# JStorm 调度测试

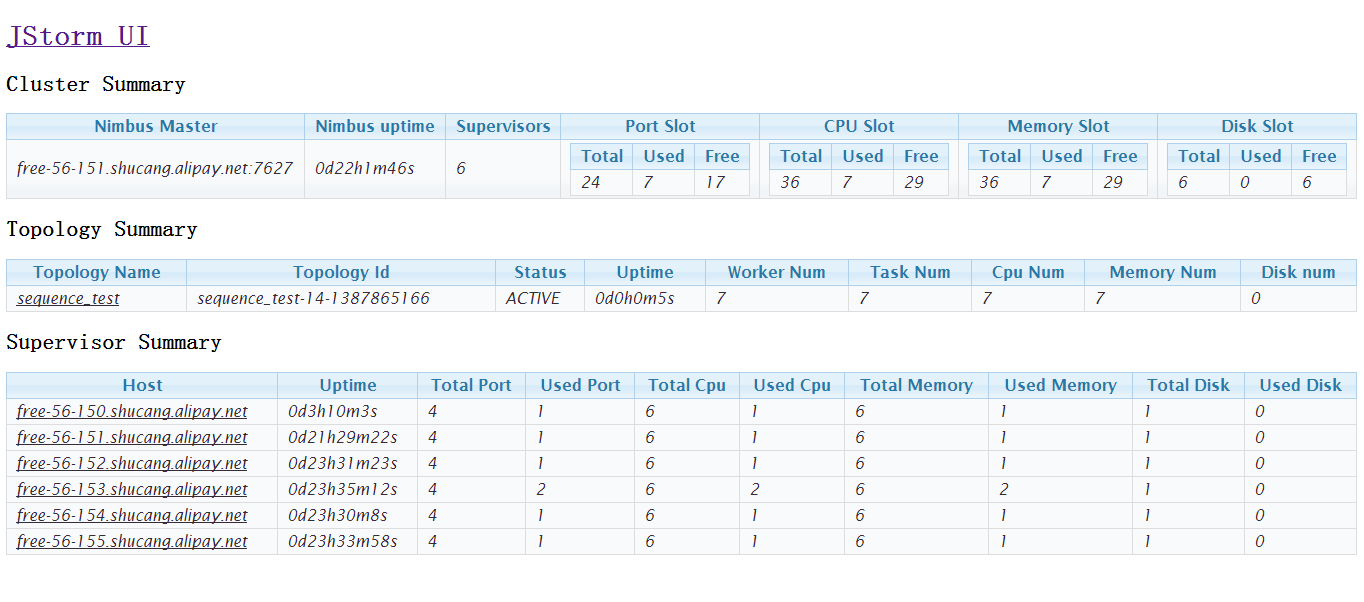
## 最普通分配， 所有资源足够

### 测试目标

测试是否支持资源分配， 检测worker分布是否均匀

### 测试结果

支持资源分配， worker分布均匀



## 设置CPU\_SLOTS\_PER\_TASK

### 测试目标

检测是否支持CPU\_SLOTS\_PER\_TASK

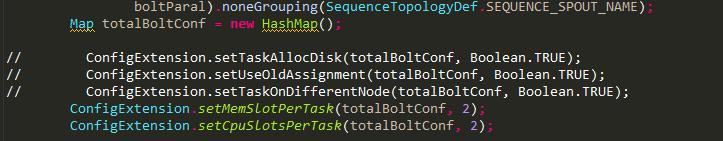
检测web， 查看对应的worker或supervisor是否cpu计数有误。

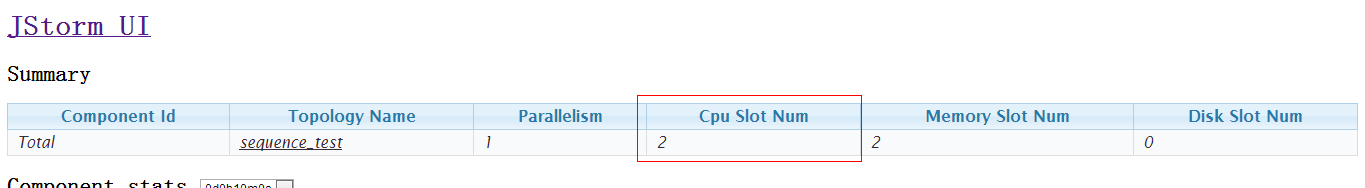
### 测试结果：

通过

对应component 申请到正确的task数

源码





## 设置MEM\_SLOTS\_PER\_TASK

### 测试目标

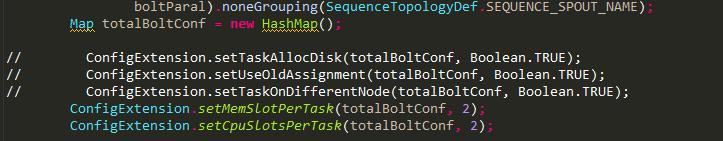
检测是否支持MEM\_SLOTS\_PER\_TASK

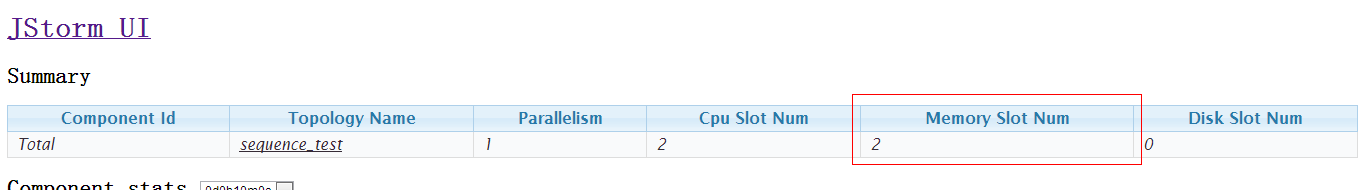
检测web， 查看对应的worker或supervisor是否memory计数有误。

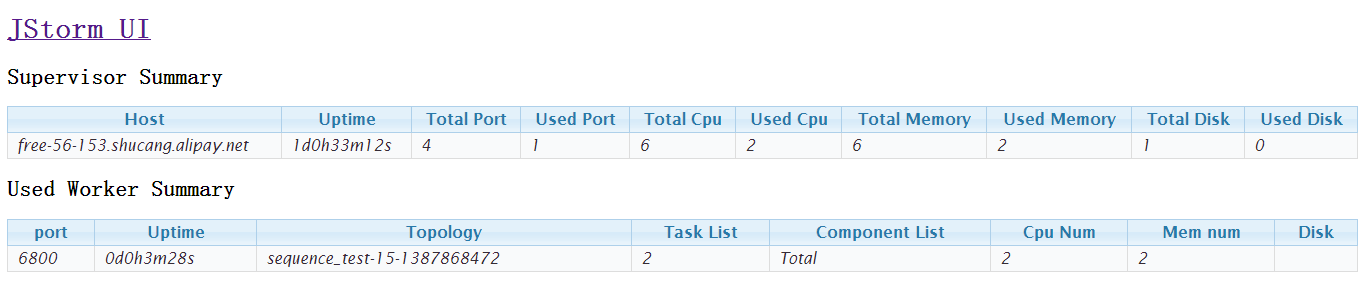
### 测试结果：

测试通过， 对应component申请到正确的内存slot

源码：









## 设置TASK\_ALLOC\_DISK\_SLOT

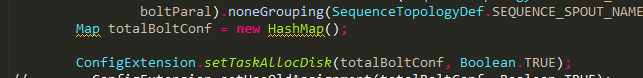
### 测试目标

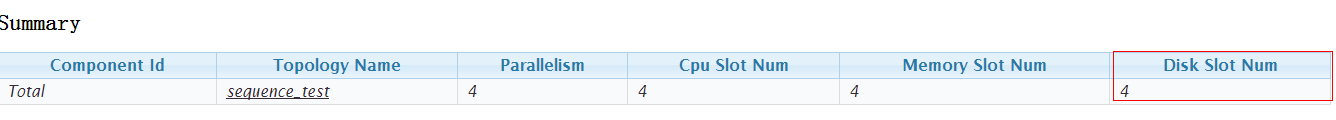
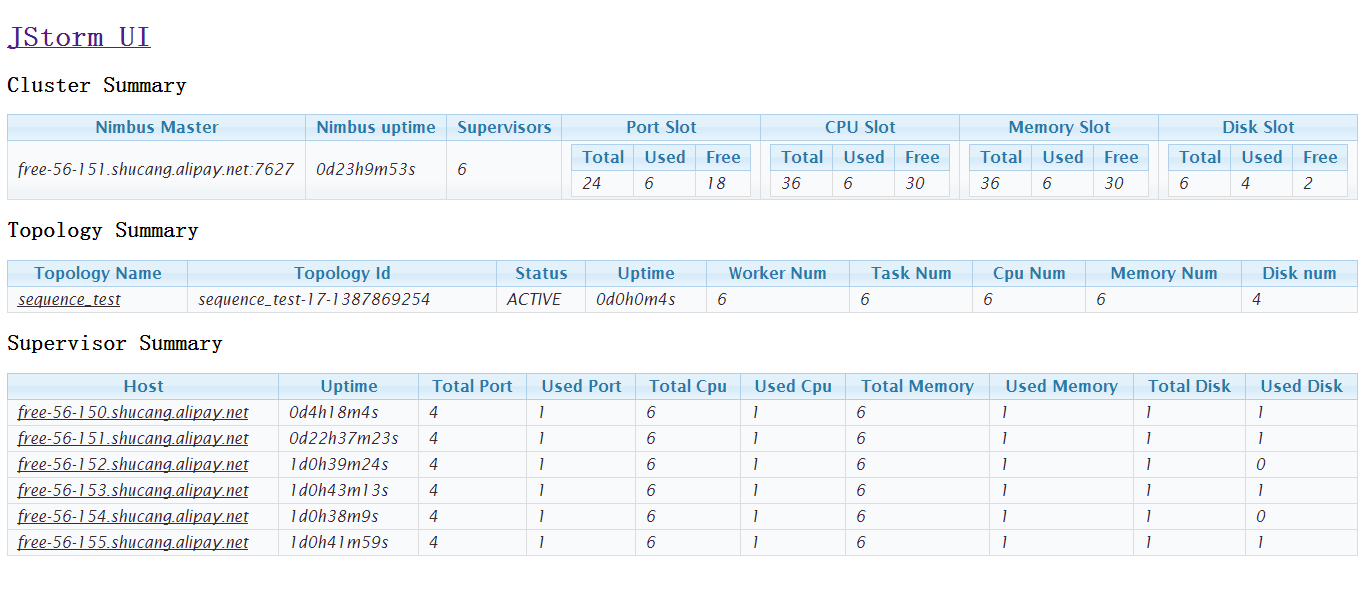
检测是否支持TASK\_ALLOC\_DISK\_SLOT

检测web， 查看对应的worker 或supervisor 是否有使用disk

是否task\_on\_different

### 测试结果：





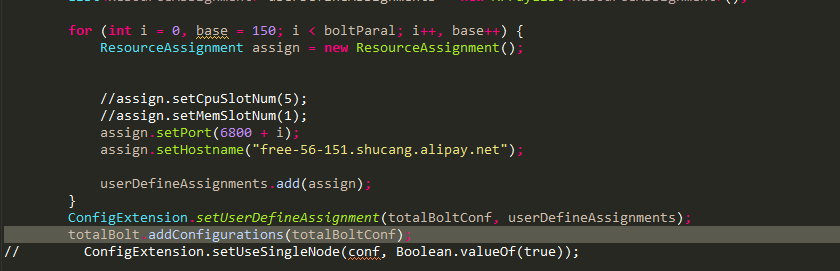
并且自动做了TASK\_ON\_DIFFERENT\_NODE

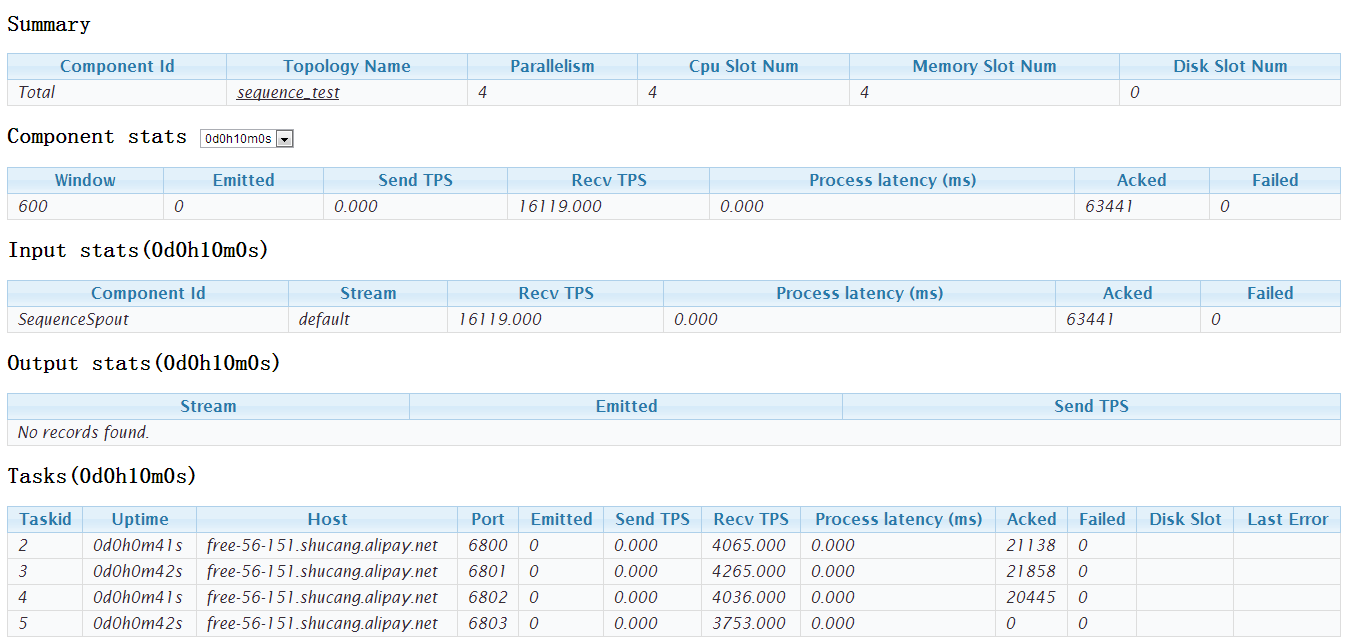
## 是否支持USE\_USERDEFINE\_ASSIGNMENT， 并且指定worker资源是足够的

### 测试目标

USE\_USERDEFINE\_ASSIGNMENT 配置正确， 检测是否分配到指定的worker上面去

### 测试结果：



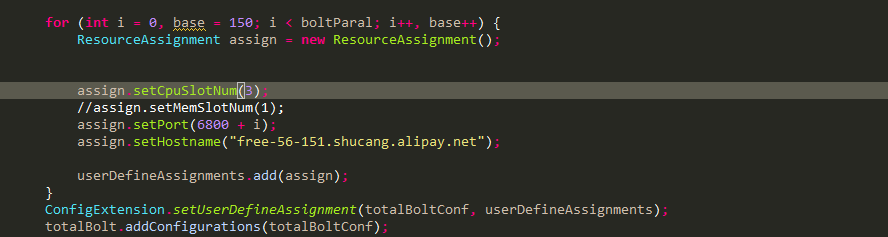


## 是否支持USE\_USERDEFINE\_ASSIGNMENT， 指定worker资源是不够的，

### 测试目标：

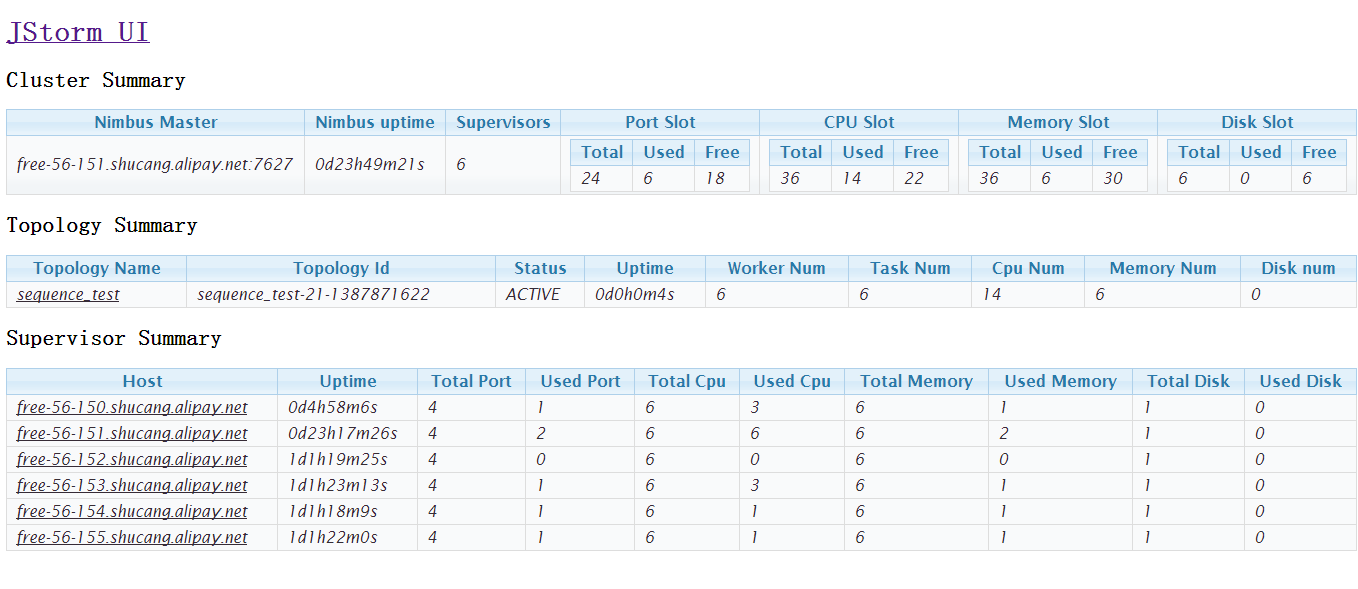
检测有没有worker代替了指定worker

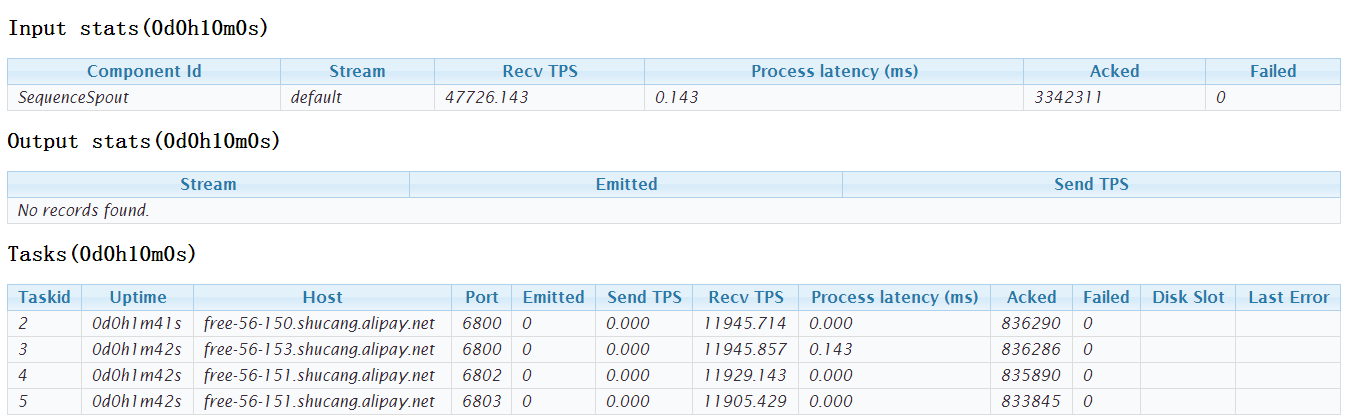
### 测试结果：



使用free-56-151.shucang.alipay.net port 6800 ~ 6804

但是CPU资源不够





## 是否支持USE\_OLD\_ASSIGNMENT， 第一次使用

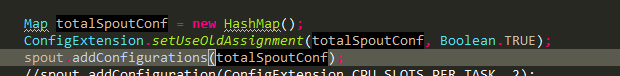
### 测试目标：

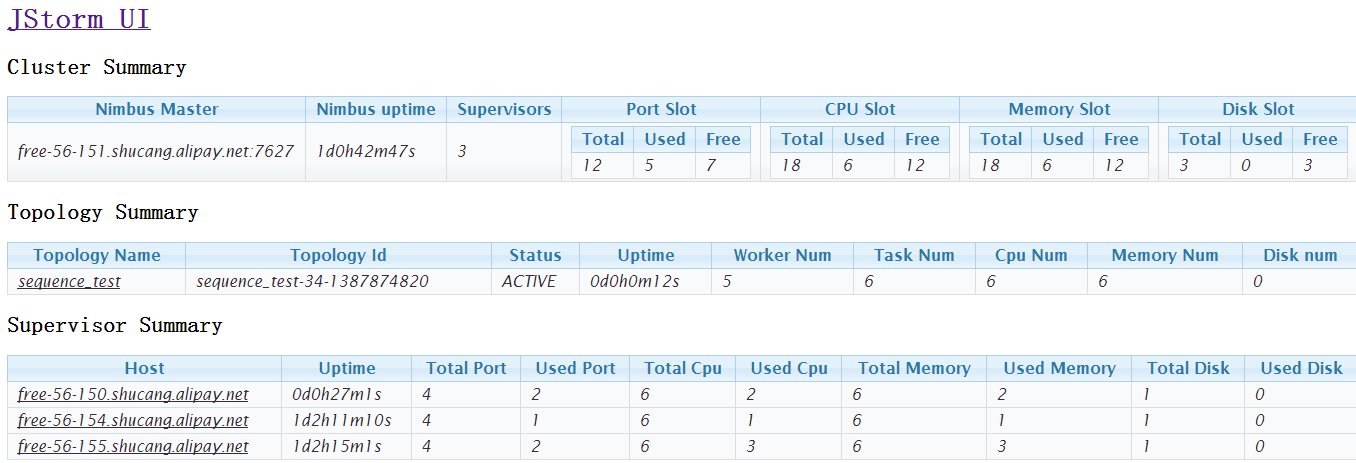
检测是否是按资源分配算法进行分配

### 测试结果：

通过







## 是否支持USE\_OLD\_ASSIGNMENT， 已经使用过一次

### 测试目标

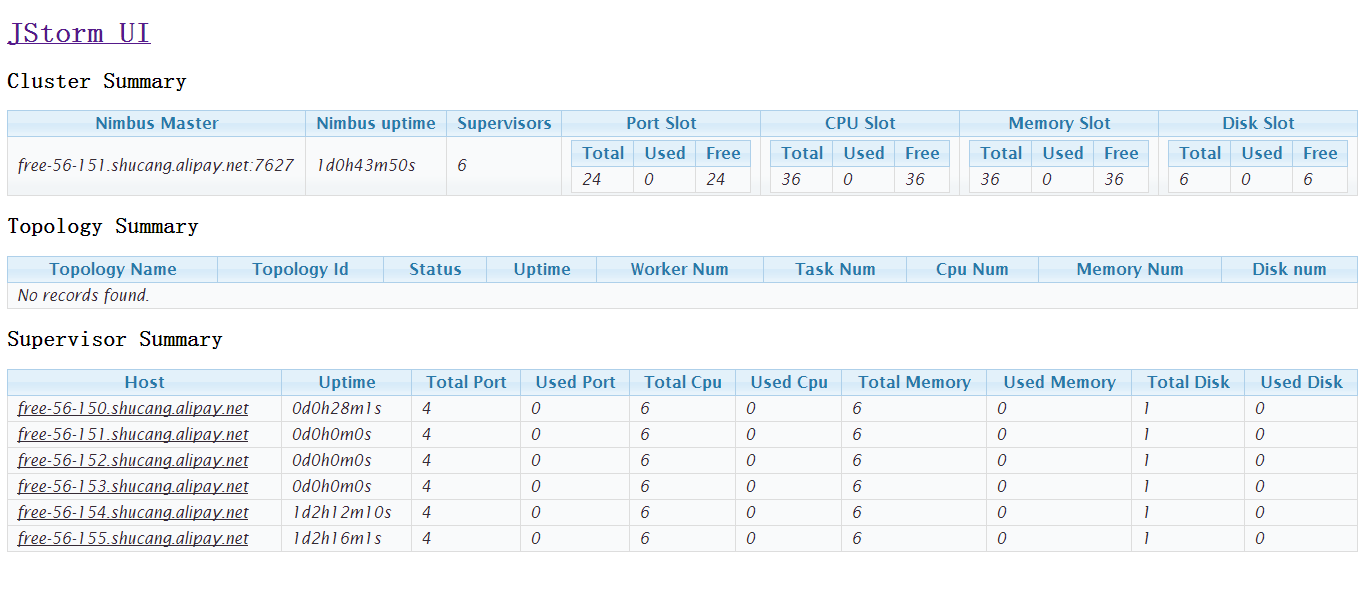
并且老的资源还未被使用， 是否分配到老的资源上去

### 测试结果：

通过

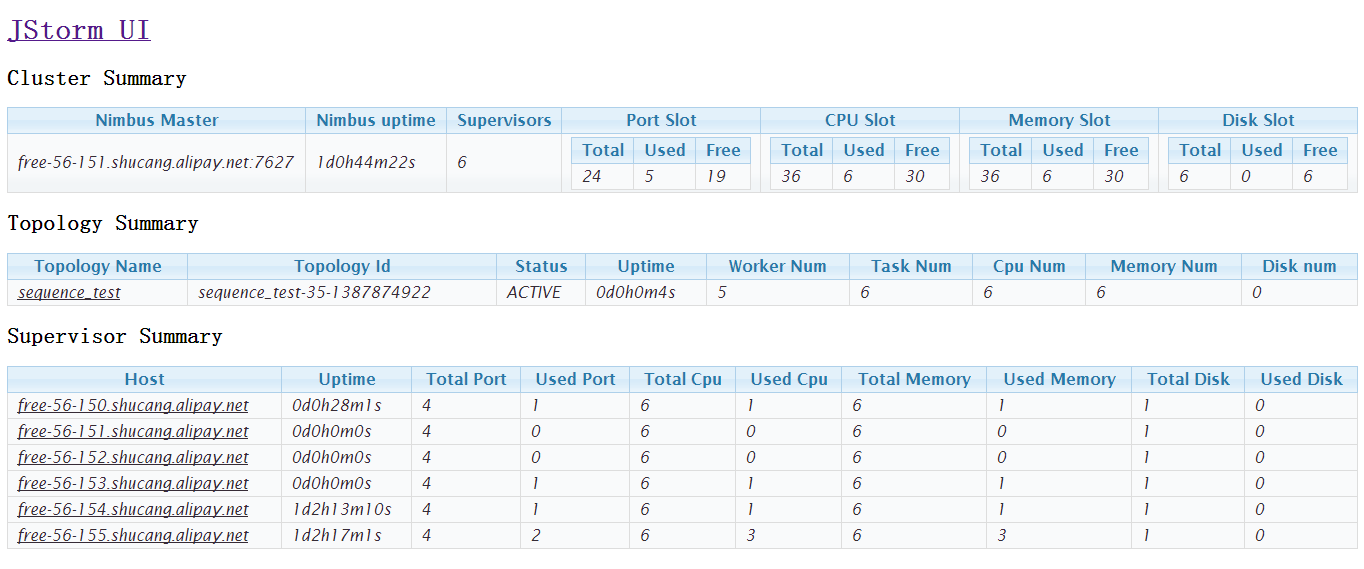
在[“是否支持USE\_OLD\_ASSIGNMENT， 第一次使用”](#_是否支持USE_OLD_ASSIGNMENT，_第一次使用)测试基础上进行测试

增加supervisor



提交jar

继续使用老的资源



## 是否支持USE\_OLD\_ASSIGNMENT， 老资源已被占

### 测试目标

并且老的资源已被使用， 是否分配到新的资源上去

### 测试结果

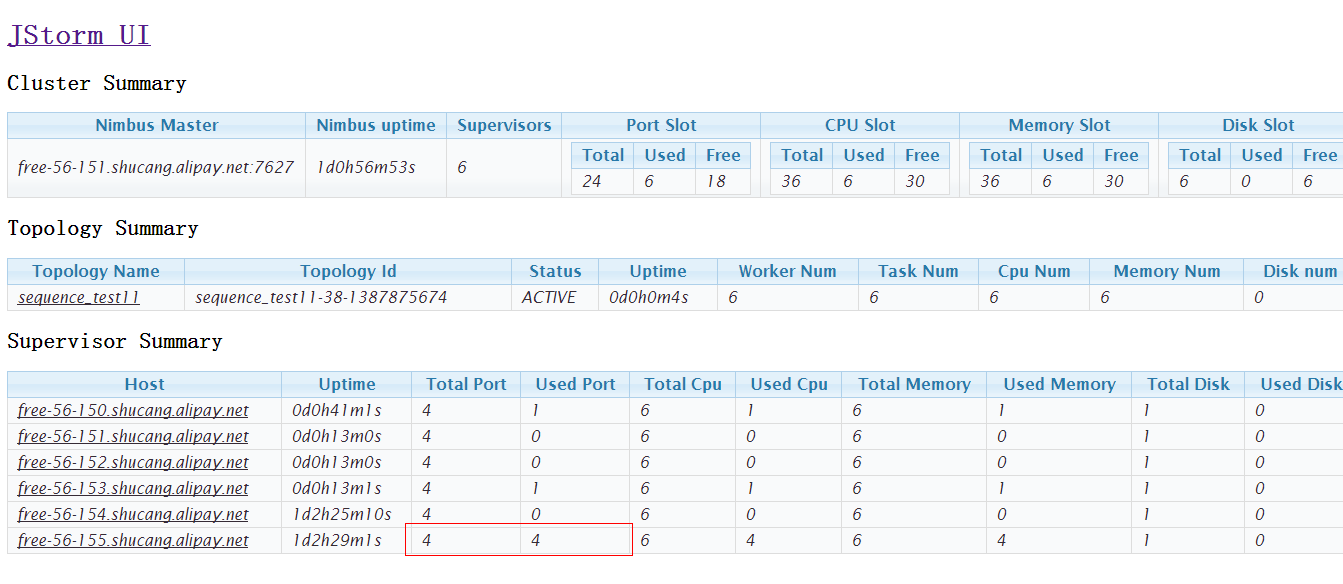
通过

继续上一轮测试[“是否支持USE\_OLD\_ASSIGNMENT， 已经使用过一次”](#_是否支持USE_OLD_ASSIGNMENT，_已经使用过一次)

杀死topology

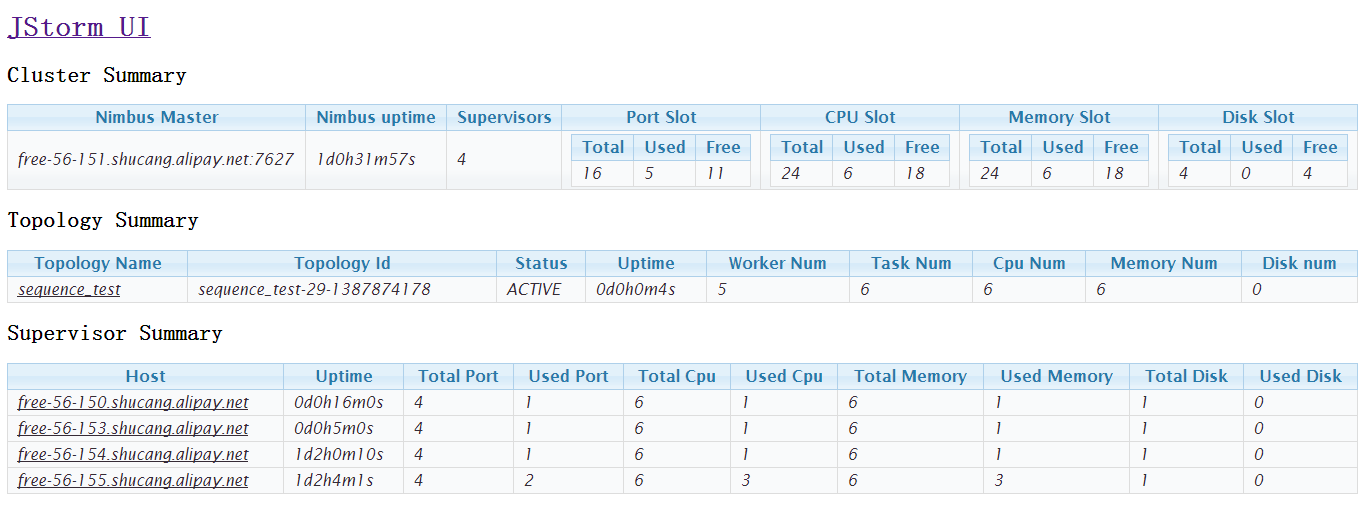
启动另外一个topology，占用端口155上的端口

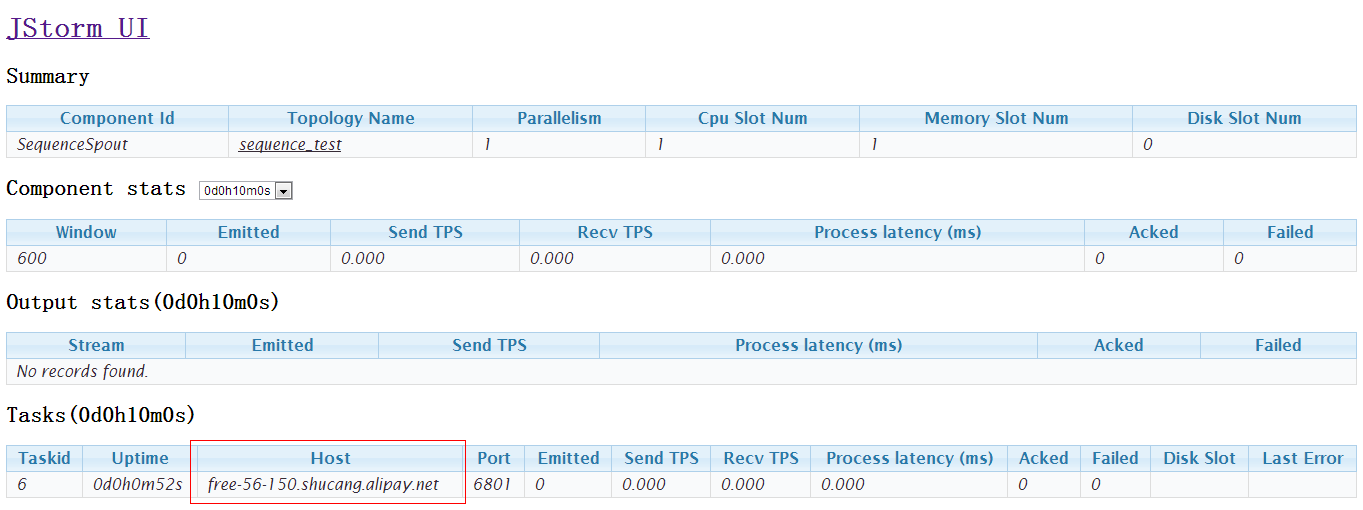
在测试中，4个端口全部被占

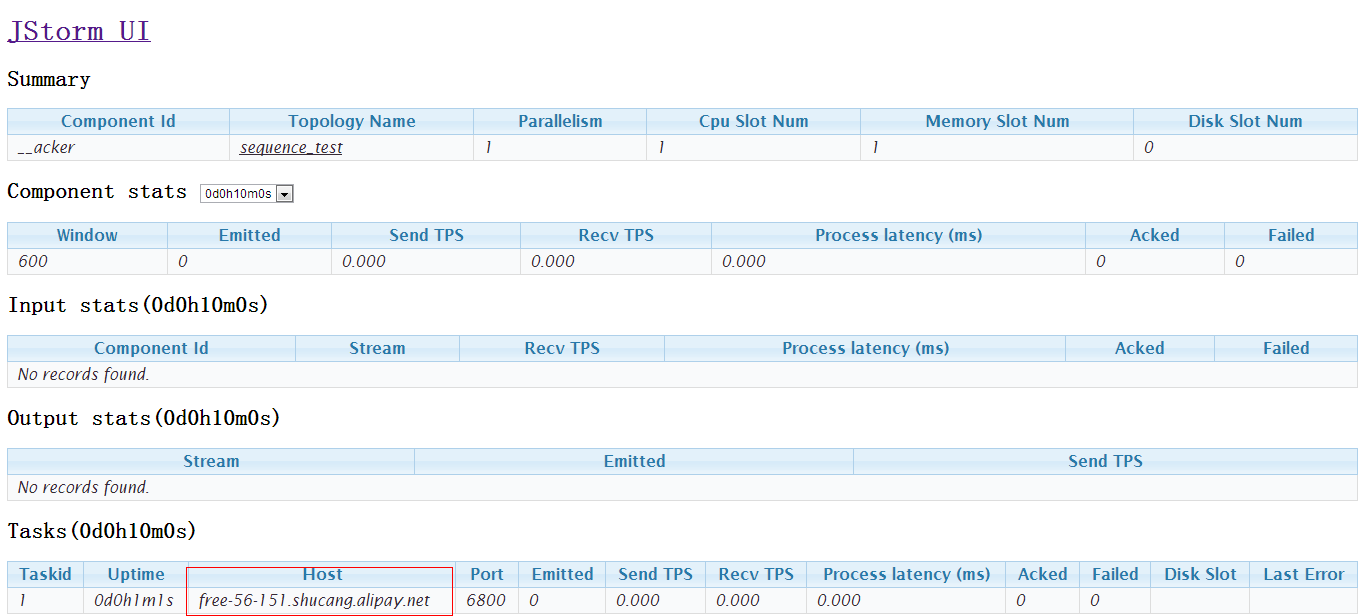


再次启动sequence\_test\_old\_assign

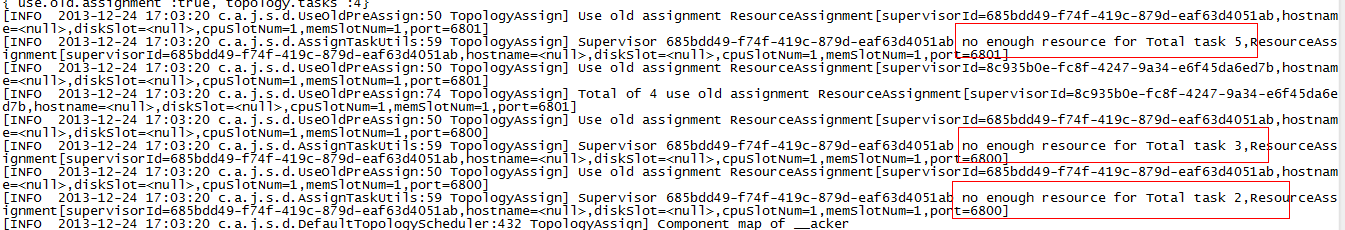
全部分配新的资源







nimbus log显示老的resource被占用

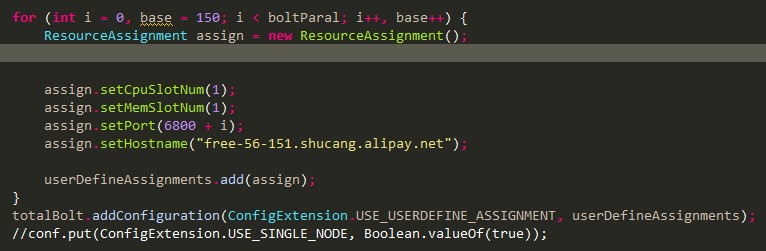


## 使用USE\_USERDEFINE\_ASSIGNMENT， 杀死worker，

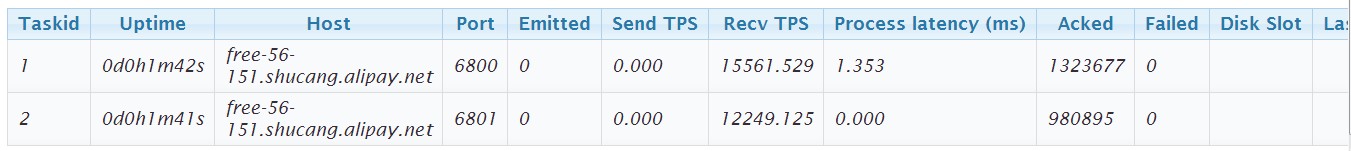
### 测试目标

检测worker被杀死后会不会重新分配一个新的worker

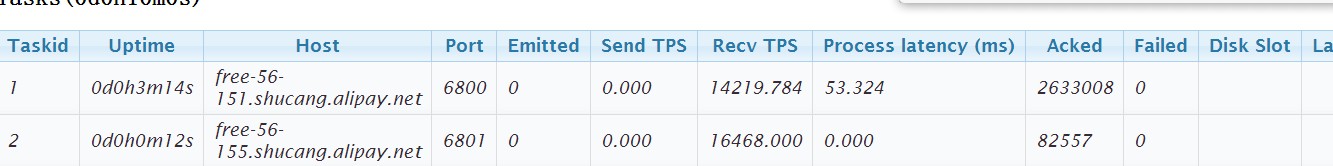
### 测试结果：



正常分配:



杀死之后重新分配:



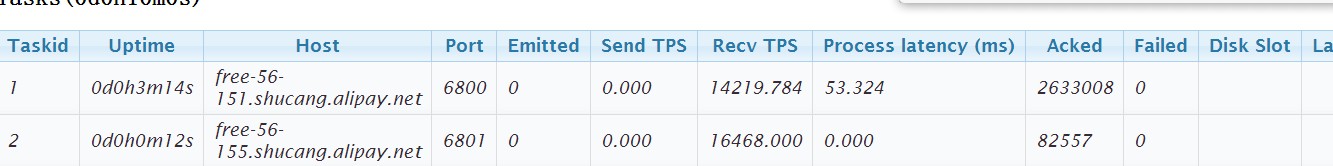
## 使用USE\_USERDEFINE\_ASSIGNMENT， 杀死worker后 等分配好新的worker后，再杀死这个worker，

继”使用USE\_USERDEFINE\_ASSIGNMENT， 杀死worker”测试之后

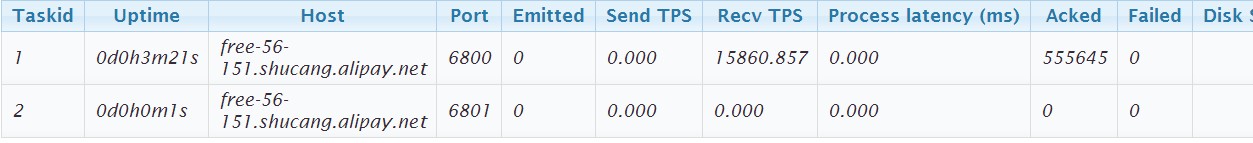
### 测试目标

杀死新分配的worker后,按照指定的assignment重新分配worker

### 测试结果：



杀掉155上的worker后重新分配到了151



## 使用USE\_OLD\_ASSIGNMENT， 杀死worker后

### 测试目标

检测有没有新的worker替代它

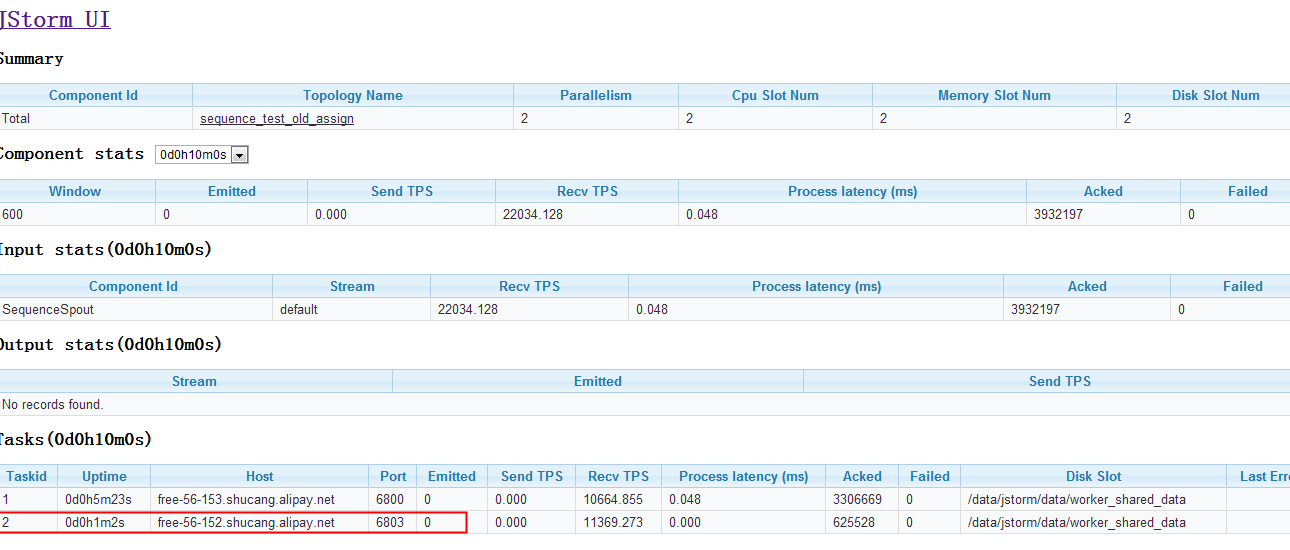
### 测试结果：

测试通过

在“是否支持USE\_OLD\_ASSIGNMENT， 老资源已被占”基础上进行本轮测试，



杀死的worker已经被更改



## 同一个component 同时使用了USER\_DEFINE\_ASSIGNMENT, USE\_OLD\_ASSIGNMENT

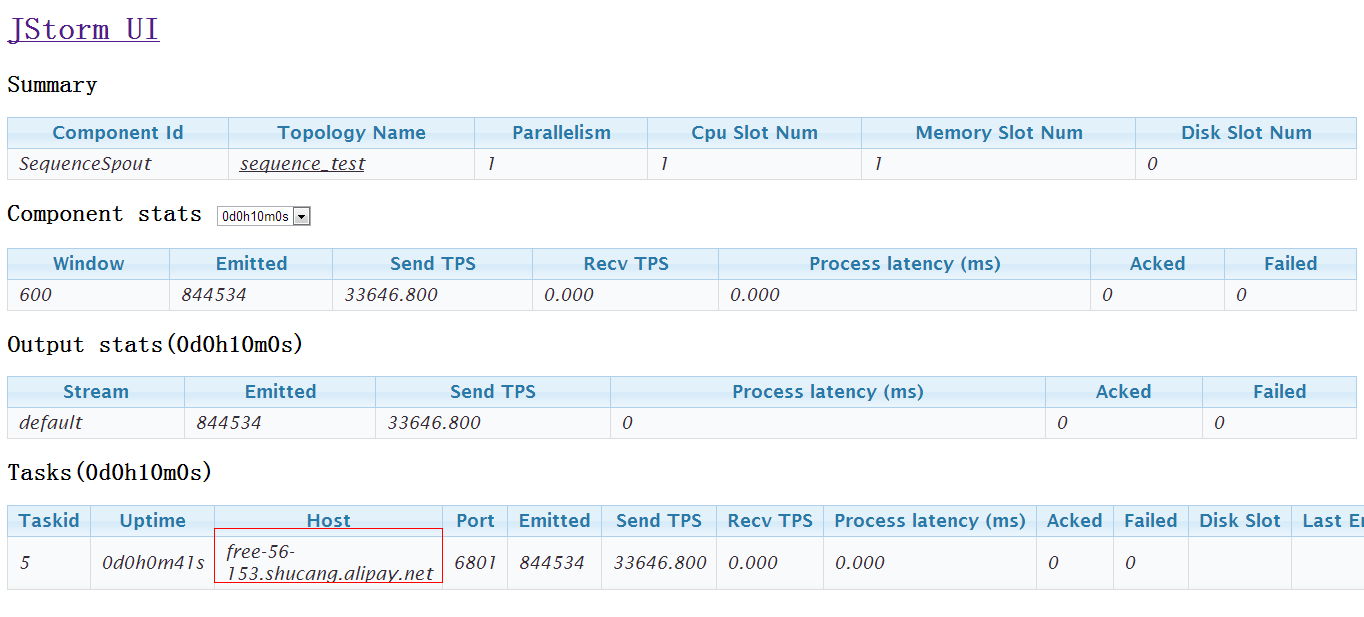


测试目标：

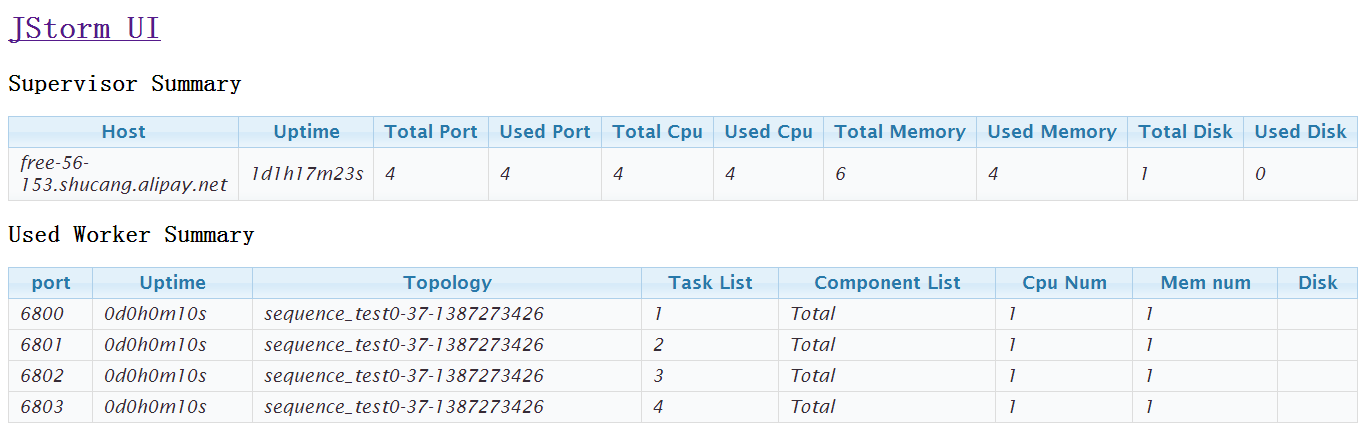
4 种case

1. **USER\_DEFINE\_ASSIGNMENT 满足， USE\_OLD\_ASSIGNMENT 不满足**

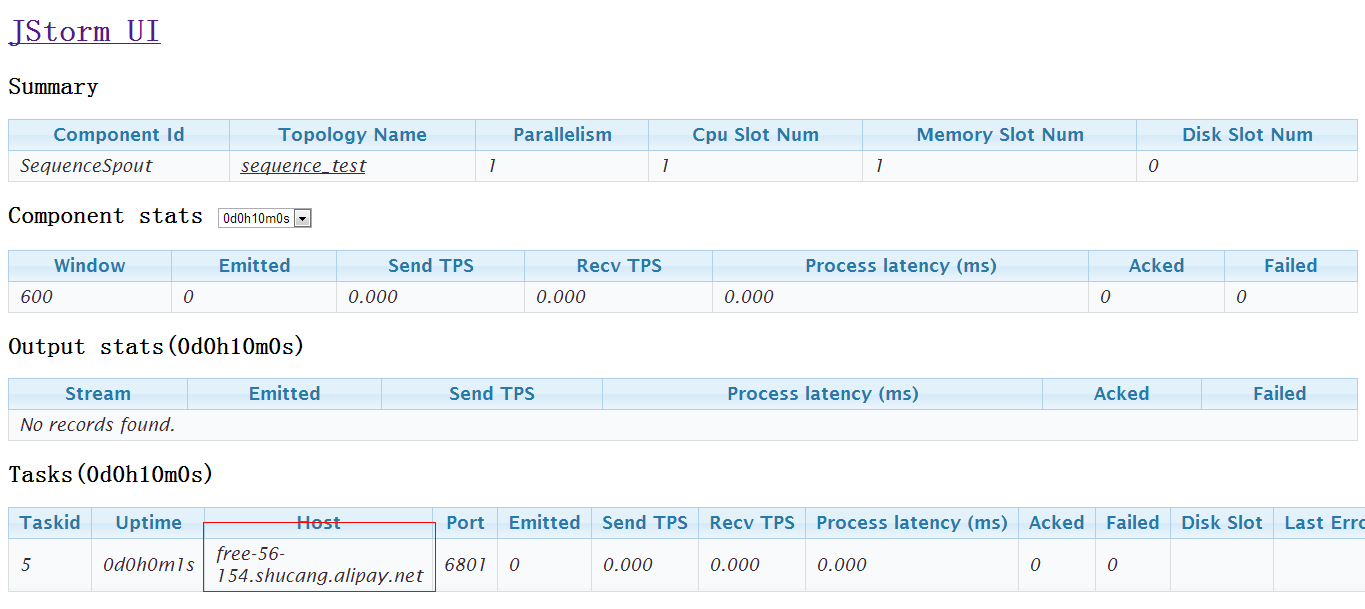
第一次提交sequence\_test时spout的位置:



杀掉sequence\_test,启动sequence\_test0将153的slot占满:

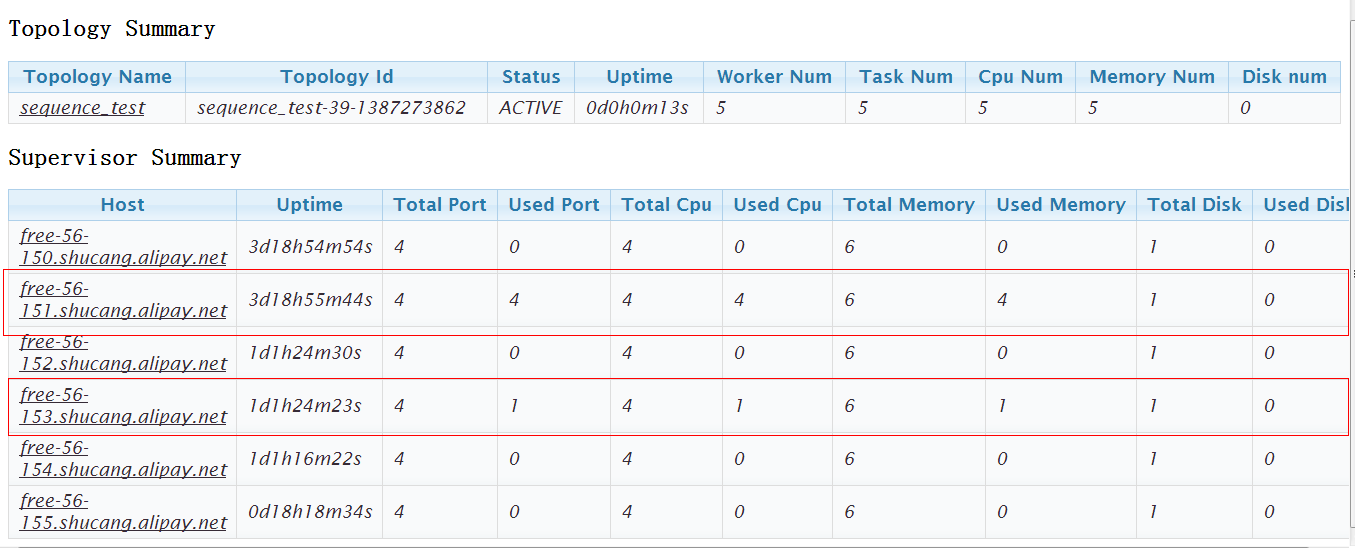


此时重新启动sequence\_test USER\_DEFINE\_ASSIGNMENT 满足,USE\_OLD\_ASSIGNMENT 不满足,spout被分配到其他机器上:

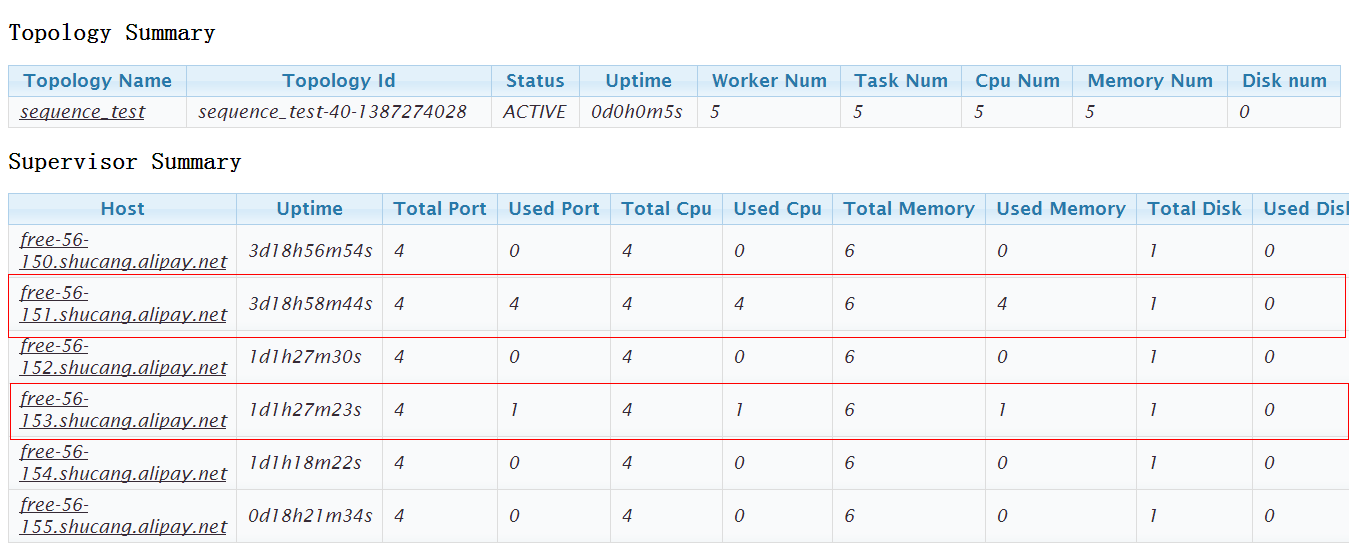


1. **USER\_DEFINE\_ASSIGNMENT 满足， USE\_OLD\_ASSIGNMENT 满足**

第一次分配情况:



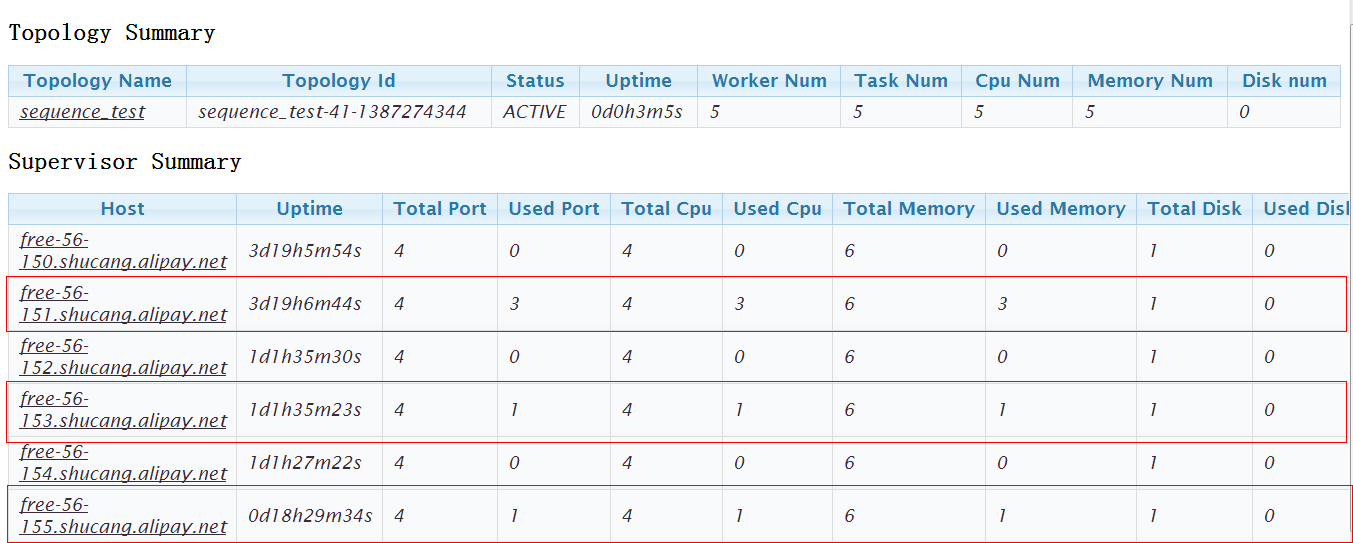
杀掉后重新提交:



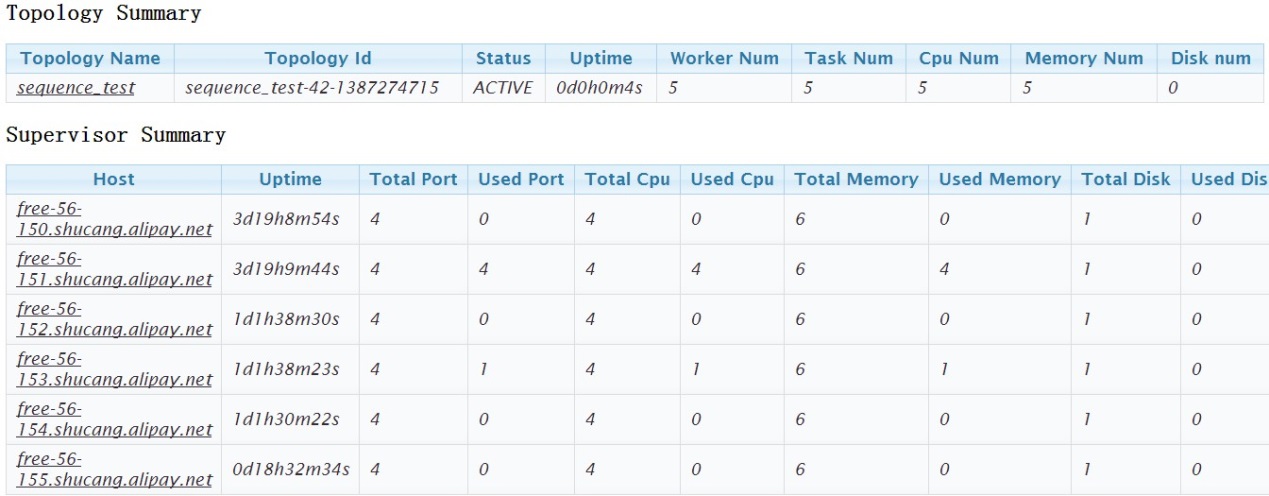
分配情况一致

1. **USER\_DEFINE\_ASSIGNMENT 不满足， USE\_OLD\_ASSIGNMENT 满足**

提交topology后人为杀掉一个worker,重新在其他机器启动新的worker人为造成USE\_OLD\_ASSIGNMENT与USER\_DEFINE\_ASSIGNMENT不一致的情况

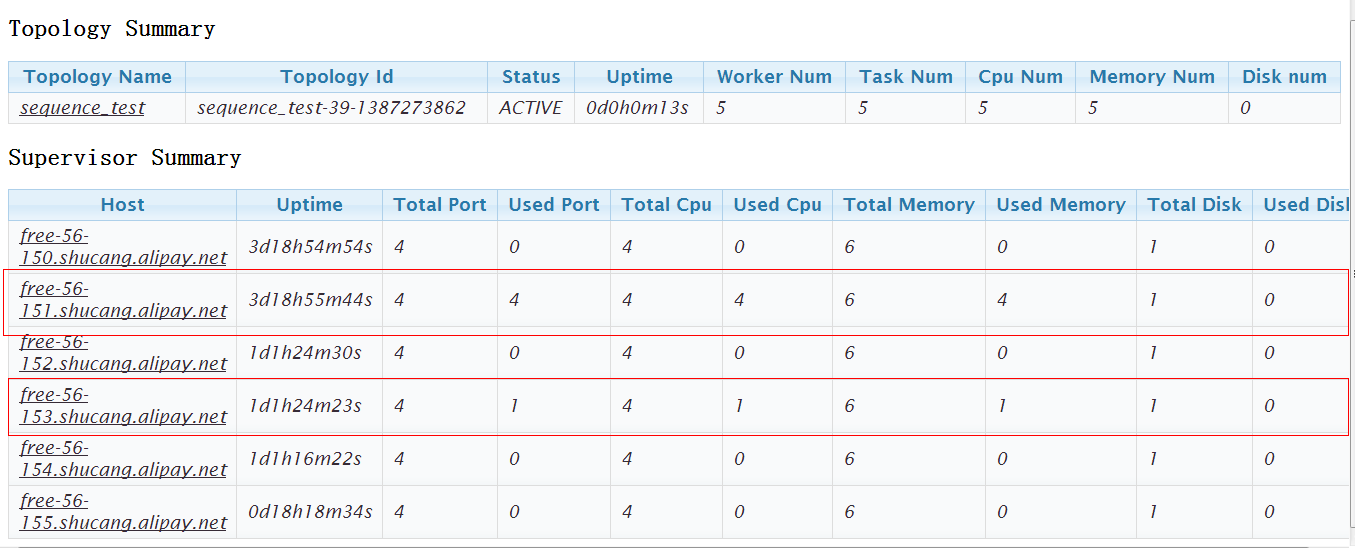
****

杀掉后重新启动 优先遵循USER\_DEFINE\_ASSIGNMENT：

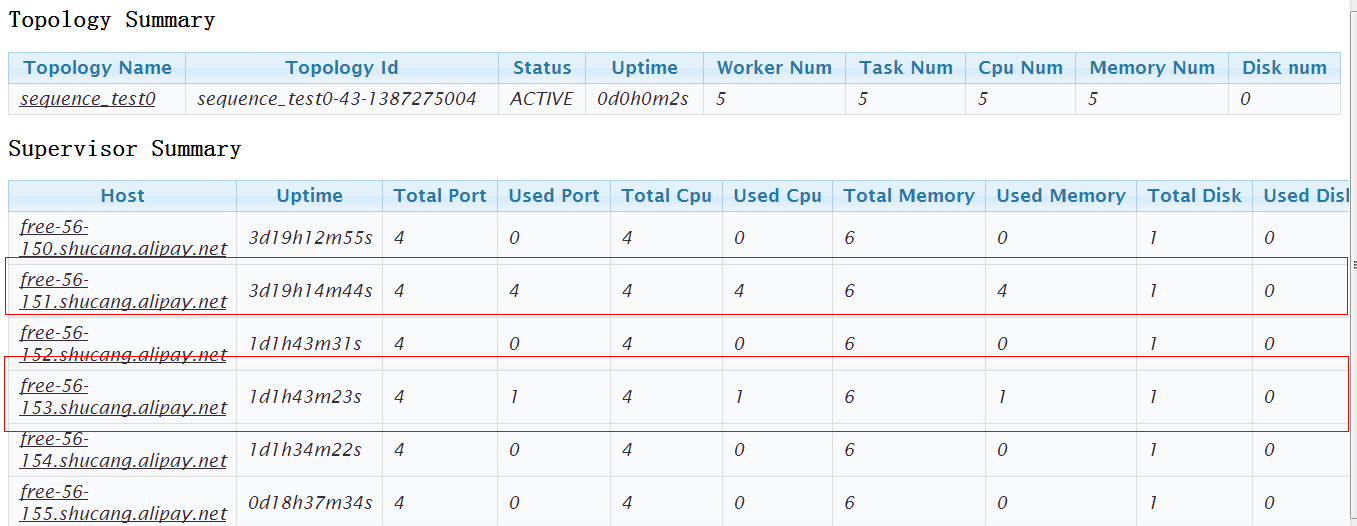


1. **USER\_DEFINE\_ASSIGNMENT 不满足， USE\_OLD\_ASSIGNMENT 不满足**

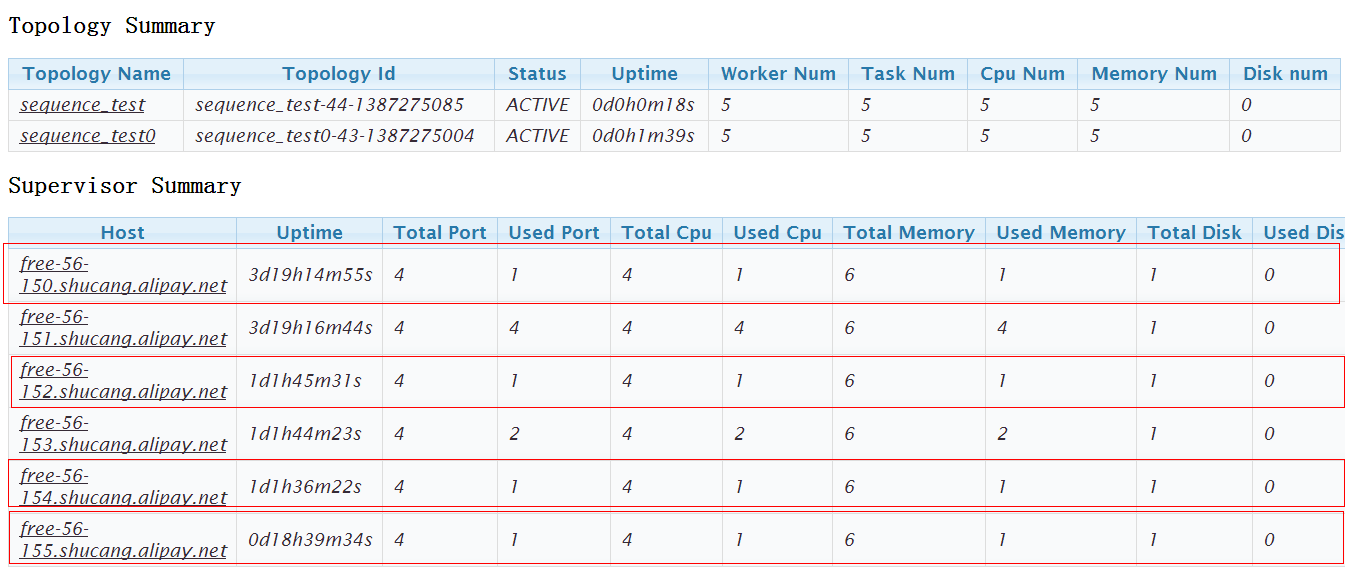
第一次提交:



然后杀掉后启动一个相同的topology名字为sequence\_test0 使两个条件均不满足:



此时重新提交sequence\_test,根据现有的资源情况根据默认算法再分配:

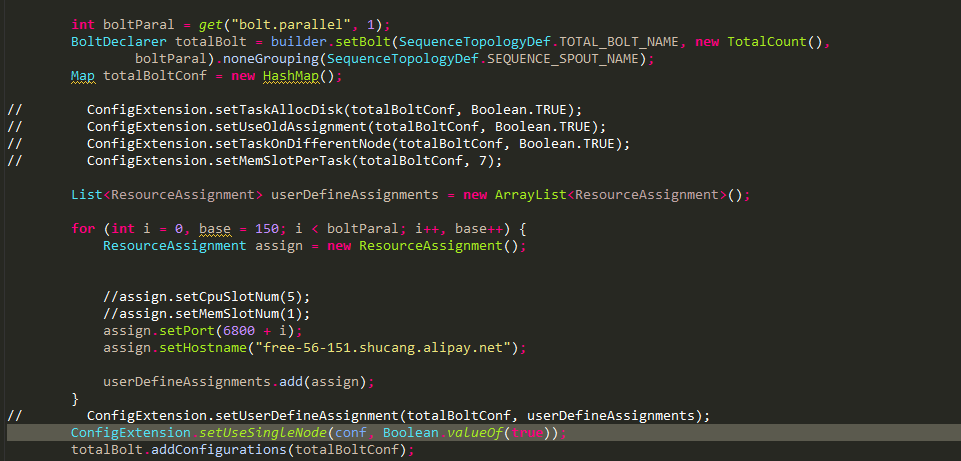


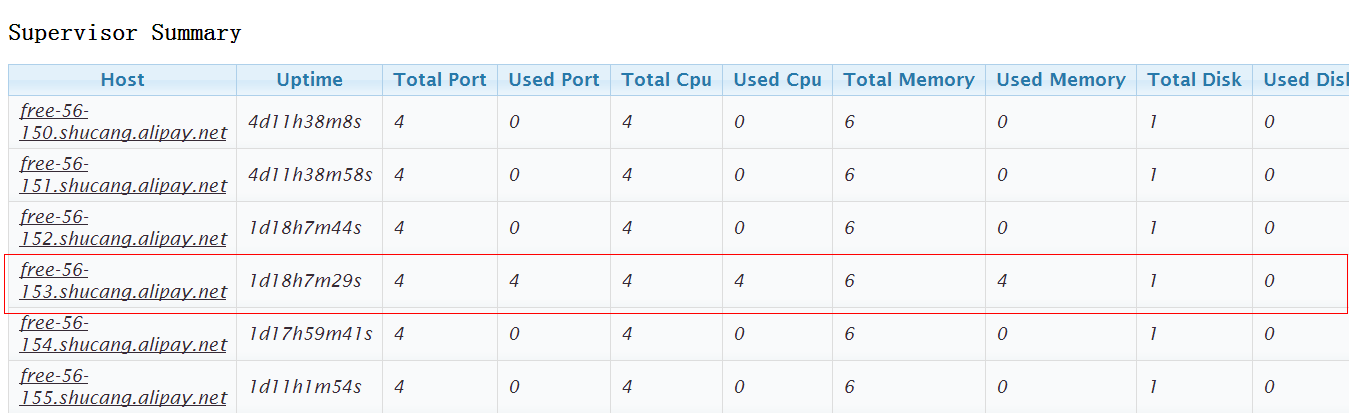
## 使用USE\_SINGLE\_NODE

测试目标：

4种case：

1. 存在单个supervisor 满足所有资源申请， 没有使用 USER\_DEFINE\_ASSIGNMENT

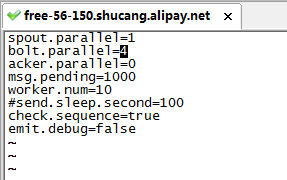




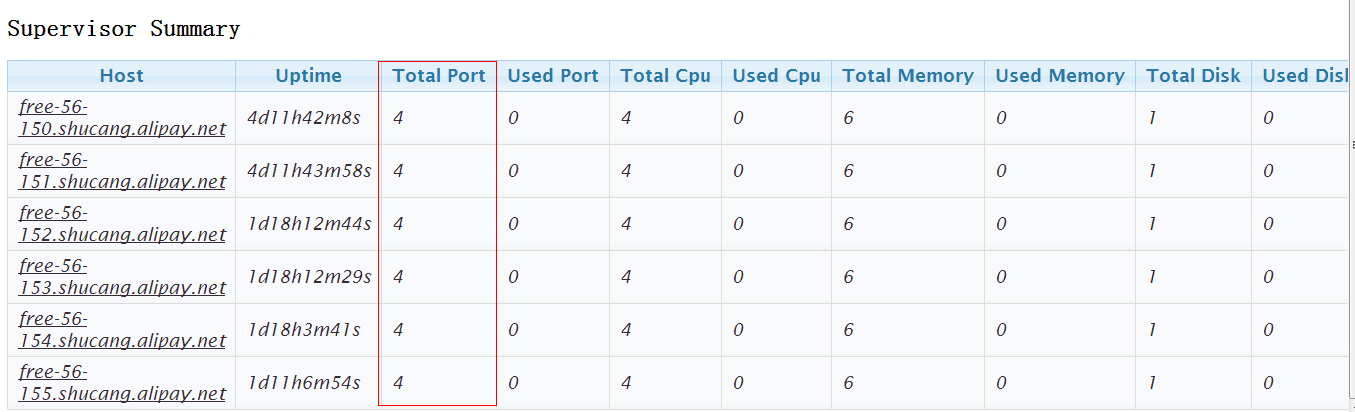
分配正确

1. 没有单个supervisor 满足所有资源申请， 没有使用 USER\_DEFINE\_ASSIGNMENT

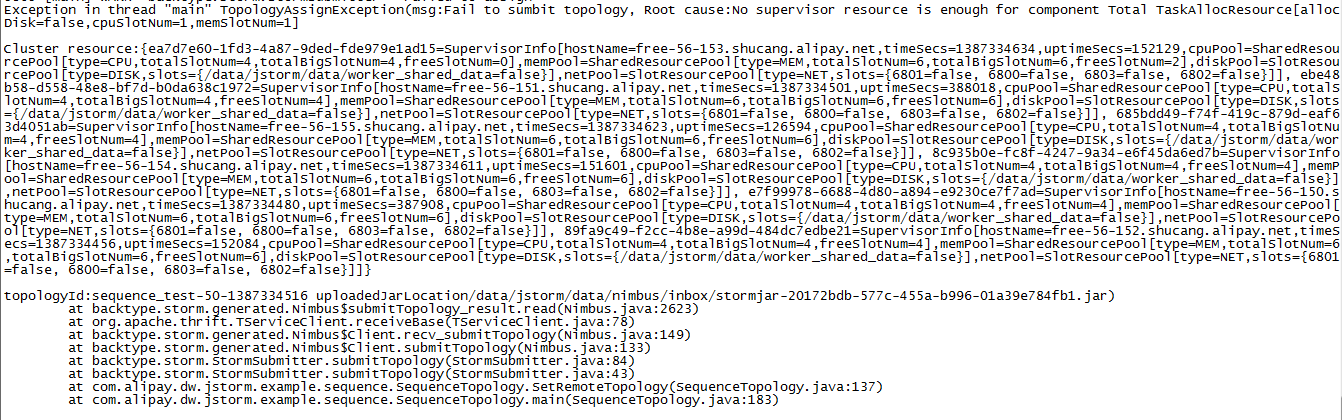
4个bolt+1个spout需要5个port slot:



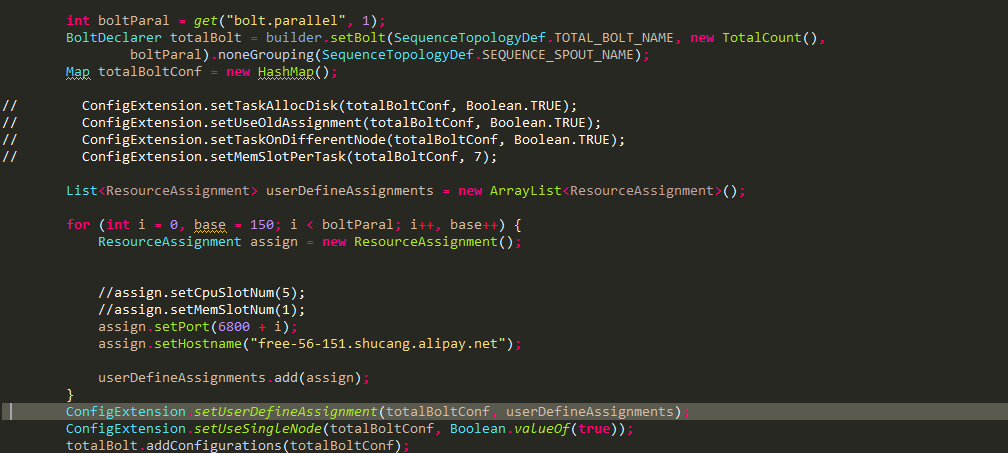
没有符合的supervisor:

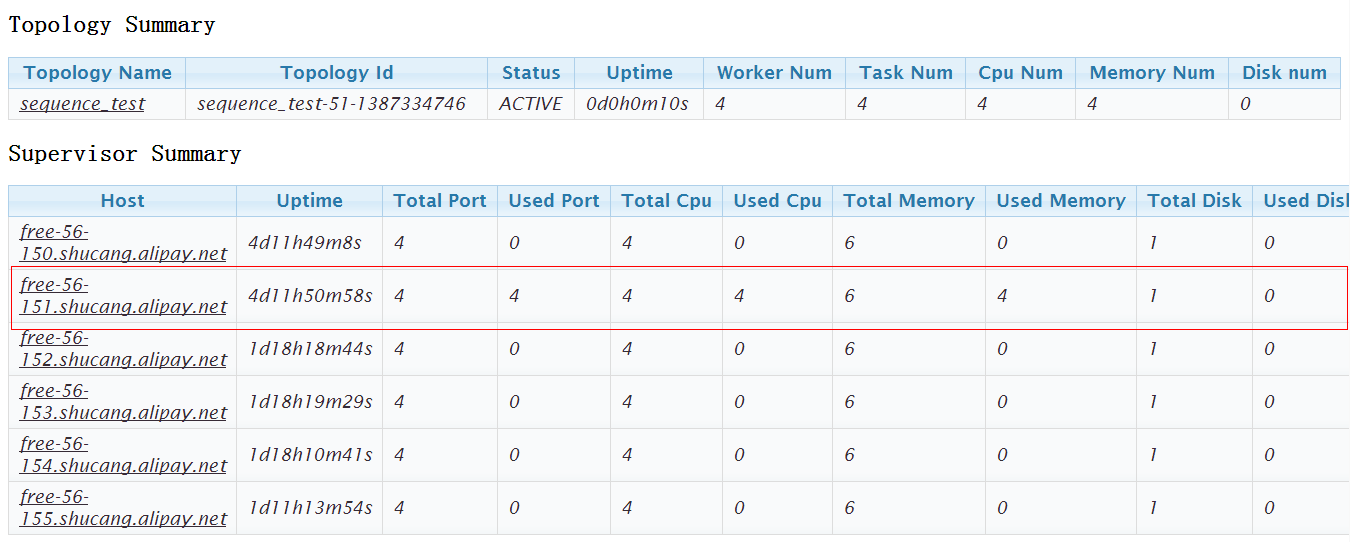


提交失败:



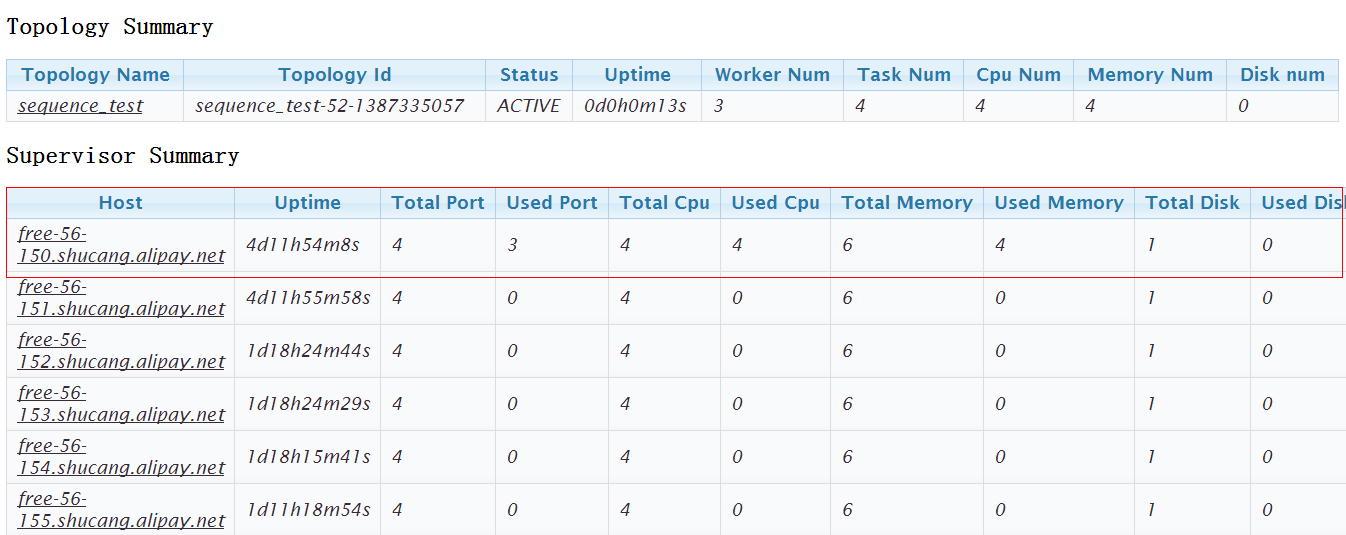
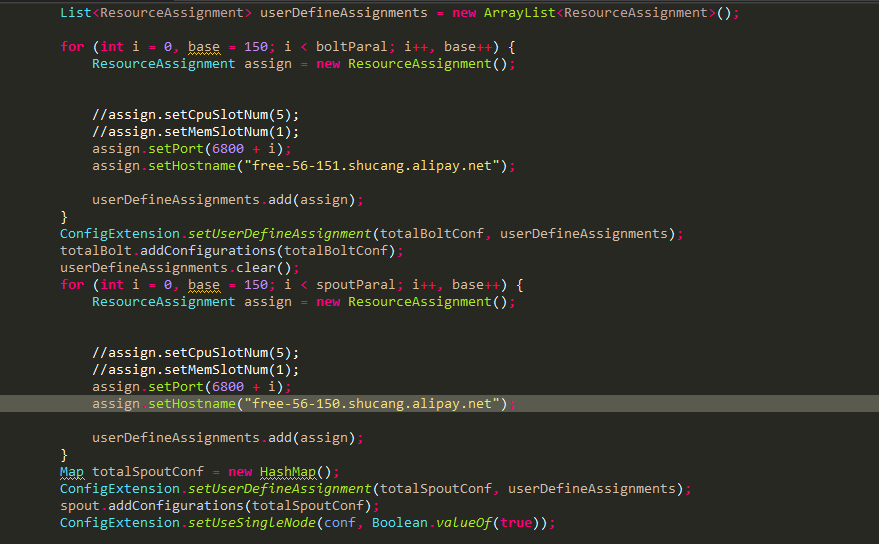
1. 存在单个supervisor 满足所有资源申请， 使用USER\_DEFINE\_ASSIGNMENT， 并且USER\_DEFINE\_ASSIGNMENT与USE\_SINGLE\_NODE 不冲突





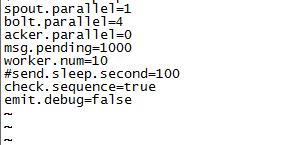
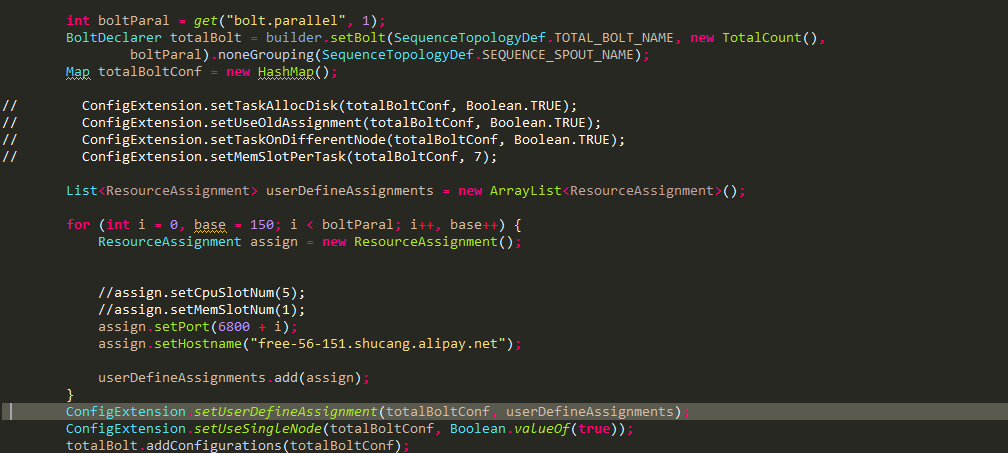
分配成功

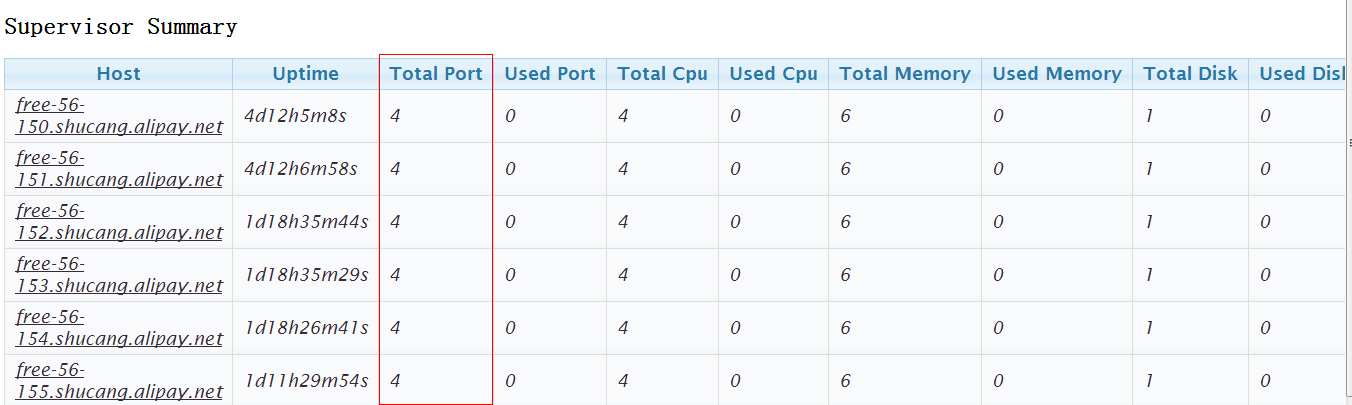
1. 存在单个supervisor 满足所有资源申请， 使用USER\_DEFINE\_ASSIGNMENT， 并且USER\_DEFINE\_ASSIGNMENT与USE\_SINGLE\_NODE 冲突

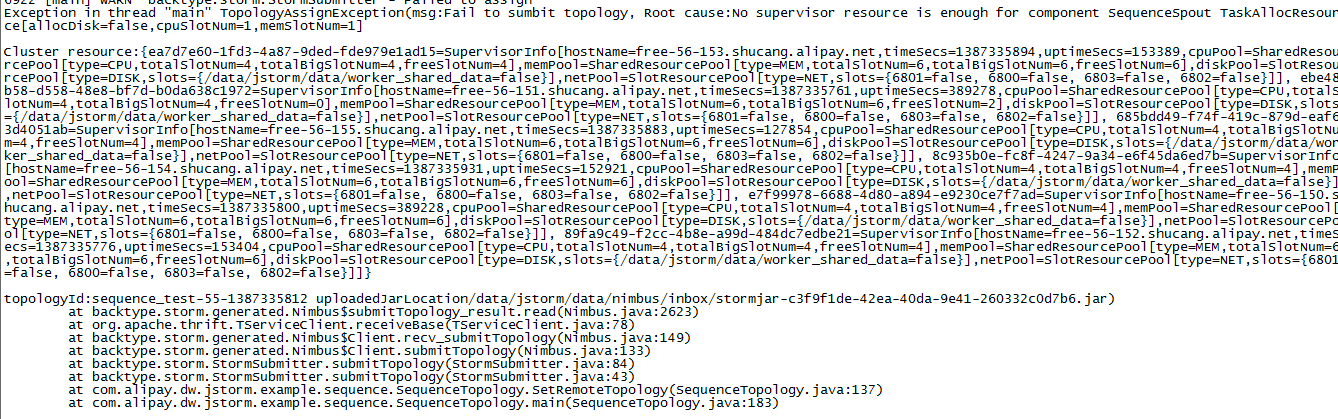


优先满足USE\_SINGLE\_NODE

1. 没有单个supervisor 满足所有资源申请， 使用USER\_DEFINE\_ASSIGNMENT， 并且USER\_DEFINE\_ASSIGNMENT与USE\_SINGLE\_NODE 不冲突

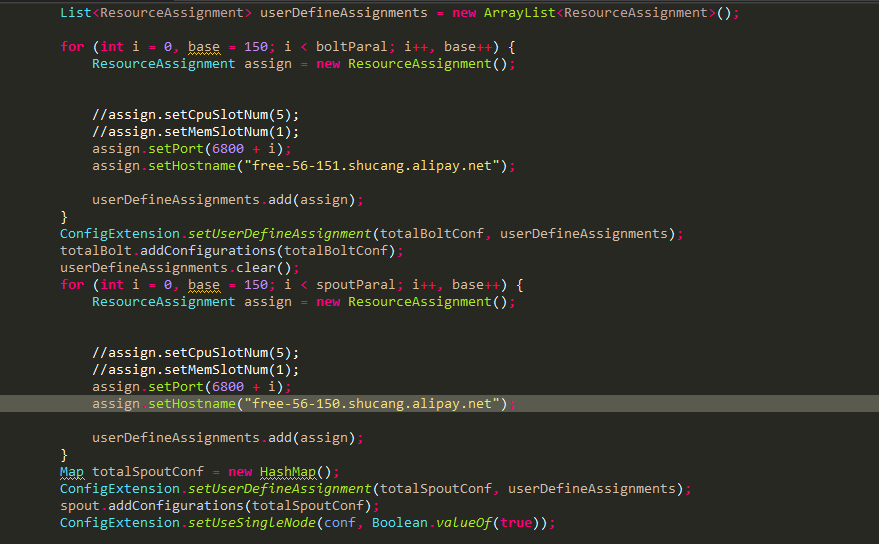


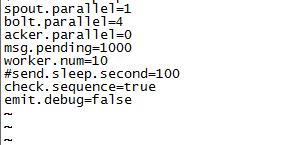


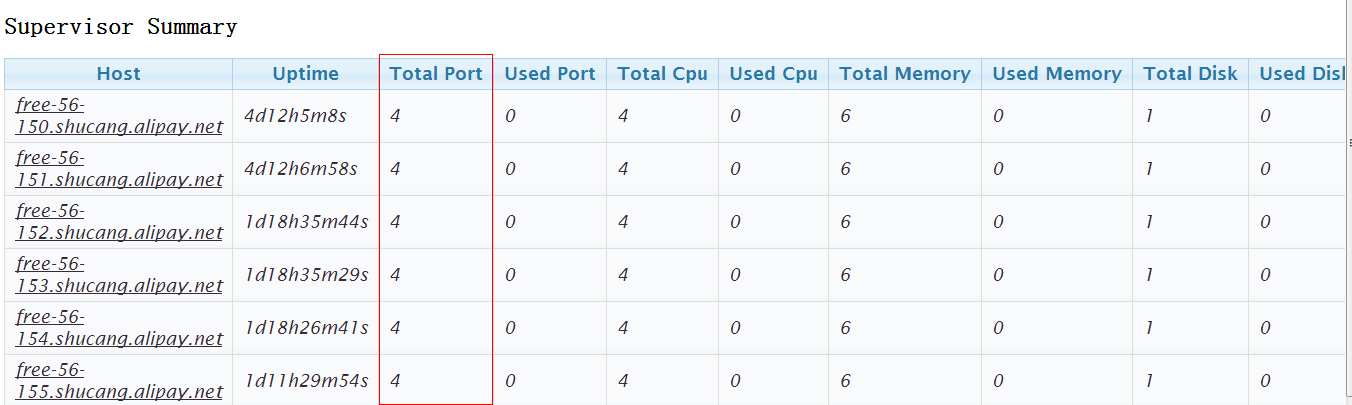


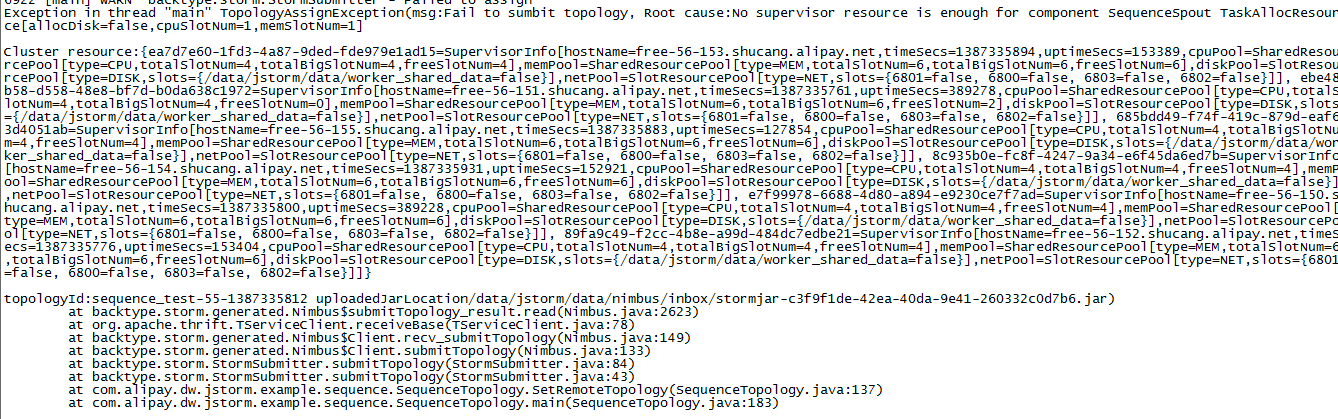
无法提交

1. 没有单个supervisor 满足所有资源申请， 使用USER\_DEFINE\_ASSIGNMENT， 并且USER\_DEFINE\_ASSIGNMENT与USE\_SINGLE\_NODE 冲突









无法提交

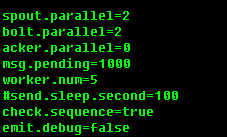
## worker 数比supervisor 数要多， 资源足够的时候

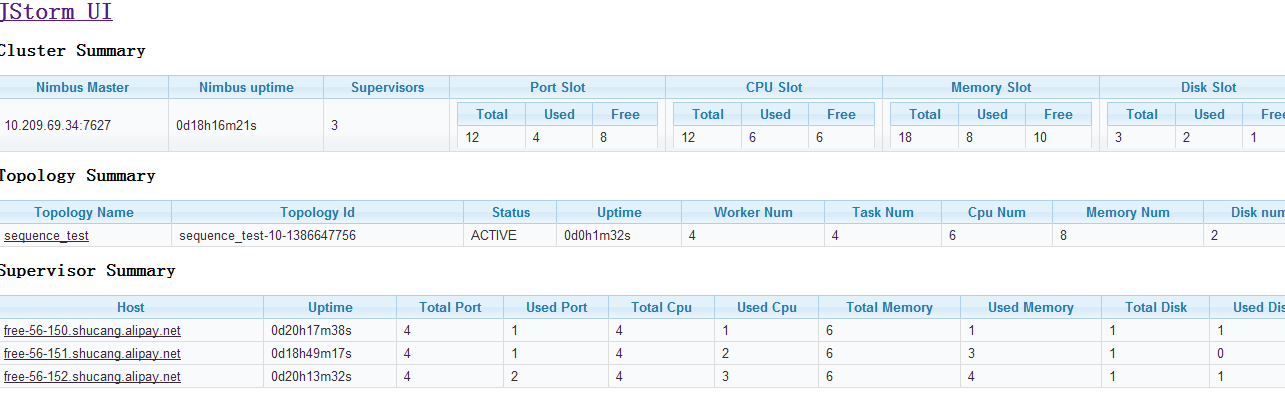
### 测试目标：

检测分配是否均匀

### 测试结果

通过





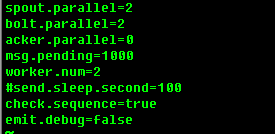
## worker 数比supervisor 要少， 资源足够的时候

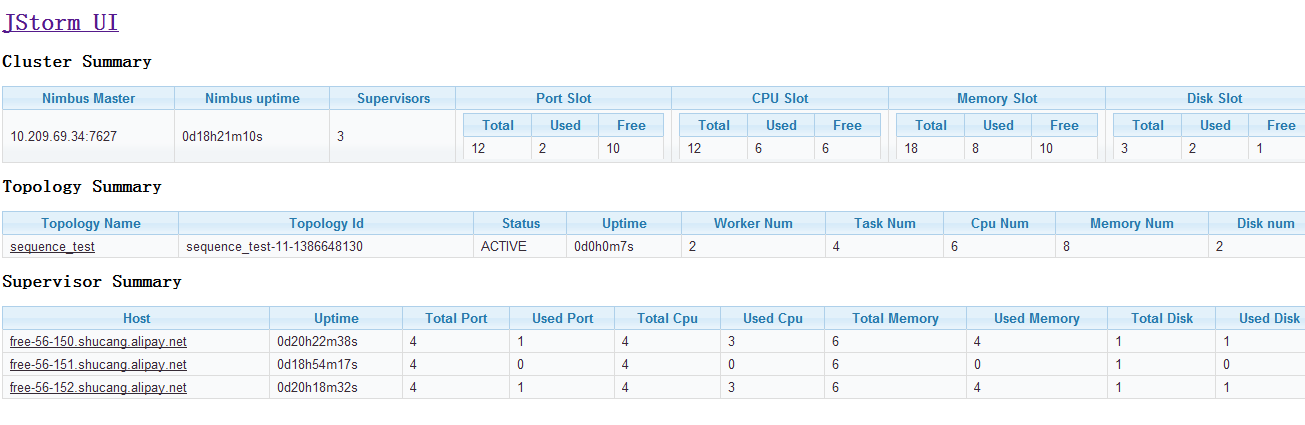
### 测试目标：

测试分配是否均匀

### 测试结果：

通过





## 整体上CPU 资源不够的时候

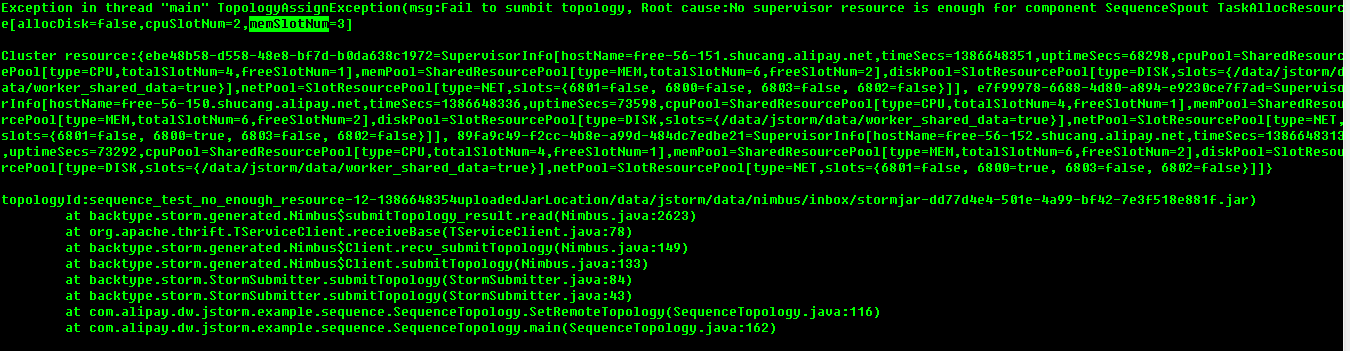
### 测试目标：

测试是否可以分配

### 测试结果：

通过

无法分配



## 整体上Memory 不够的时候

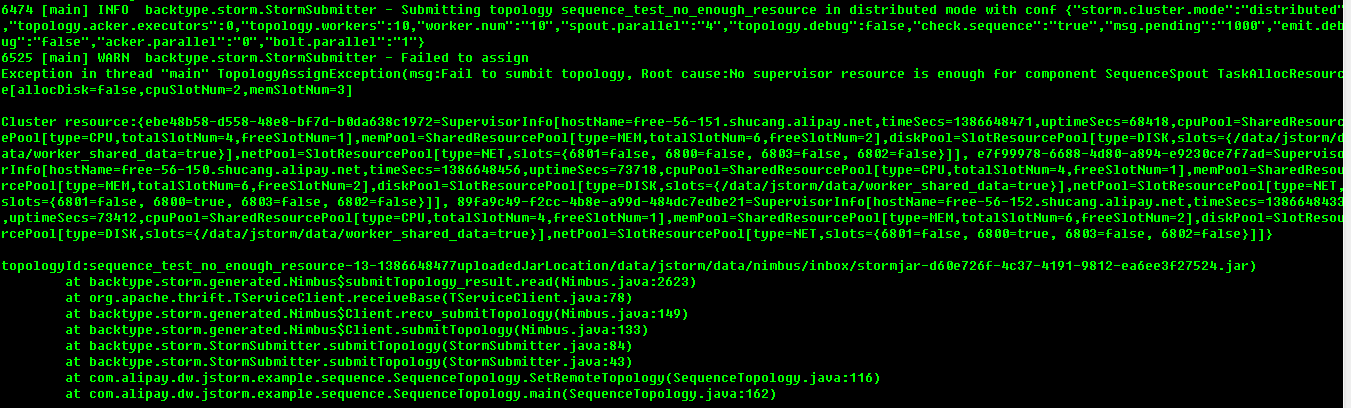
### 测试目标：

测试是否可以分配

### 测试结果：

通过

无法分配



## 整体上Disk资源不够的时候

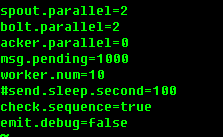
### 测试目标：

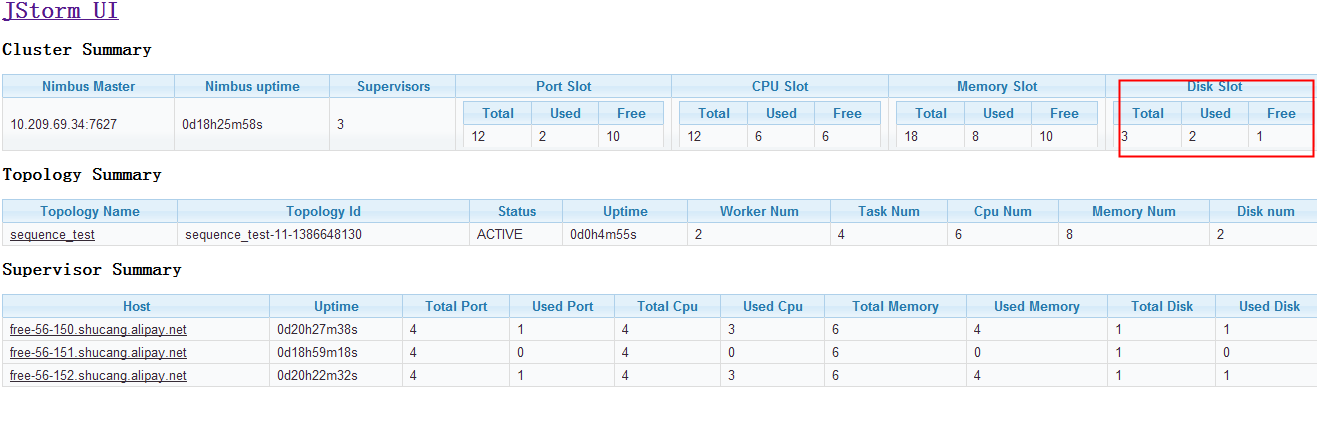
测试是否可以分配

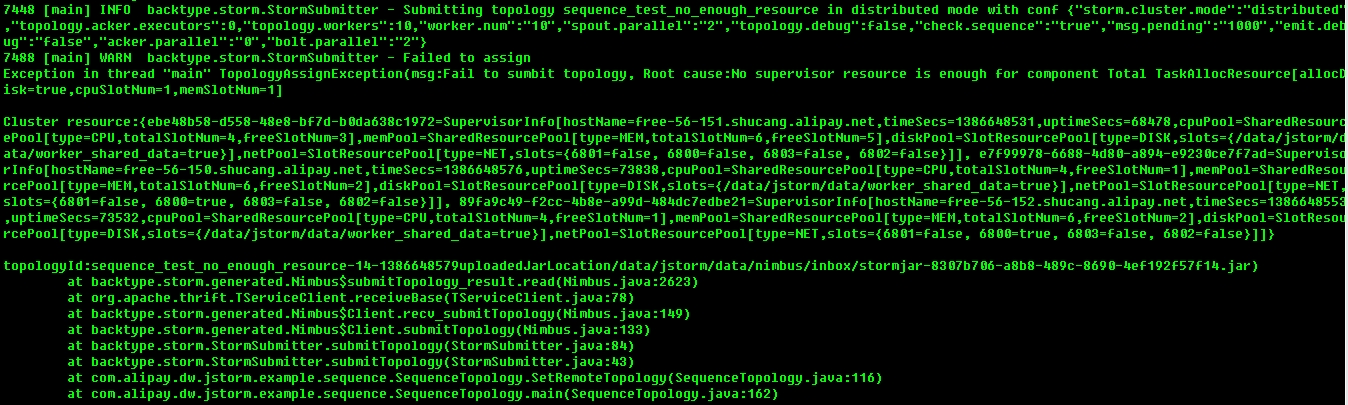
### 测试结果：

通过

无法分配





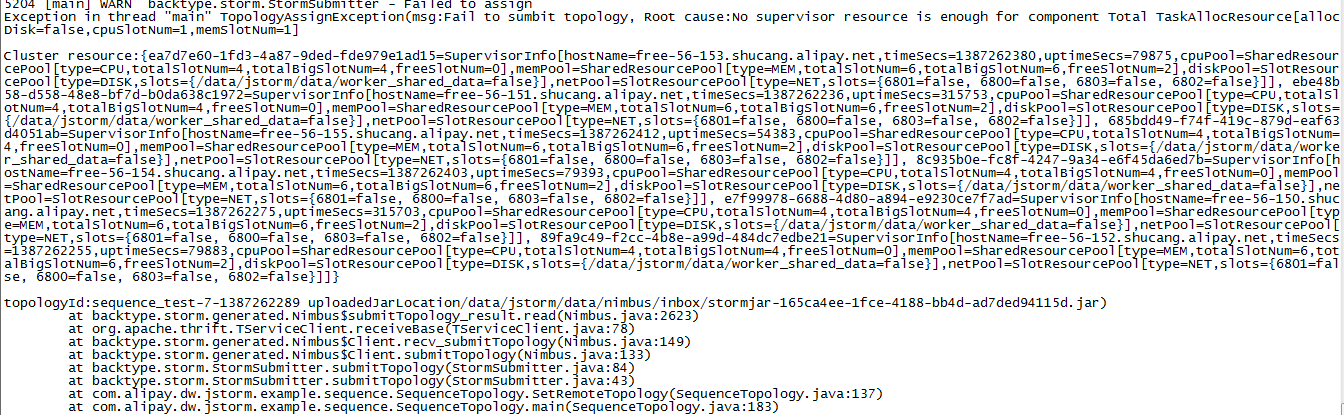


## 整体上port资源不够的时候

### 测试目标：

整体上Port资源不够时无法提交任务

### 测试结果：



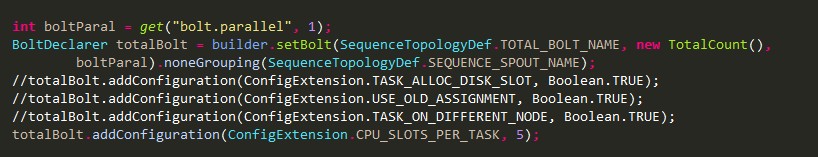
无法提交

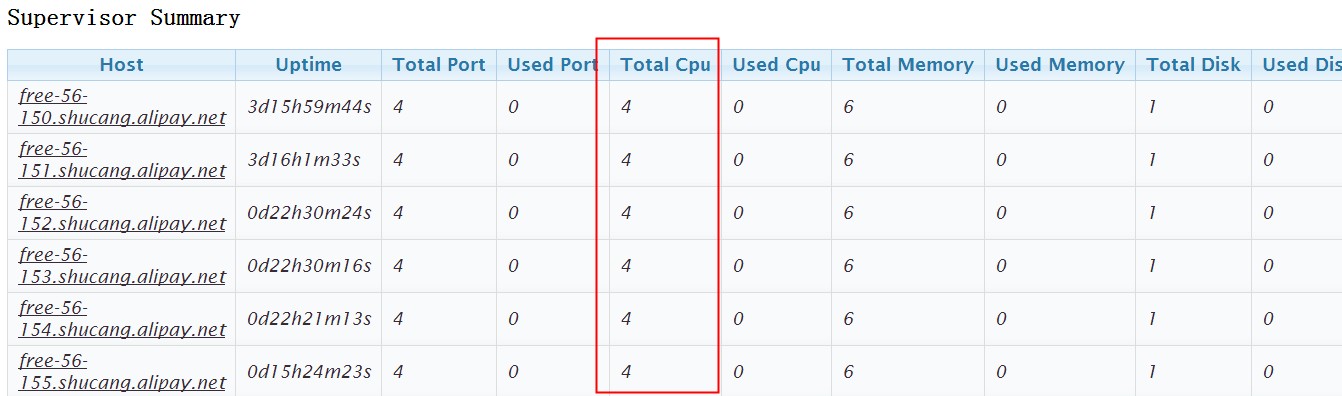
## 整体上所有资源可以满足，但是单个supervisor满足不了CPU

### 测试目标：

这种情况下应该出现无法提交的情况

### 测试结果：







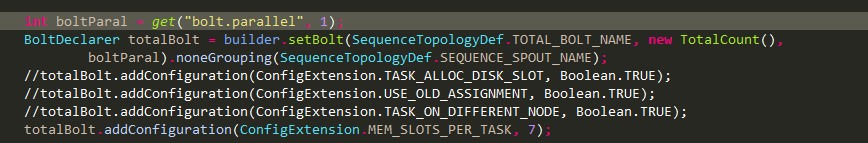
无法提交

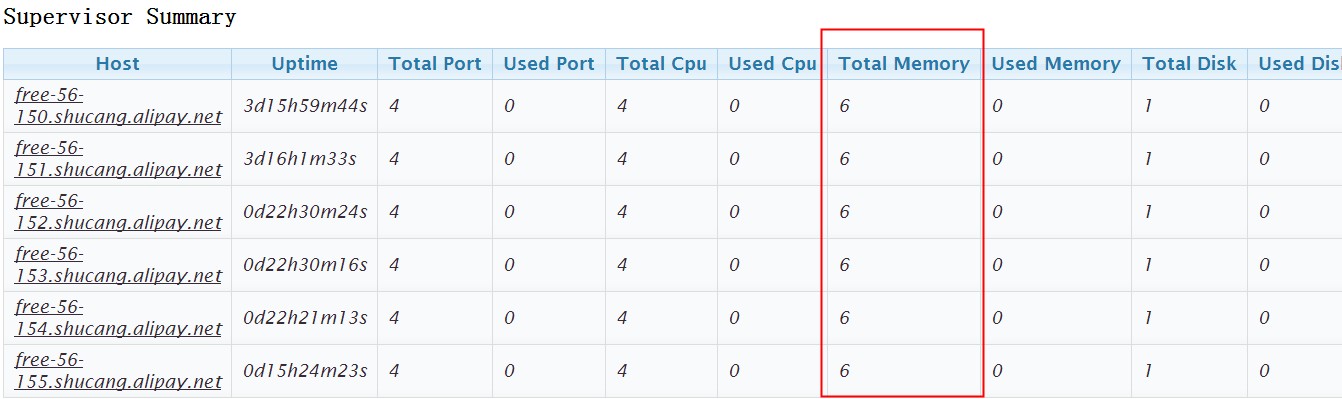
## 整体上所有资源可以满足，但是单个supervisor满足不了Memory

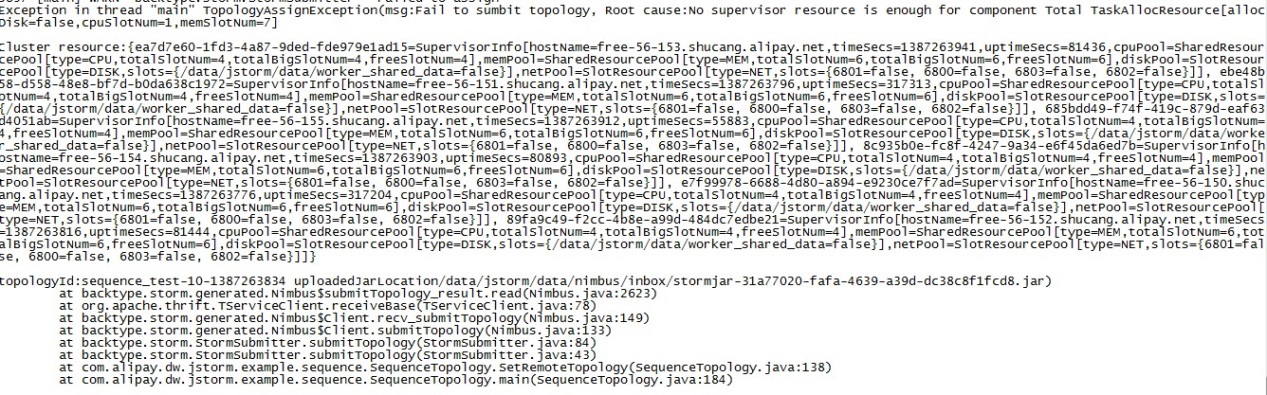
### 测试目标：

这种情况下应该出现无法提交的情况

### 测试结果：







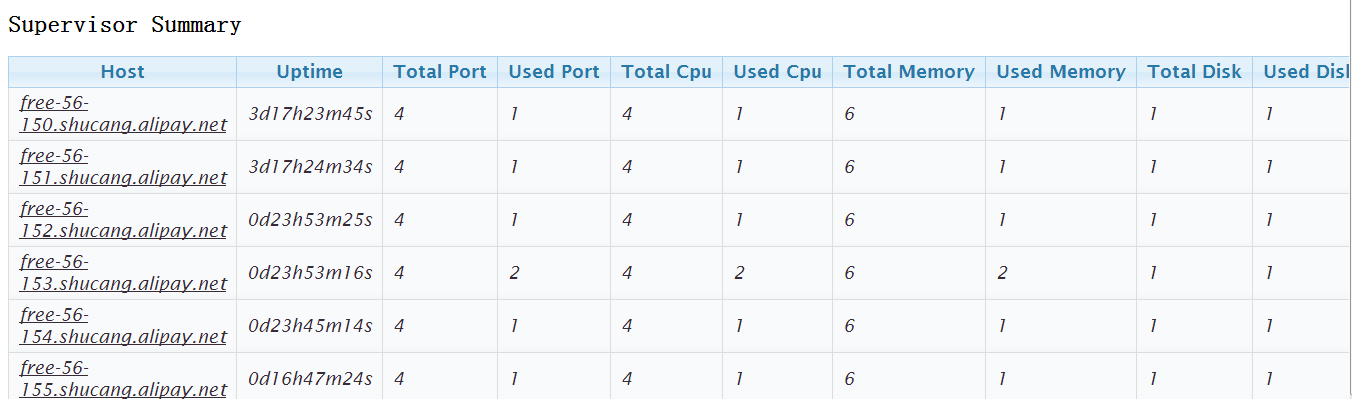
## 整体上所有资源可以满足，但是单个supervisor满足不了disk

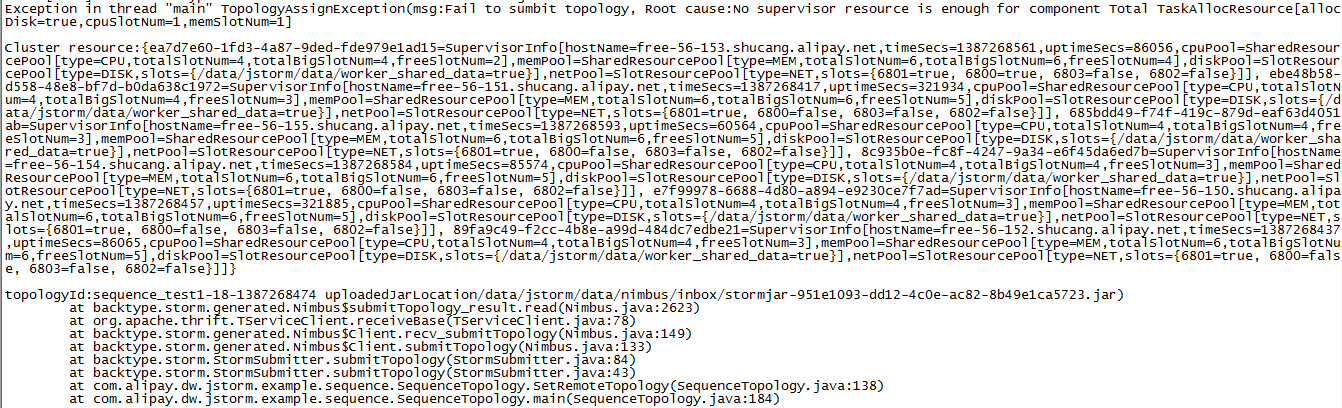
### 测试目标：

这种情况下应该出现无法提交的情况

### 测试结果：

先提交一个topology消耗完所有disk,再提交第二个,其他资源都满足但无法提交





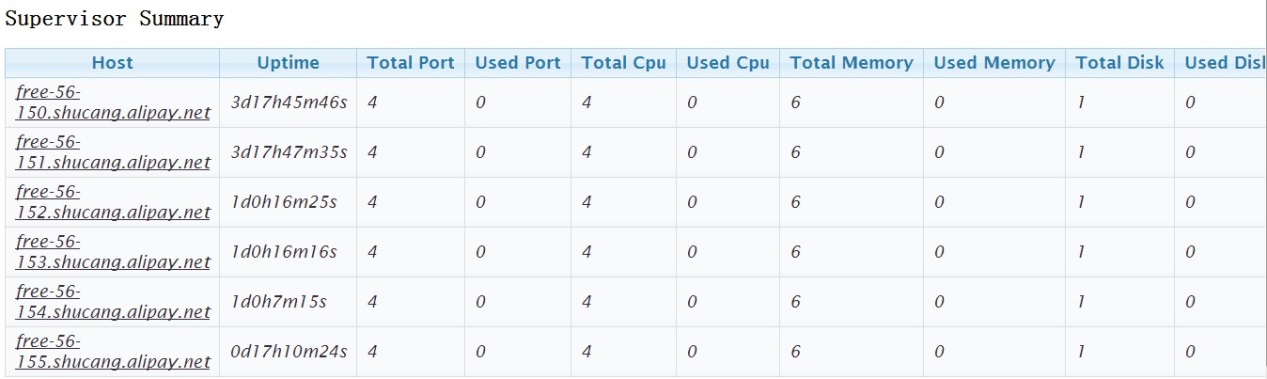
## 整体上所有资源可以满足，但isolation 无法满足

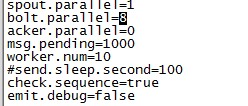
### 测试目标：

这种情况下应该出现无法提交的情况

### 测试结果：

总共6个supervisor

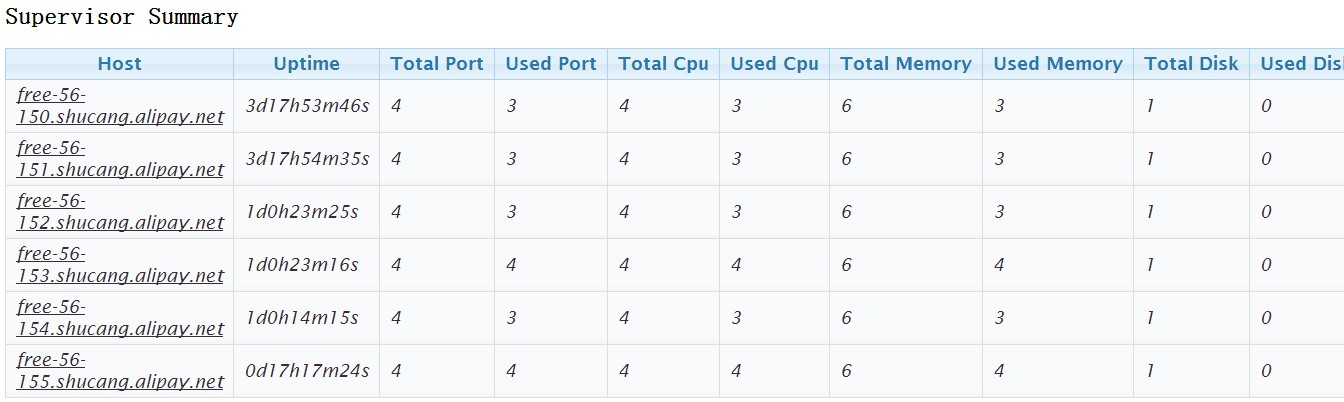




8个bolt 无法满足isolation 提交失败

## 运行10个topology，检查平衡性如何

### 测试结果：



10个topology的情况下未出现不平衡的情况