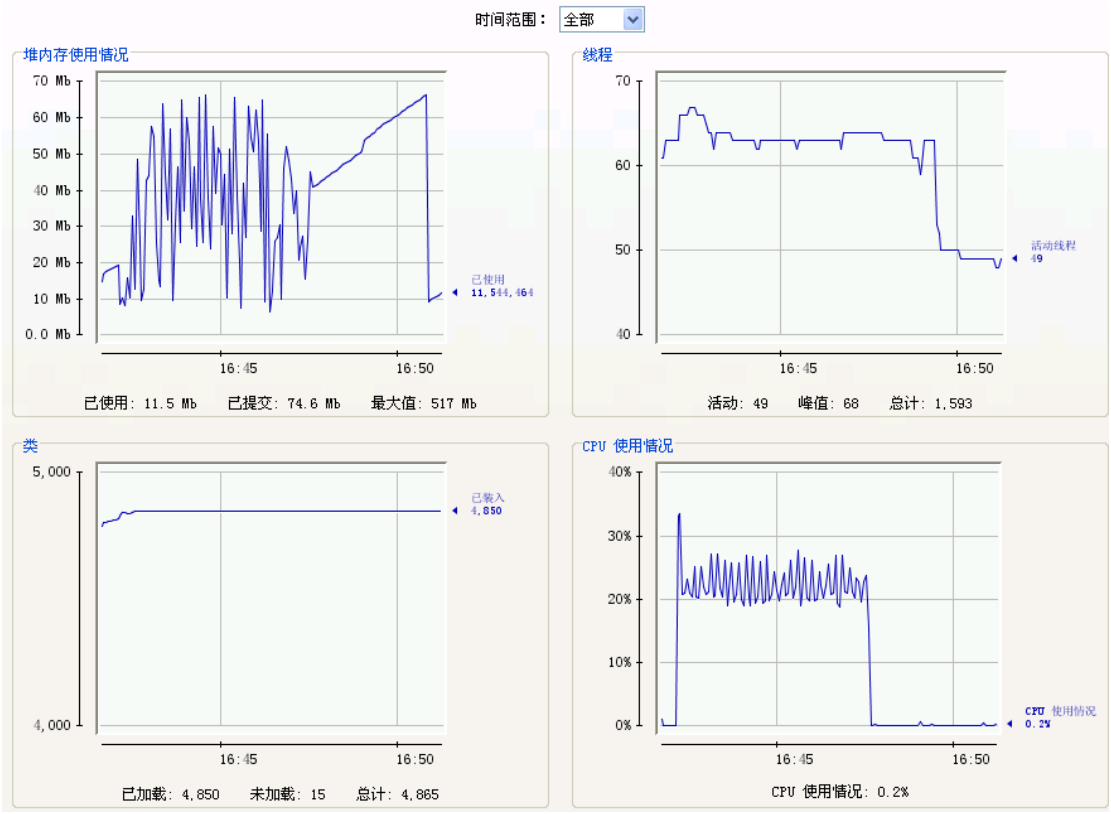
A. Producer、Broker、consumer 独立在不同的机器

1 个 producer 发送消息总量 100W ,5 个 consumer

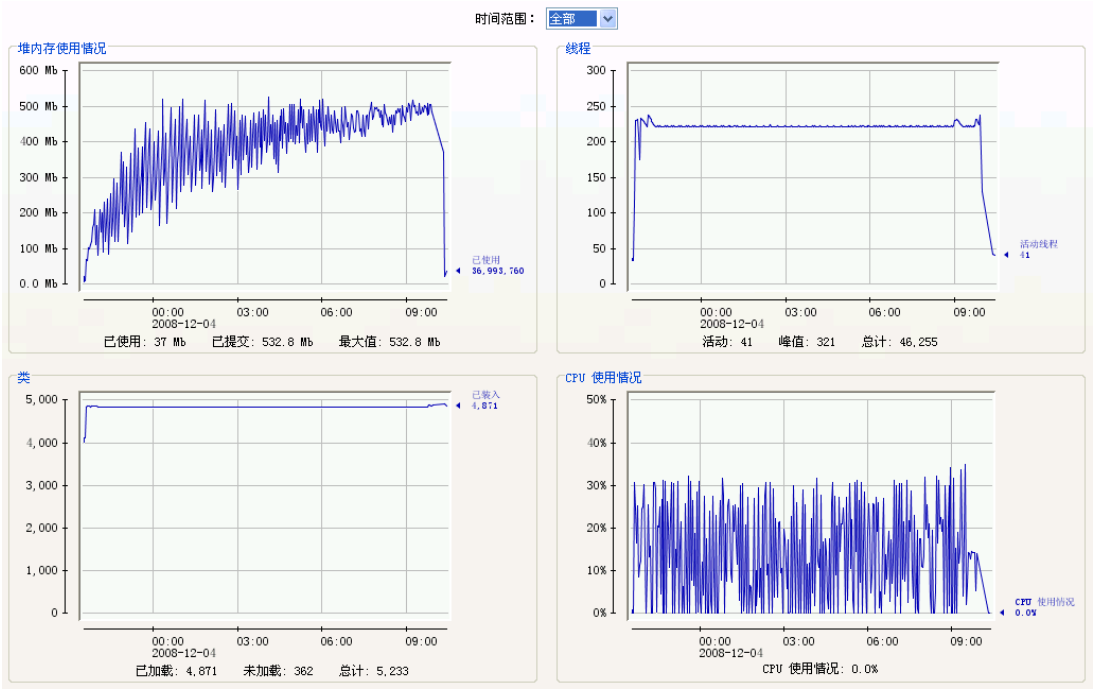
发送 3058 条/s 左右，

发送速度比较平稳。

由于 consumer 接收消息没有复杂处理，所以 broker 没有任何消息堆积。



B. Producer、Broker、consumer 独立在不同的机器，
消息总量不限制，
30 个 producer,60 个 consumer，
跑了一个晚上出现内存泄漏



C. Producer、Broker、consumer 独立在不同的机器，消息总量 300W，50 个 producer,100 个 consumer，
消息堆积量对 producer 的影响:
100w 消息堆积时，load average: 3.39, 3.90, 3.86，当时的发送速度比较平衡，3300 条/s 左右，
150W 消息堆积时，系统 load average: 8.32, 8.09, 6.16, 当时的发送速度 3000 条/s 左右，
200w 消息堆积时，系统 load average 下降到 5 左右，当时的发送速度只有 800 条/s 左右，
300w 消息堆积时，系统 load average 下降到 4 左右，当时的发送速度只有 150 条/s 左右，
消息堆积量对 consumer 的影响:
如果在没有 producer 情况下，300w 消息堆积时，系统 load average 在 4 左右，当时的分发速度 1000 条/s 左右。
如果有 producer 也在发送的情况，300w 消息堆积时出现 producer 与 consumer 抢占 IO 的使用，所以出现 producer 高时 consumer 低或 producer 低时 consumer 高，但两者相加不会超过 1000，但在 50w 消息堆积时，两者相加有较好的表现，能达到 2000-2500。在少量的消息堆积(5W 以内)时，两者相加能达到 5000 以上。在此看来，消息堆积数量对吞吐能力有比较大的影响。

D. Producer、Broker、consumer 独立在不同的机器，
消息总量 300W，
使用 64 位的 JDK 和新的 JVM 配置参数，
50 个 producer,100 个 consumer，
消息堆积量对 producer 的影响:
100w 消息堆积时，load average: 3.39, 3.90, 3.86，当时的发送速度比较平衡，3800 条/s 左右，
150W 消息堆积时，系统 load average: 8.32, 8.09, 6.16, 当时的发送速度 3200 条/s 左右，
200w 消息堆积时，系统 load average 下降到 5 左右，当时的发送速度只有 1300 条/s 左右，
300w 消息堆积时，系统 load average 下降到 4 左右，当时的发送速度只有 200 条/s 左右。

问题:

1) 单纯在压力测试下，很难达到一个消费和生产的平衡，不是说有一个简单的生产/消费的一个连接数比例可以达到不累积的效果。
2) 在压力测试下，消费速度大大小于生成速度，很容易产生消息累积，分析后认为主要是 io 吞吐量基本一定的情况下，而生产者好像有一定优先。（？）
3) 消费端被阻塞的频率和时间都远大于生产端，分析可能是消费端更容易产生 kaha 文件碎片，收空间收集线程的影响更大。

4) 生产环境下应该打开**流量控制开关**，以避免生成端占用大部分 io 能力，阻塞消费端的情况。

■ 消息正确性和完整性测试

测试方法：

发送端对消息进行编号并进行 md5 编码，接受端使用 md5 校验消息正确性，另外对使用 hashmap 校验是否有消息重复。

测试结果：

1、md5 校验完全正确，说明消息正确性可以保证。

2、在 1 个 consumer 时，没有重复消息产生，在多个 consumer 时，有重复消息产生，而且重复数量和连接数有线形关系，可以断定 amq 在消费时有线程问题，会参数重复消息。

3、 集群测试

因为 amq 集群稳定性很差，所以没有产生有效的数据，在测试过程中可以观察到以下几个问题：

1) M/S 和 Network 集群都会频繁产生节点同步异常，amq 集群实现方式目前还不是很稳定。
2) 在 M/S 和 Network 集群情况下，吞吐量都会下降到单节点配置的 1/2 和 1/3，应该是节点间状态同步和消息复制的开销。
3) 节点间的负载均衡坐的还是不够，经常会发现单一节点连接过多或者消息过多累积的情况。

4、 测试结论

1) 连接数对性能影响较小，主要影响因素为 io，在每个连接发送或者接收频率较低的情况下，amq 可以支持更多的连接数，另外，每新增一个连接，amq 需要增加两个线程处理。nio 协议在 5.2 中还不够稳定，对减低系统负载的作用也不够明显。
2) 超过 10K 的大消息对吞吐量和服务器 load 都影响很大，但是对大消息可以使用 amq 的流协议处理，这次没有测试。对我们系统中的消息基本都不会超过 2k，这种情况下性能差别不是很大。
3) 数据存储配置，jdbc 的性能远比 kaha 低，jdbc 单节点流量大概是 500~600 条，kaha 单节点处理可达 4000~5000，相差一个数量级，但是 jdbc 的稳定性很好，发送和接收非常稳定，kaha 只因为文件存储的原因，当文件存储 hash bin 扩展或者文件碎片管理的时候经常会全部阻塞，导致短时间（1s~几秒）不可访问。
4) 生产端流量控制还是有必要的，避免高峰时阻塞消费。（？）
5) 高级特性（Master/Slave,Networkconnector,Virtual Destination,Composite Destination）等 5.2 都还不够稳定，目前建议部署为客户端负载均衡，类似 memcached 的部署方式，这样也可以屏蔽 mq，做到 mq 无关，另外这样性能也最好，基本可以做到线形扩充。
6) 目前主要问题为消息重复消费问题，需要定位问题代码所在。具体表现为消息丢失（发现过 1、2 次），消费数多于发送数，消息可能有重复消费。（注：这个问题提交到 amq 后，答复在 5.3 已经解决，在 5.2 上关闭 cache 后也可以避免这个，useCache=false，经过测试，5.2 上使用这个选项后问题解决，性能有所下降但是并不严重，在 10%以内）。
7) 每一个 producer 或 consumer 连接，broker 会产生 2 个线程来提供服务，看来过多

的连接会加重 broker 的线程上下文切换的成本。

8) 打开 useAsyncSend=true 功能，能提供带来 10%-20%的发送速度提升，在 IO 频繁操作的情况，可能会更明显，但如果 broker 出现当机时，极有可能出现消息丢失。
9) 64 位的 JDK 和新的 JVM 配置参数对整体性能提供有一定的帮助。