## 测试原理

我们的目标是测试出在不同类型的服务器（比如I3，I7等服务器）上，KUP的Core Engine最多能支持多少个Node4设备（Node4在给Core Engine发送4路视频流的同时，Platform也在从Core Engine拉取这些视频流进行查看），如下图：



我们使用Node4模拟器来模拟多个Node4设备进行测试；用Client Simulator模拟客户端（Platform）同时看视频流的场景

## 服务器准备

本文档中准备了三台服务器，1台运行Device Server & Streaming Server，1台运行Node4模拟器（也叫做h264服务）， 1台运行Arbiter服务和Client 模拟程序（因为Arbiter占资源较少，所以可以将这两个部署在一起）

本文中服务器分配如下：

10.101.10.210作为Arbiter服务器

10.101.0.11作为Device Server和Streaming Server的服务器

10.101.0.10作为Node4模拟器

## 服务器安装配置

在测试之前所有服务器都需要预安装一些软件，软件安装方法请参考《ALIFF-KAIUP-CORE-Ubuntu-deployment-guide.docx》

作为Arbiter的服务器要安装JDK、MySQL等软件

作为Device Server和Node4模拟器的机器要安装ffmpeg、libevent、thrift、zmq和glog等软件

### 3.1 配置准备

解压测试软件包到用户家目录下，比如/home/kaiadmin/。这样所有脚本和配置文件都会在deploy-package文件夹下

在进行性能测试前需要三个主要步骤：

(1). 修改配置文件：里面会设置各服务器的IP等

(2). 服务部署：将Arbiter、Device Server和Node4模拟器分别部署到相应的服务器上

(3). 添加设备: 添加测试所需要的设备

(4). 服务启动：启动各个服务

### 3.2 修改配置文件

~/deploy-package/stress-test.conf是主配置文件

下面对文件里需要修改的项进行解释：

arbiterHost Arbiter服务器IP，就是本机IP

arbiterUserName Arbiter服务器上测试用的账户，比如kaiadmin

mysqlHost MySQL服务器所在的IP（一般情况下MySQL服务器会和Arbiter部署在同一台机器上，如果MySQL和Arbiter在一台机器，配置成127.0.0.1；如果不在，配置成具体IP）

mysqlDBName MySQL数据库名

mysqlPwd MySQL密码

wholeDeviceNum 设备的数目（就是此次性能测试你想测试多少设备，比如测试200个设备的情况）

eachDsDeviceNum 每个Device Server能支持的设备数量（因为Device Server能支持的设备数量有限）

dsNumOnThisMachine 某一台Device Server服务器上要部署多少个Device Server进程（一般情况下将这个值设置为20，就是说在每个Device Server服务器上部署20个Device Server，但是不一定就会运行20个，也可以只运行其中的几个。这个需要运行的Device Server的数目是程序根据设备总数量和每个Device Server上添加的设备数自动计算出来的）

h264Host Node4模拟器服务所在的服务器IP

h264Rate Node4发送视频的帧率（默认是15，一般不用修改）

h264UserName Node4模拟器服务器上面测试用的账户

dsMachineNumber 想要部署几台服务器作为Device Server服务器（如果要测试的设备数目太多，而一台服务器上能运行的Device Server和每个Device Server能支持的设备的数目都是有限的。所以如果要测试的设备数目太多的话，就需要有多台服务器作为Device Server的服务器）

注意：一旦在部署的时候部署成只有一个ds machine，就不能修改dsMachineNumber了

以下区域里面，如果只有一台服务器作为Device Server服务器，只需配置ds--machine 1，否则，需要配置多个

---------------------------------------------------------------------------------------------------------------------------------

#ds--machine 1

dsUserName=kaiadmin Device Server服务器上测试用的账户名称

dsHost1=10.101.0.11 第一台Device Server服务器的IP

dsNeeded1=2 第一台Device Server服务器上运行几个Device Server（如果只有一个Device Server服务器这个不用配置，程序会自动计算）

#ds--machine 2

dsHost2=10.101.0.106 第二台Device Server服务器的IP

dsNeeded2=2 第二台Device Server服务器上运行几个Device Server

#ds—machine 3

…

注意:

如果有多个Device Server服务器，那么必须满足以下公式

wholeDeviceNum / eachDsDeviceNum = dsNeeded1 + dsNeeded2 + …

如果只有一个Device Server服务器，dsNeeded1的值不用设置，他会自动设置为dsNeeded的值

---------------------------------------------------------------------------------------------------------------------------------

### 2.3 SSH登陆配置

因为在部署服务和之后的运行服务时都会涉及到从Arbiter所在的服务器SSH免密码登陆到Device Server和Node4模拟器服务器上面。所以我们需要先配置SSH的免密码登陆：

假设Arbiter、Device Server、Node4模拟器所在的服务器上我们使用的测试账户都是kaiadmin，那么我们就需要在Device Server和Node4上面配置SSH的key，以允许Arbiter上的kaiadmin用户能免密码登陆到这两台服务器上

SSH免密码登陆请参考：<http://blog.csdn.net/lin_fs/article/details/7309714>

### 2.4 部署服务

将3个服务部署到各自的服务器上

cd ~/deploy-package

sh deploy-all.sh

查看运行日志，看有没有什么错误，如果出现“deploy succeed”，说明部署成功

注意：

deploy-all.sh会将三个服务运行所需要的脚本、配置文件等拷贝到各自服务器的/home/kaiadmin/目录

clean-all-deploy.sh 会清除所有的部署，即将各服务器上部署的服务的文件都删除掉

部署成功后，查看以下文件夹：

在Arbiter 服务器的/home/kaiadmin/下会有stress-test-arbiter文件夹

在Device Server服务器的/home/kaiadmin/ 下会有stress-test-ds文件夹

在Node4模拟器服务器（h264服务）的/home/kaiadmin/ 下会有stress-test-h264文件夹

### 2.5 添加设备

因为压力测试是要测试Arbiter和Device Server的承受能力，所以需要添加大量的设备；由于没有这么多真实设备，所以需要用模拟设备来进行测试。

可以直接使用将sql文件导入的方式添加设备到数据库里面。比如直接将juzz4v2-5K-kainodes.sql 导入到名为“juzz4v2”的数据库里面。这样就添加了5000个Node4的模拟设备到数据库了。

### 2.6 运行服务

cd ~/stress-test-arbiter/

run-all.sh 该脚本用于运行所有服务

close-all.sh 用于停止所有服务

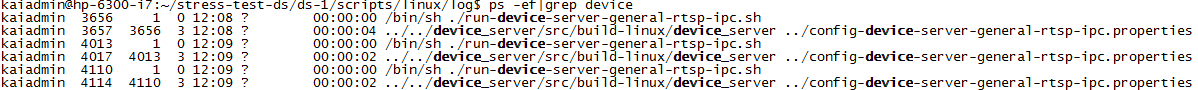
### 2.7 查看服务运行情况

以下命令都是在各服务各自的服务器上执行的，

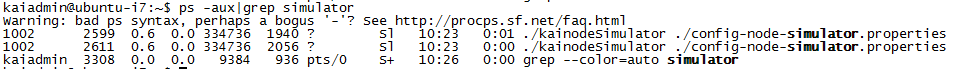
查看Arbiter服务：ps –ef|grep java，输出如下



查看Device Server服务： ps –ef|grep device，输出类似下面



查看Node4模拟器服务： ps –ef | grep simulator



## 测试点

Device Server主要有三个功能：

1. 管理真实设备。在压力测试的时候，配置文件中的wholeDeviceNum设备数量就可以认为是要添加到Device Server服务上面的所有设备。Arbiter会将这些设备分配给Device Server服务器上面的这些Device Server服务。
2. 接收Node发送的视频流。因为我们做压力测试的时候没有这么多的真实Node设备，所以我们用Node模拟器来模拟真实设备。一个Node4模拟器会对应一个Device Server，给这个Device Server发送4路视频数据。
3. 对外提供视频流，以供Platform进行获取。即我们可以使用Client模拟器来模拟从Platform同时查看视频流的操作。

由于Device Server有上面几个功能，所以Device Server服务所在的服务器压力就会比较大。所以我们压力**测试的目的**就是要测试出Device Server在兼具以上几个功能的时候，**每台Device Server服务器上面能运行的Device Server服务的数量和每个Device Server服务上面能运行的设备数量**

所以我们主要有以下几个测试点：

### 3.1 查看Device Server从Node模拟器接收视频流的情况

当执行完run-all.sh之后，Arbiter就会将所有设备添加到Device Server上，同时Node4模

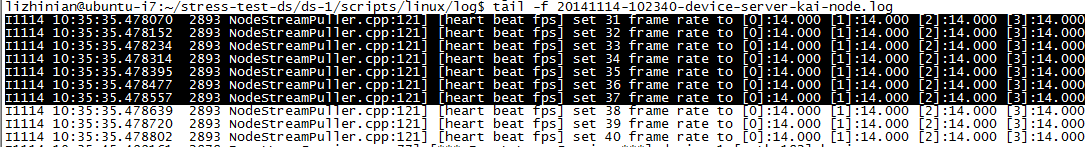
拟器会发送视频流到Device Server。查看此时Device Server的运行情况：

**1). 查看Device Server是否重启**

到Device Server服务器的~/stress-test-ds目录，执行sh check-if-ds-restarted.sh。如果输出中有类似”RESTART…”的语句， 说明Device Server有重启。如果有重启，说明设备数量超过Device Server的极限，请减少设备数量后重新测试。

**2). 查看Device Server收到的帧率情况**

分别到每个Device Server的目录下，查看Device Server从Node得到的帧率。如果比我们设置的帧率小太多(比如设置的是15，但这里得到的小于10)，减少设备数量后重新测试。



**3). 查看Device Server上的负载情况**

用htop查看服务器的CPU和内存占用情况。如果CPU和内存占用过多，减少设备数量后重新测试。

如果以上三个检查项都正常，请继续往下执行。

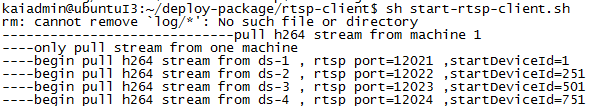
### 3.2 模拟Platform从Device Server拉取视频码流，查看是否有丢帧

此操作就是模拟Platform从Device Server拉取视频码流的操作：

1. 开始从Device Server拉取视频流

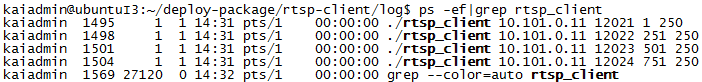
登录到Arbiter服务所在的机器上，切换到~/deploy-package/rtsp-client下面，运行start-rtsp-client.sh， 显示以下内容说明开始拉取视频码流：

(因为Arbiter服务所在的服务器压力较小，所以我们就模拟Platform和Arbiter在一台机器上运行时Platform从Device Server拉取视频流的情形。以下这个start-rtsp-client.sh的作用就是模拟Platform从Device Server拉取视频流)



查看拉取码流的客户端运行情况

ps -ef|grep rtsp\_client ，输出结果



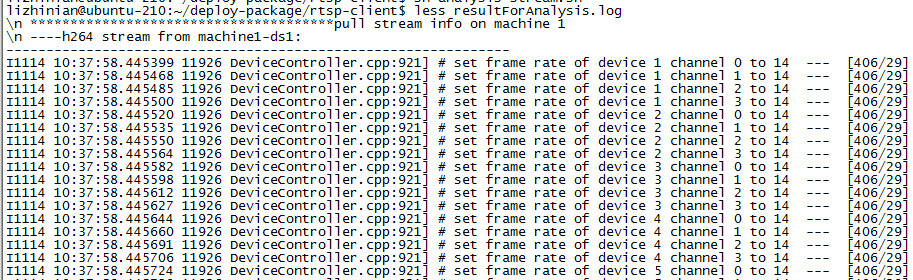
1. 视频码流获取情况查看

运行analysis-stream.sh，该脚本会分析从每一个Device Sever上拉取的视频码流的数据，并将分析结果写到resultForAnalysis.log日志里面

（因为从每一个Device Server拉取到的视频码流都会写到一个单独的日志文件里面，要是有20个Device Server的话，我们就需要看20个日志文件，以便查看从每个Device Server上拉到的码流是否有丢帧，这样效率太低了。所以analysis-stream.sh这个脚本就是用来分析从每个Device Server上面拉取到的视频流的情况，只将分析结果打出来写到resultForAnalysis.log文件里）

查看该日志文件，我们会看到从Device Server上拉取的每一个设备的视频码流的情况。文件内容类似如下所示，其中0 to \*\*，to后面的数值就是从Device Server拉到的实际帧率。每一个Device的channel 0 1 2 3分别表示了Node4设备上的四个摄像头的帧率。

（这个文件里面” ----h264 stream from machine1-ds1”以下的行就表示从第一台Device Server服务器的第一个Device Server服务上面拉取到的帧率情况； “----h264 stream from machine1-ds2” 这个表示第一台Device Server服务器上面的第二个Device Server服务….依次类推）



注意：

运行start-rtsp-client.sh之后，就开始从Device Server拉取视频码流，但是刚开始拉取的时候，可能Device Server的视频流尚未准备好。 所以如果我们立马执行analysis-stream.sh的话，查看resultForAnalysis.log日志可能会发现不是像以上截图中的样子，或者显示的帧率都是”0 to 0 ---”。

因此我们可以在运行start-rtsp-client.sh之后，等1分钟左右执行analysis-stream.sh；或者是隔十几秒多执行几次analysis-stream.sh， 之后再查看resultForAnalysis.log日志，看帧率情况。

1. 视频帧率分析

----如果从resultForAnalysis.log中看到拉取到的帧率和我们在配置文件里面配置的有较大出入，可以通过如下步骤查找原因：

(1). 登录到Device Server 服务器，查看该服务器的CPU、内存占用情况。查看Device Server是否有重启，查看Device Server从Node得到的帧率。

(2). 登录到Node模拟器服务器上，查看CPU和内存占用情况。

如果两台服务器上面的资源占用都不大但是帧率异常的话，就找开发人员定位。

如果资源占用快到服务器的极限了的话，就说明Device Server服务器能承载的Device Server数量已经到极限了。我们测出了在该帧率下服务器的上限。

如果资源占用没有到极限且帧率和我们设置的差不多，说明服务运行良好。这时候可以关闭所有服务，增加设备数量之后再继续测试服务器上限。

1. 关闭拉取码流的客户端请执行stop-rtsp-client.sh
2. 服务器资源状况查看

服务器资源主要看： CPU、内存

--CPU查看：使用htop查看，如果发现服务器每个核的CPU占用率都在95%以下的话， 说明CPU占用正常，如果超过了的话，说明资源比较紧缺

--内存查看：使用htop，如果”Mem”中显示已经使用的内存占用基本逼近总内存，或者Swp使用也非常多的话，说明内存已到极限了

1. 测试报告数据

这个只需要在找到某个Device Server的性能上限时才需要书写。需要书写成《Core-Engine-Performance-Testing》中表格的样子，以供销售人员进行服务器性能参考。