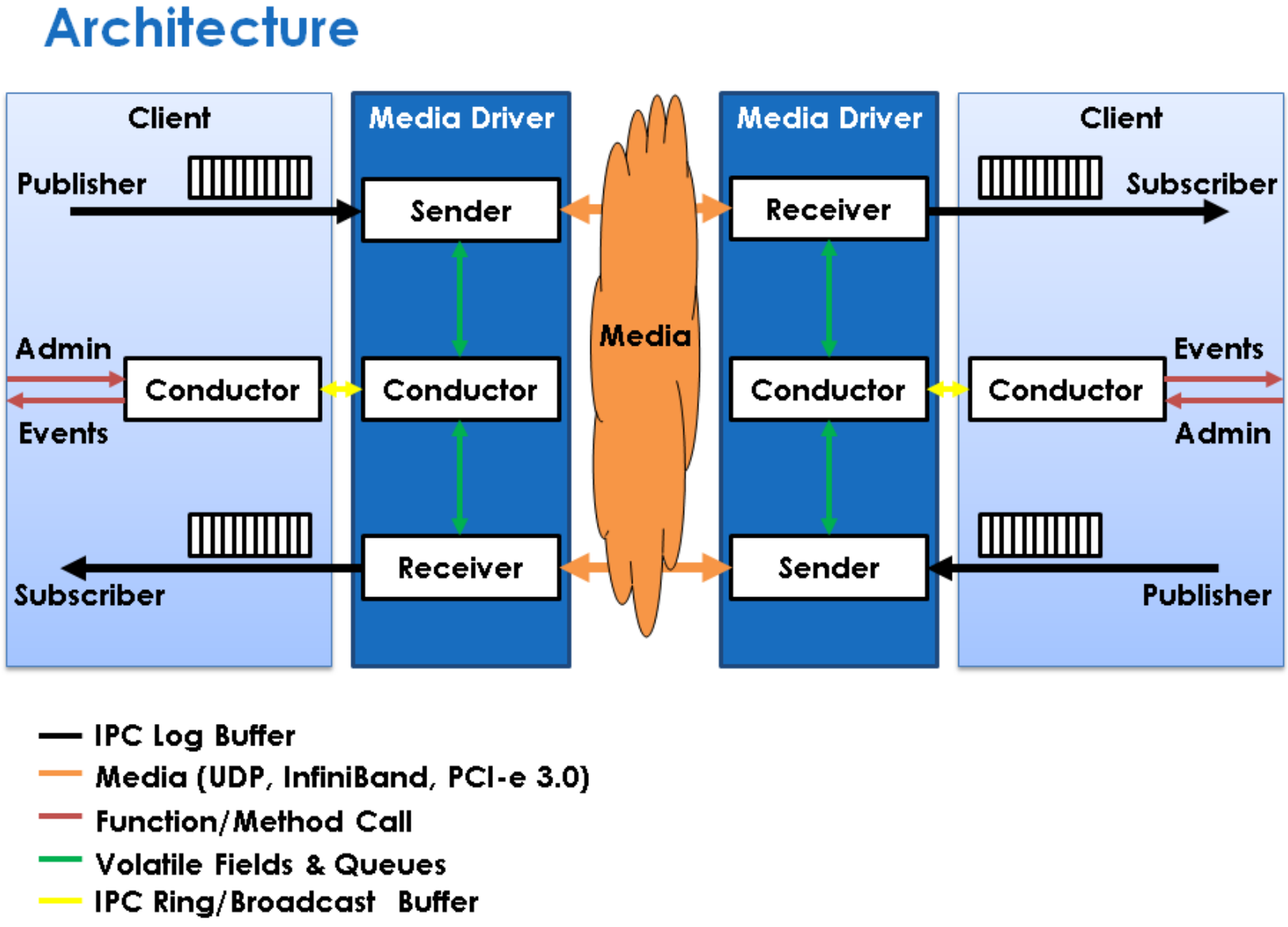
Aeron

1. Aeron 可以做什么
2. Aeron 架构
3. Aeron 示例代码
4. Aeron 配置与调优
5. Aeron 流控与拥塞控制
6. Aeron 可靠性保证

1. Aeron 可以做什么

Aeron支持UDP单播(Unicast), 多播(Multicast), 以及进程间通信(IPC), 并保证消息的可靠性和有序性.
UDP多播(Multicast)在Trading, Dealing后台系统中是消息通信的基础, 依赖于同样提供消息可靠以及有序的JGroups组件.

2. Aeron 架构


Architecture
Client
Publisher
Admin
Conductor
Events
Subscriber
Media Driver
Sender
Conductor
Receiver
Media
Media Driver
Receiver
Conductor
Sender
Client
Subscriber
Events
Conductor
Admin
Publisher
IPC Log Buffer
Media (UDP, InfiniBand, PCI-e 3.0)
Function/Method Call
Volatile Fields & Queues
IPC Ring/Broadcast Buffer

3. Aeron 示例代码

3.1 非内嵌方式

```
MediaDriver.Context ctx0 = new MediaDriver.Context();
Aeron.Context ctx1 = new Aeron.Context();
MediaDriver driver = MediaDriver.launch(ctx0); // 1
Aeron aeron = Aeron.connect(ctx1);
Subscription sub = aeron.addSubscription(
   "aeron:udp?endpoint=224.0.1.1:40456|ttl=4|mtu=16384", 10);
Publication pub = aeron.addPublication(
   "aeron:udp?endpoint=224.0.1.1:40456|ttl=4|mtu=16384", 10);
....
....
```

3.2 内嵌方式

```
MediaDriver.Context ctx0 = new MediaDriver.Context();
Aeron.Context ctx1 = new Aeron.Context();
MediaDriver driver = MediaDriver.launchEmbedded(ctx0); // 1
ctx1.aeronDirectoryName(ctx0.aeronDirectoryName()); // 2
Aeron aeron = Aeron.connect(ctx1);
Subscription sub = aeron.addSubscription(
   "aeron:udp?endpoint=224.0.1.1:40456|ttl=4|mtu=16384", 10);
Publication pub = aeron.addPublication(
   "aeron:udp?endpoint=224.0.1.1:40456|ttl=4|mtu=16384", 10);
....
....
```

3.3 单播发布订阅

```
// PING-PONG
//ping机器
Publication pub0 = aeron.addPublication(
    "aeron:udp?endpoint=localhost:40123", 10);
Subscription sub0 = aeron.addSubscription(
    "aeron:udp?endpoint=localhost:40124", 10));

//pong机器
Publication pub1 = aeron.addPublication(
    "aeron:udp?endpoint=localhost:40124", 10);
Subscription sub1 = aeron.addSubscription(
    "aeron:udp?endpoint=localhost:40123", 10));
```

3.4 多播发布订阅

```
Subscription sub = aeron.addSubscription(
   "aeron:udp?endpoint=224.0.1.1:40456|ttl=4|mtu=16384", 10);
Publication pub = aeron.addPublication(
   "aeron:udp?endpoint=224.0.1.1:40456|ttl=4|mtu=16384", 10);
```

3.5 发布订阅的一些细节

```
// 订阅拉取消息时不要阻塞发布端
IdleStrategy strategy = new BusySpinIdleStrategy();
new Thread(() -> {
    while(running.get()) {
        strategy.idle(sub.poll(fragmentHandler, 256));
    }
}).start();

// 发布时可能需要按需重试
long r = 0L;
byte[] data = new byte[512]; // fill data
UnsafeBuffer buffer = new UnsafeBuffer(allocateDirect(1024))
buffer.putBytes(0, data)
strategy.reset();
while((r = pub.offer(buffer, 0, data.length)) < 0L) {
    strategy.idle((int)r);
    // r
    // -1:NOT_CONNECTED. -2:BACK_PRESSURED
    // -3:ADMIN_ACTION. -4:CLOSED. -5:MAX_POSITION_EXCEEDED
}
```

3.6 等待策略

```
IdleStrategy strategy = new BusySpinIdleStrategy();
strategy.idle(r);
// 在jdk9中, 等同于调用Thread.onSpinWait();
// jdk9以下, 等同于cpu空转

IdleStrategy strategy = new BackoffIdleStrategy(96,32,1,5);
strategy.idle(r);
// cpu先空转96个cycle, 之后降级再Thread.yield() 32 次
// 之后降级LockSupport.parkNanos(1),
// 直到LockSupport.parkNanos(5)

// 采用BusySpinIdleStrategy的话会耗费较多的cpu
// 在实际中要做一些trade-off, 在官方的LowLatencyMediaDriver中
// 采用的都是BusySpinIdleStrategy
```

3.7 Fragment重整合

```
// 对于大于mtu的消息发送端会进行分割成多个Fragment发送
// 在接收端需要把多个Fragment重整合成一个消息. 非重整合的示例如下
FragmentHandler h = (buffer, offset, length, header) -> {
    // do something with single fragment.
};
new Thread(() -> {
    while(running.get()) strategy.idle(sub.poll(h, 256));
}).start();
```

```
// 重整合示例如下
FragmentHandler h = new FragmentAssembler(
    (buffer, offset, length, header) -> {
        byte[] payload = new byte[length];
        buffer.getBytes(offset, payload);
        onMessage(payload);
});

new Thread(() -> {
    while(running.get()) strategy.idle(sub.poll(h, 256));
}).start();
```

3.8 错误处理

```
//实现ErrorHandler并注册到MediaDriver.Context和Aeron.Context
public class AeronErrorHandler implements ErrorHandler {
    @Override public void onError(Throwable e) {
        if(e instanceof ConductorServiceTimeoutException) {
            // handle fatal error
        } else if(e instanceof DriverTimeoutException) {
            // handle fatal error
        } else { LOGGER.error("aeron error", e); }
    }
}

MediaDriver.Context ctx0 = new MediaDriver.Context();
ctx0.errorHandler(new AeronErrorHandler()) //注册进MediaDriver
Aeron.Context ctx1 = new Aeron.Context();
ctx1.errorHandler(new AeronErrorHandler()) //注册进Aeron
```

3.9 发布以及取消发布事件

```
public class Available implements AvailableImageHandler {
    @Override public void onAvailableImage(Image image) {
        LOGGER.info("onAvailableImage from {} session {}",
            image.sourceIdentity(),image.sessionId());
    }
}
public class Unavailable implements UnavailableImageHandler {
    @Override public void onUnavailableImage(Image image) {
        LOGGER.info("onUnavailableImage from {} session {}",
            image.sourceIdentity(), image.sessionId());
    }
}
// 将相应事件注册到Aeron中, 注意, 订阅端触发这两个事件
Aeron.Context ctx1 = new Aeron.Context();
ctx1.availableImageHandler(new Available());
ctx1.unavailableImageHandler(new Unavailable());
// 组播条件下有3台机器互相通信, 并且每台机器都调用了一次
// aeron.addPubliccation, aeron.addaddSubscription
// 那么每台机器会触发3次onAvailableImage,生成3个image logbuffer
```

4. Aeron 配置与调优

4.1 URI中的一些相关参数

```
aeron:udp?endpoint=224.0.1.1:40456|interface=127.0.0.1 \
    ttl=4|mtu=16384|term-length=134217728

// endpoint     :  多播地址
// interface    :  收发消息地址 127.0.0.1表示loopback
// ttl          :  IP_MULTICAST_TTL 设定multicast封包的ttl值
// mtu          :  虚拟mtu, 超过mtu的消息会以Fragment发送
// term-length :  Memory Mapped File 长度.
```

4.2 针对MediaDriver的重要参数

```
System.setProperty("aeron.socket.so_sndbuf", "8388608");
System.setProperty("aeron.socket.so_rcvbuf", "8388608");
//分别设置socket的SO_SNDBUF和SO_RCVBUF

MediaDriver.Context ctx0 = new MediaDriver.Context();
ctx0.initialWindowLength(8388608);//流控,MTU<=T<=SO_RCVBUF

//Publishers与Sender之间的流控窗口, 默认值为term-length/2
aeron.publication.term.window.length
```

4.3 针对Aeron的重要参数

```
//Client连接Driver超时, 默认10秒
Aeron.Context.driverTimeoutMs()

//Publication连接超时,5秒内没收到Status Message(SM)为超时
Aeron.Context.publicationConnectionTimeout()

//CnC 文件夹位置
//需要和Driver的aeronDirectoryName一致, 以便进程间通信
Aeron.Context.aeronDirectoryName
MediaDriver.Context.aeronDirectoryName
```

4.4 扩展点参数

```
//指定发送端实现，默认SendChannelEndpoint
aeron.SendChannelEndpoint.supplier

//指定接收端实现，默认ReceiveChannelEndpoint
aeron.ReceiveChannelEndpoint.supplier

//多播流控策略，默认策略MaxMulticastFlowControl
aeron.unicast.FlowControl.supplier

//拥塞控制策略，默认是StaticWindowCongestionControl
aeron.CongestionControl.supplier
```

4.5 调优

```
// JVM参数调优
-XX:BiasedLockingStartupDelay=0 \
-XX:+UnlockDiagnosticVMOptions \
-XX:GuaranteedSafepointInterval=300000 \

// Aeron参数调优
-Dagrona.disable.bounds.checks=true \
-Daeron.term.buffer.sparse.file=false \
-Daeron.rcv.initial.window.length=8m \
-Daeron.mtu.length=16k \

// 网络参数调优
-Daeron.socket.so_sndbuf=8m \
-Daeron.socket.so_rcvbuf=8m \

// LowLatencyMediaDriver
 MediaDriver.Context ctx = new MediaDriver.Context()
.termBufferSparseFile(false).threadingMode(DEDICATED)
.conductorIdleStrategy(new BusySpinIdleStrategy())
.receiverIdleStrategy(new BusySpinIdleStrategy())
.senderIdleStrategy(new BusySpinIdleStrategy());
```

4.6 性能测试

```
// 针对AeronMessenger和JGroupsMessenger的性能测试, VM options:
-Djava.net.preferIPv4Stack=true
-Djgroups.bind_addr=192.168.1.203
-XX:BiasedLockingStartupDelay=0
-Daeron.mtu.length=16k
-Dagrona.disable.bounds.checks=true
-XX:+UnlockDiagnosticVMOptions
-XX:GuaranteedSafepointInterval=300000
//结果 aeron 比 jgroups 慢 3 倍左右
aeron.messenger.metric
             count = 10038230
         mean rate = 46221.26 events/second
     1-minute rate = 46310.66 events/second
     5-minute rate = 30777.44 events/second
    15-minute rate = 21112.53 events/second

jgroups.messenger.metric
             count = 30287971
         mean rate = 133901.00 events/second
     1-minute rate = 135765.93 events/second
     5-minute rate = 71567.89 events/second
    15-minute rate = 29768.04 events/second
```

5. Aeron 流控与拥塞控制

5.1 多播流控(Flow Control)

每有一个发布和订阅，那么接收消息的控制端(Conductor)会定期的发送Status Message(SM). 这条消息会携带已消费消息的streamId, offset以及recv window length, recv id等信息

在发送端(Sender), 会根据这个offset以及recv window length做流控保证消息发送的偏移量不大于offset + recv window length

在多播情况下，有可能有多个接收端(Receiver)
默认的MaxMulticastFlowControl在发送端(Sender)端采取的策略是记录多Receiver中最大的offset + recv window length为流控的偏移量

MinMulticastFlowControl则正相反，记录最小偏移量

5.2 拥塞控制(Congestion Control)

在默认的配置中
Congestion Control采用的是StaticWindowCongestionControl
即每次生成Status Message(SM)中使用固定的recv window length
效果等同于不做任何的拥塞控制

在可选的配置中，可以选择CubicCongestionControl
在发送消息和接收消息的控制端(Conductor)，会定期的发送RTT消息
来衡量网络状况，CubicCongestionControl利用此RTT时间会根据算法
生成一个动态的recv window length，并被SM消息携带发送给
接收端(Reciver)进行流控

相关引用: Cubic.pptx

6. Aeron 可靠性保证

6.1. 丢包重发送

在接收方的控制端(Conductor)
会定期的对term buffer根据当前读取位置扫描
如果发现扫描的Fragment间有不连续的情况
会延迟一小会并对发送源发送NAK消息
在发送源方的发送端(Sender)会对包进行重发送，
如果接收方的控制端(Conductor)还扫描不到重发送的包，则继续上述流程

在订阅端如果读到丢包的位置的话，是会再读这个位置直到丢包发送过来
但这个过程并不影响其他订阅，因为每个订阅有单独的term buffer

6.2 内存映射文件

在[3.9 发布以及取消发布事件]中我们提到过
每个发布都会在订阅端产生一个image logbuffer
这个image logbuffer的大小为 term length * 3
每个发布也会产生一个publication 的 logbuffer文件
大小也是term length * 3

那么举例如下 多播条件下4台机器每台机器有一个发布一个订阅
并且term length 设定为128MB
那么每台机器下的aeron文件夹结构如下

```
aeron-user
   |
   |------publications
   |          |
   |          |---------uuid-pub-1.logbuffer //128MB*3=384MB
   |
   |------images
              |
              |---------uuid-sub-1.logbuffer //128MB*3=384MB
              |---------uuid-sub-2.logbuffer //128MB*3=384MB
              |---------uuid-sub-3.logbuffer //128MB*3=384MB
              |---------uuid-sub-4.logbuffer //128MB*3=384MB
```

6.3 一些需要注意的问题

消息长度限制最大16MB

默认Aeron 允许发送的最大消息长度为Math.min(term length/8, 16mb);
超过以上大小的消息Aeron推荐我们分chunk发送, 在接收端自己组装消息
如果想要发送16MB的消息, 需要设置term length为128MB

垃圾文件清理

前文[6.2 内存映射文件]讲过每一个发布
在订阅端都会形成一个term length*3的映射文件

我们假设使用embedded media driver, term length设置为128MB
多播情况下单机启动10个driver和client, 那么耗费的硬盘空间如下:
11 * 128MB * 3 * 10 = 42240MB = 42G

那么势必我们要在启动前通过脚本清理之前的aeron文件夹

```
// 错误的写法
while(running.get()) {
  strategy.idle(
    sub.poll(new FragmentAssembler((buf, off, len, head) -> {
      byte[] payload = new byte[len];
      buf.getBytes(off, payload);
      onMessage(payload);
    }), 10)
  );
}
// 正确的写法
FragmentAssembler h = new FragmentAssembler(
  (buf, off, len, head) -> {
    byte[] payload = new byte[len];
    buf.getBytes(off, payload);
    onMessage(payload);
  }
);

while(running.get()) {
  strategy.idle(sub.poll(h, 10));
}
```

THANK YOU!