## 调度细则

1. 任务调度算法以worker为维度；
2. 调度过程中正在进行的调度动作不会对已发生的调度动作产生影响；
3. 调度过程中用户可以自定义 useDefined Assignment，和使用已有的old Assignment，这两者的优先级是：useDefined Assignment>old Assignment;
4. 用户可以设置task.on.differ.node参数，强制要求同组件的task分布到不同supervisor上；

**默认调度算法：**

**1、**以worker为维度，尽量将worker平均分配到各个supervisor上；

2、以worker为单位，确认worker与task数目大致的对应关系（注意在这之前已经其他拓扑占用利用的worker不再参与本次动作）；

3、建立task-worker关系的优先级依次为：尽量避免同类task在同一work和supervisor下的情况，尽量保证task在worker和supervisor基准上平均分配，尽量保证有直接信息流传输的task在同一worker下；

## 2 Worker资源足够，测试调度分配是否合理

### 2.1 常规测试一

* **配置：**task.on.differ.node=false，topology\_nums=1, topology\_level=1, workers=6,

spout.parallel=6, bolt\_0.parallel=6, acker.executors=6;

每台机子canused\_slot=16

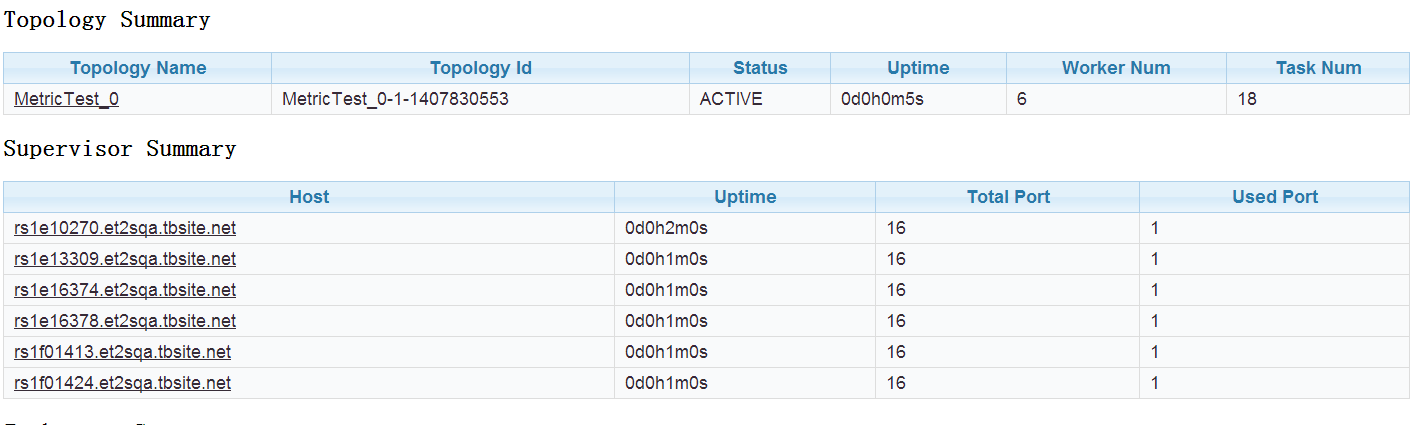
* **测试目标（当每台机子端口一致时，work平均分配，同类task尽量分开，且task尽量均匀分布于work和supervisor）**

S\_x 代表spout-task， B\_x\_x 代表第x个bolt-task，A\_x 代表acker-task

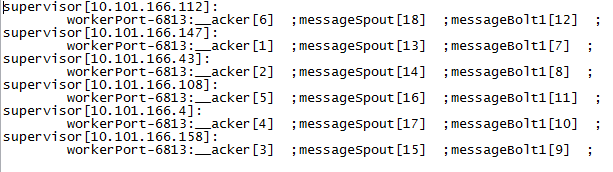
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Supervisor** | | **1** | **2** | **3** | **4** | **5** | **6** |
| Work1 | spout-task | 1 | 1 | 1 | 1 | 1 | 1 |
| Bolt0-task | 1 | 1 | 1 | 1 | 1 | 1 |
| acker-task | 1 | 1 | 1 | 1 | 1 | 1 |

* **测试结果:**

Worker被一一分配到supervisor上



Task按规定分配到work上去:



**合理**

### 2.2 常规测试二

* **配置：**task.on.differ.node=false，topology\_nums=1, topology\_level=1, workers=24,

spout.parallel=10, bolt\_0.parallel=18, acker.executors=12;

机1端口数量=4、机1端口数量=4、机1端口数量=4、机1端口数量=4、

机1端口数量=4、机1端口数量=4

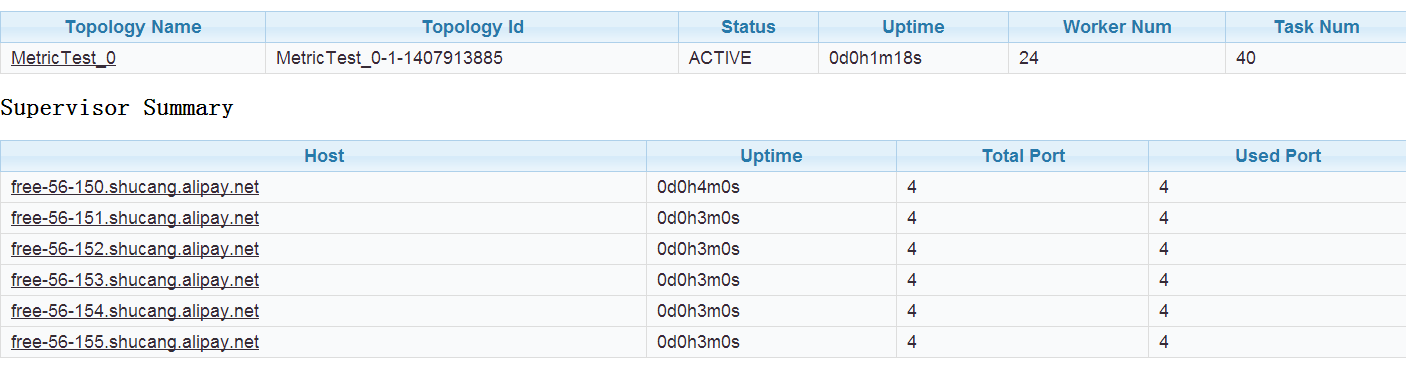
* **测试目标**

**①**以worker为维度，尽量将worker平均分配到各个supervisor上；

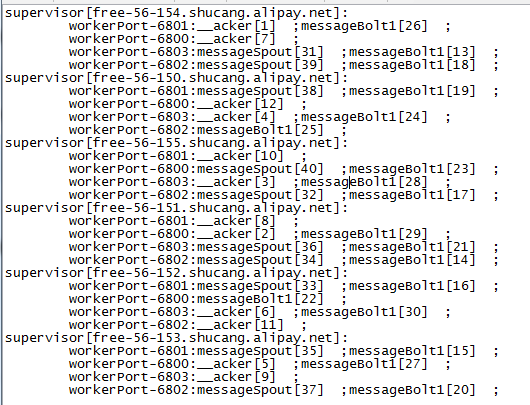
②保证同类型task尽量均分到不同worker和supervisor的基础上，其次保证每一个worker和supervisor的task数量均衡，最后保证直接相关的task放到一个进程上去；

* **测试结果:**

Worker被一一分配到supervisor上



Task按规定分配到work上去（共有24端口，先分配12个acker保证能够放到不同的进程且每个supervisor有2个acker；再分配18个bolt-task，其中6个task会与acker在一个进程；最后分配10个spout-task分配，应该和bolt-task在一起）



**合理**

### 2.3 常规测试三

* **配置：**task.on.differ.node=false，topology\_nums=1, topology\_level=2, workers=12,

spout.parallel=8,bolt\_0.parallel=9,bolt\_1.parallel=9,

acker.executors=8;

机1端口数量=4、机1端口数量=4、机1端口数量=4、机1端口数量=4、

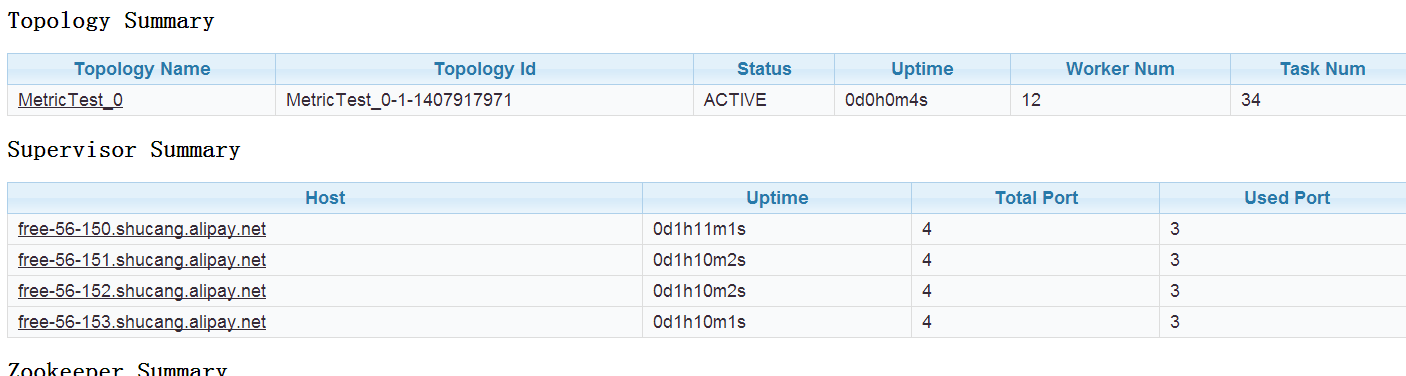
* **测试目标**

**①**以worker为维度，尽量将worker平均分配到各个supervisor上；

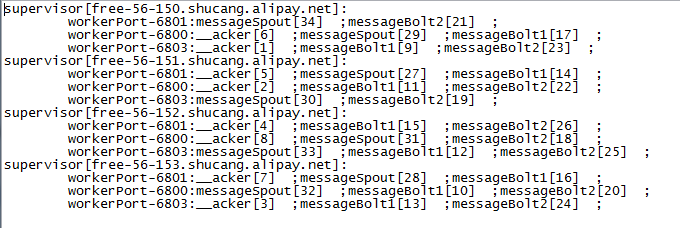
②保证同类型task尽量均分到不同worker和supervisor的基础上，其次保证每一个worker和supervisor的task数量均衡；最后保证直接相关的task放到一个进程上去，

* **测试结果:**

Worker被平均分配到supervisor上



Task按规定分配到work上去



**合理**

## 3 worker 设置测试

### 3.1 worker数量设置大于端口数量

* **配置：**task.on.differ.node=false，topology\_nums=1, topology\_level=1, **workers=20**,

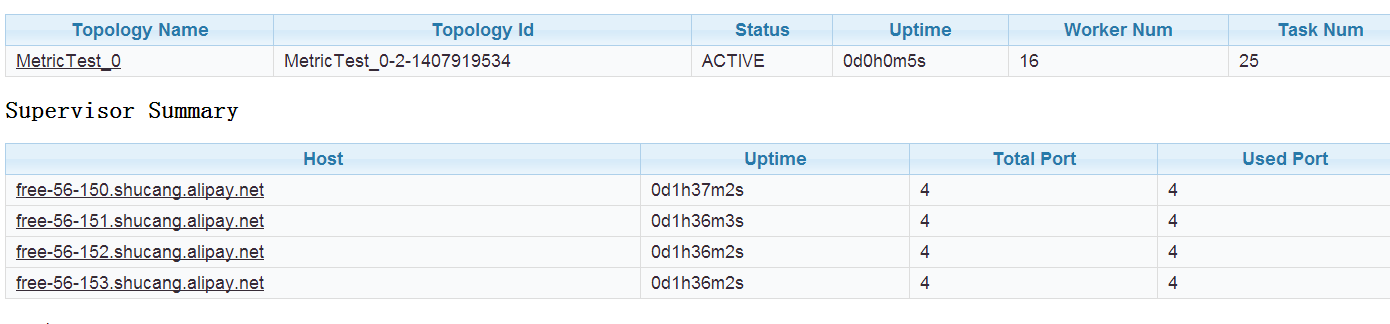
spout.parallel=8,bolt\_1.parallel=9,acker.executors=8;

机1端口数量=4、机1端口数量=4、机1端口数量=4、机1端口数量=4、

* **测试目标**

当worker数量不足时，调度可以分配；分配数采用的是最大可利用的端口数量

* **测试结果:**



**分配成功**

### 3.2 worker进程内存设置效果（略）

Worker占用内存大小可以自定义

## 4 storm集群变化对已有调度行为的影响

### 4.1 增加superivsor（略）

不影响已有拓扑任务调度

### 4.2 杀死worker（略）

重启worker，且调度正常

### 4.3 杀死supervisor（略）

拓扑继续跑，一切正常

### 4.4 其中一台机子宕机（略）

之前该机子跑的task将重新参与调度，调度结果符合要求

### 4.5 增加topology

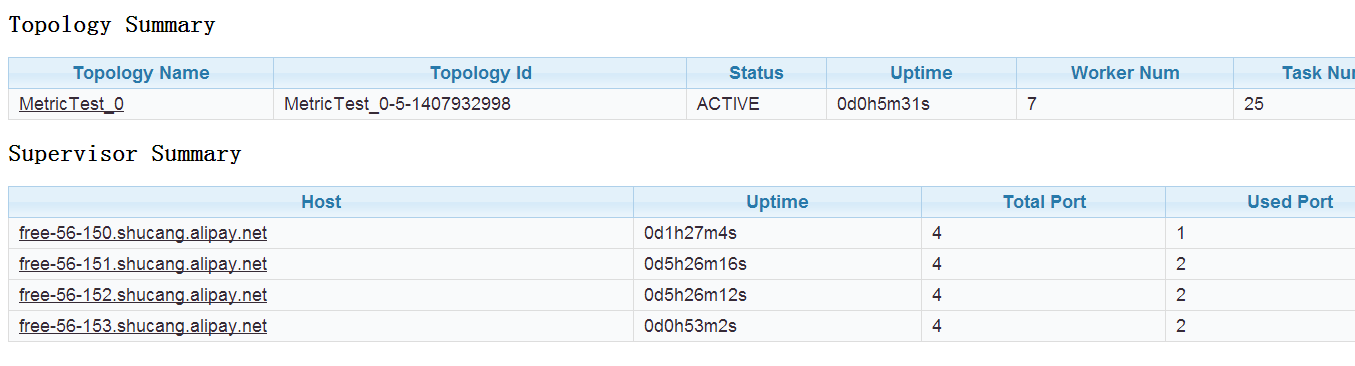
* **测试目标**

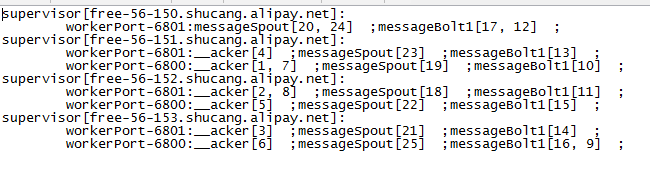
1：测试已有topology对新启动的topology任务调度无影响；

2：已有topology启用过的worker不再被其他topology使用；

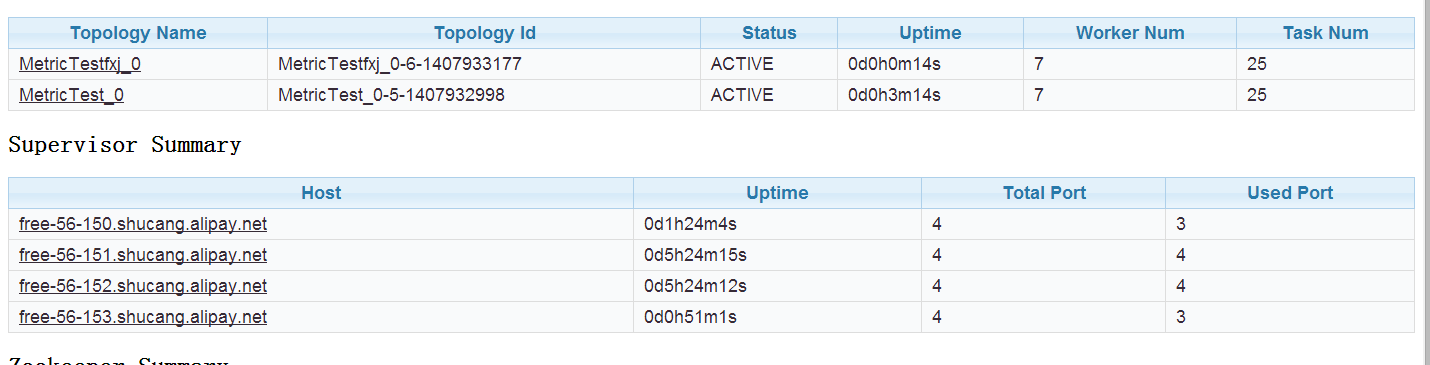
* **测试结果:**

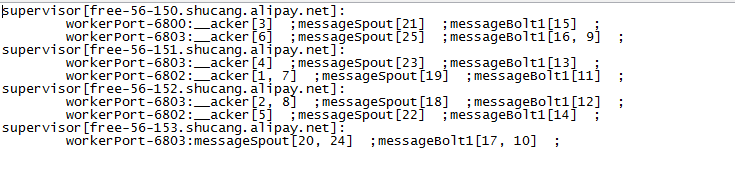
先启动topology1：





再启动topology2





**合理**

### 4.6 增加supervisor，再rebalance

* **配置：**task.on.differ.node=false，topology\_nums=1, topology\_level=1, **workers=7**,

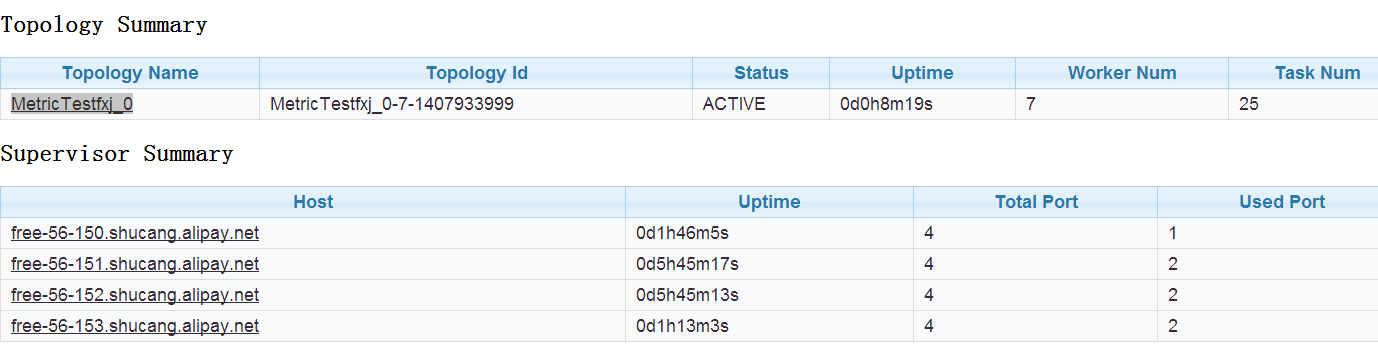
spout.parallel=8,bolt\_1.parallel=9,acker.executors=8;

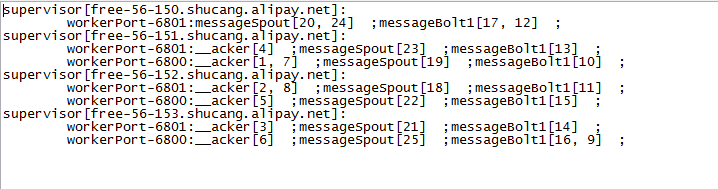
* **测试目标**

增加supervisor，再rebalance操作后，新增加的supervisor会加入任务调度过程，调度分配合理

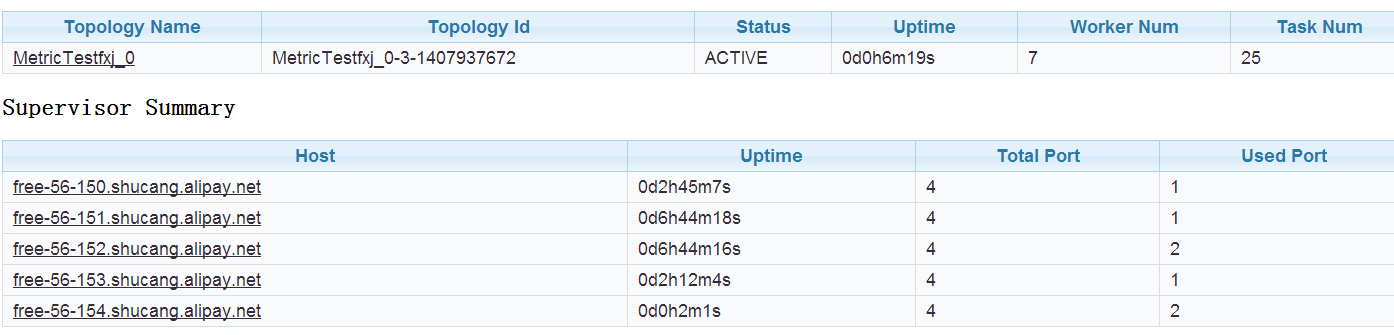
* **测试结果:**

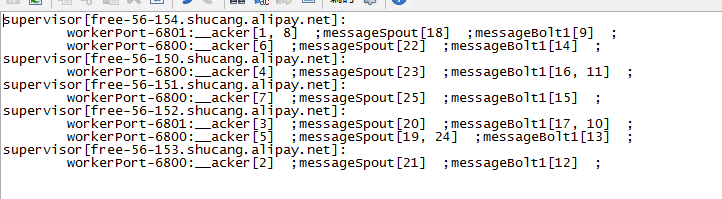
最初：





增加supervisor，再rebalance操作：



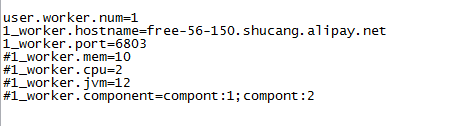


合理（新增加的supervisor会加入任务调度过程）

# 4 用户自定义任务调度测试

自定义调度有用户自己去制定调度，以下贴出了读取配置文件设置自定义调度的过程：

**文件**

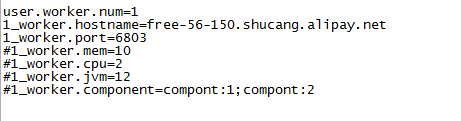


**对应代码:**



## 4.1 新建worker,但是不加入task

* **配置：**



* **测试目标**

用户自定义worker，但没有设置task，则调度过程将按照默认调度

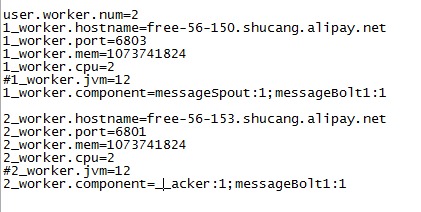
* **测试结果:**



**与预期相符**

## 4.2 自定义完整的调度

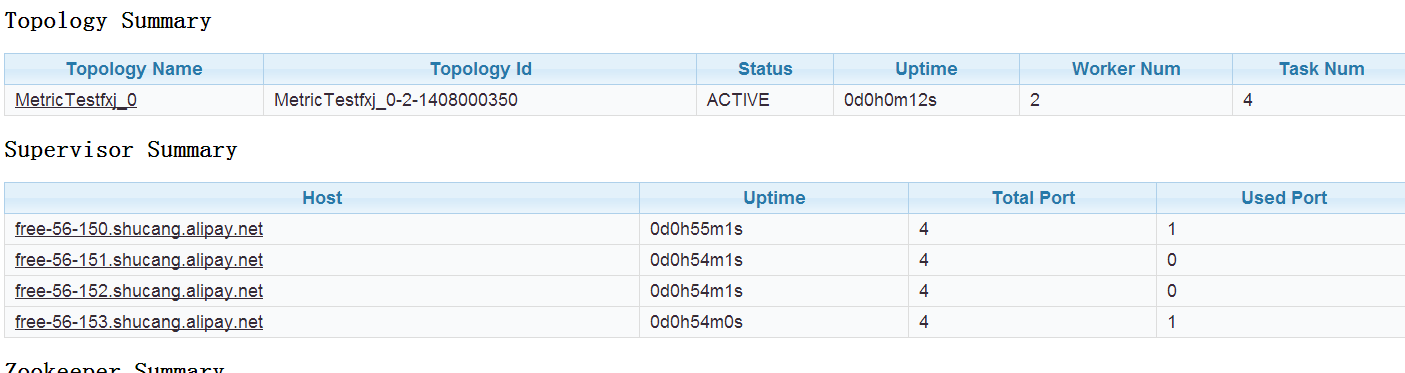
* **自定义任务配置：**

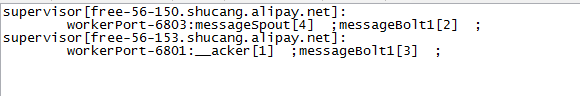


* **测试目标**

自定义任务调度与拓扑跑起来时的任务调度一致

* **测试结果:**





**与预期相符**

## 4.3 用户部分定义的调度

* **自定义任务配置：**

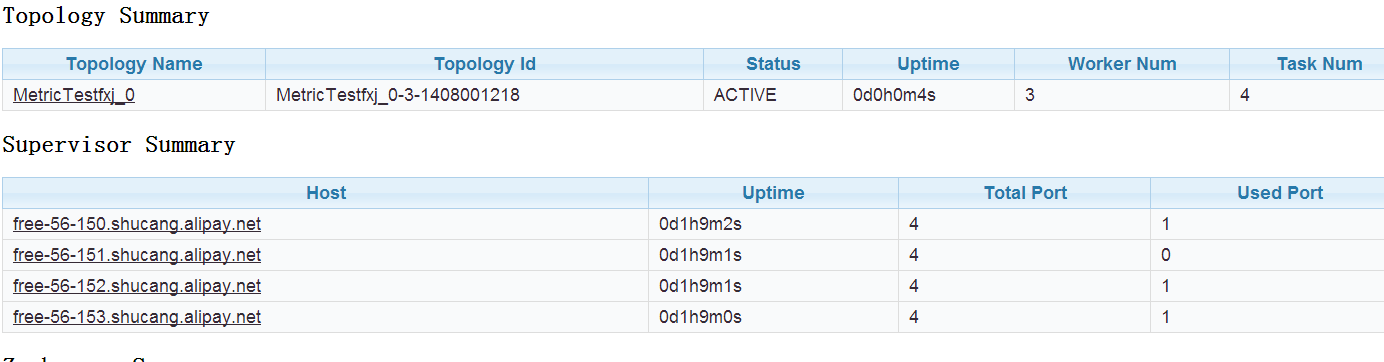
设置了3个worker和4个task，但目前用户自定义的2个worker和2个task，这种情况叫用户部分自定义。

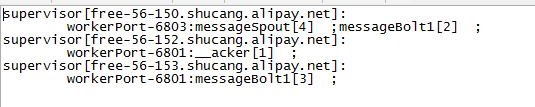


* **测试目标**

调度过程中，先按照用户定义的任务调度，余下部分按照默认调度；

* **测试结果:**





**与预期相符**

## 4.4 用户自定义worker数量大于允许worker数量（略）



将按照默认算法调度任务，自定义任务调度失效

## 4.5 用户自定义任务出错（略）

将按照默认算法调度任务，自定义任务调度失效

## 4.6 在已有的拓扑下，再增加个拓扑

**第一个拓扑是用户自定义，第二拓扑采用默认调度：**

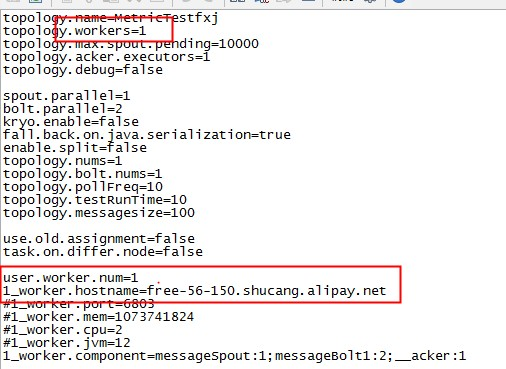
自定义任务调度正在使用的worker，不再参与接下来的任务调度行为

**第一个拓扑是默认调度，第二拓扑采用用户自定义：**

默认调度正在使用的worker，不再参与接下来的任务调度行为

## 4.7 未定义端口号，只定义了hostname

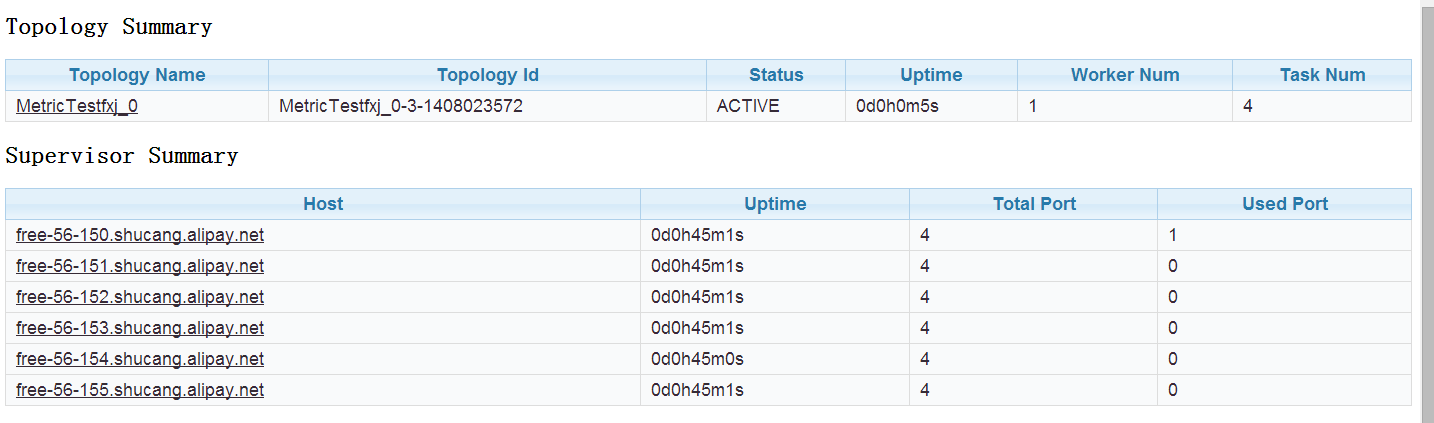
未设定端口号：

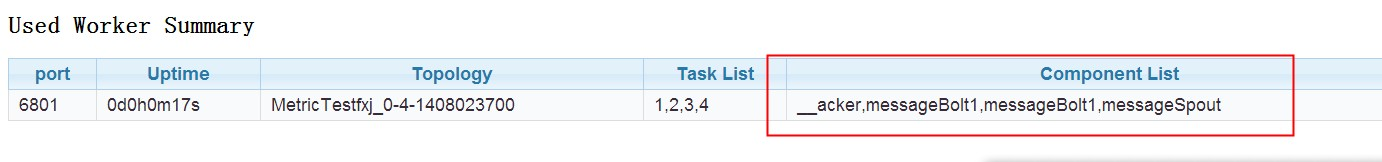


**测试目标:**

自定义的worker是否按设置被调度到了相应supervisor上

**测试结果:**





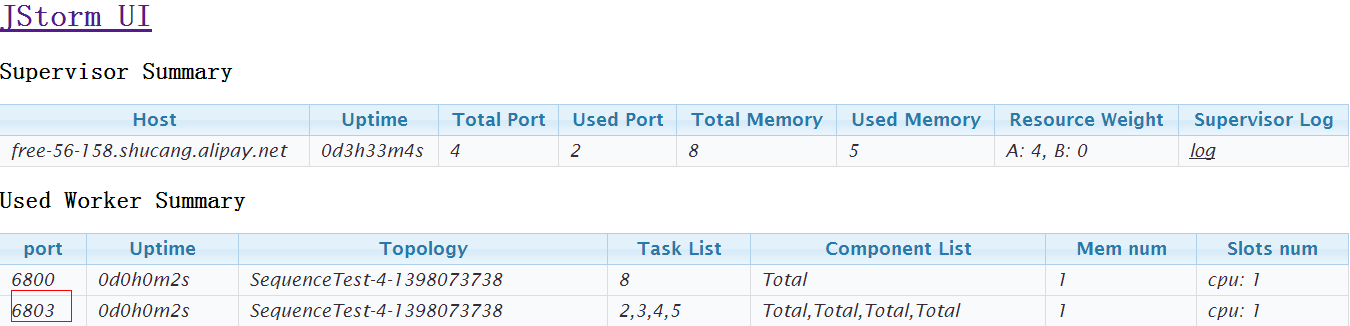
4个total bolt task被分配到一个worker并且被分配到了指定Hostname上

## 4.8 自定义端口号被占用

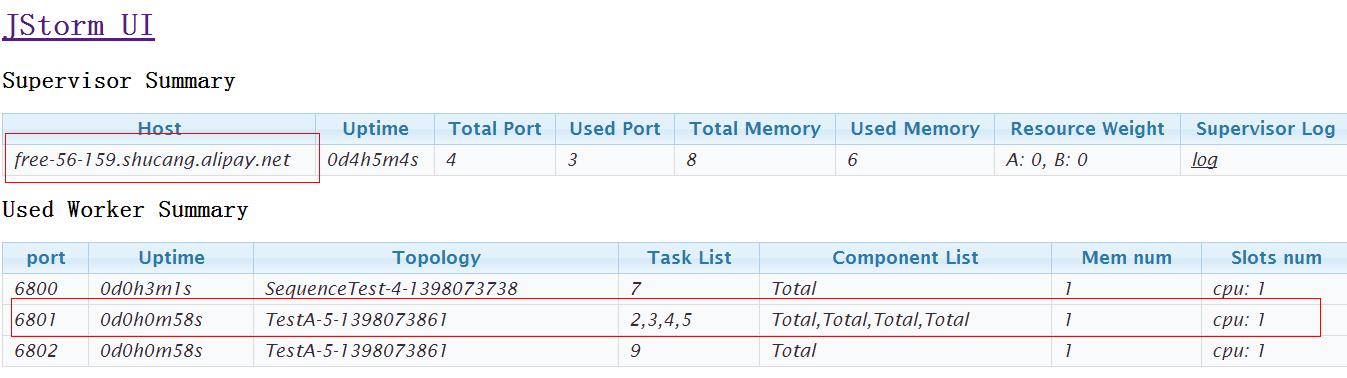
**测试目标:**

当端口号被占用时，自定义的worker是否按照正常调度选择其他supervisor

**测试结果:**



158的6803被占用,使用默认调度:



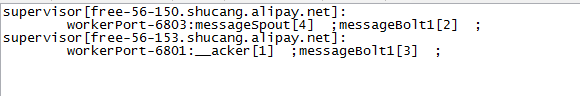
# 5 old任务调度测试

## 5.1 测试old任务调用有效性

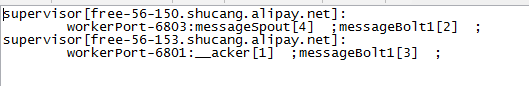
* 预先设置好用户的old assignment，



已知zk上存在的old assignment如下：



* 再跑新的拓扑，任务调度将会参照zk存储的任务调度，结果：



与之前保持一致

old assignment 使用合理，符合预期结果

## 5.2 新跑的拓扑与old assignment不一致（略）

**Old assignment能够在满足新跑拓扑的配置：**

则采用old assignment

**Old assignment的worker数量小于新跑拓扑配置的worker数量：**

则放弃old assignment

**Old assignment的部分满足新跑拓扑的配置：**

则部分采用old assignment，余下采用默认调度

# 6 old assignment和用户自定义同时启用

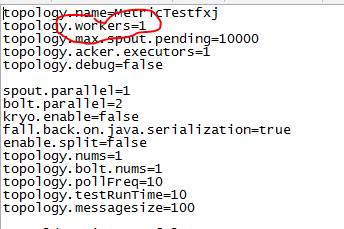
优先级顺序：用户自定义>old assignment，符合预期

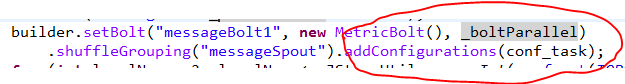
# 6 测试task.on.differ.node

task.on.differ.node可以强制设置同类型的task在不同supervisor上

## 6.1 普通例子1

* 配置：



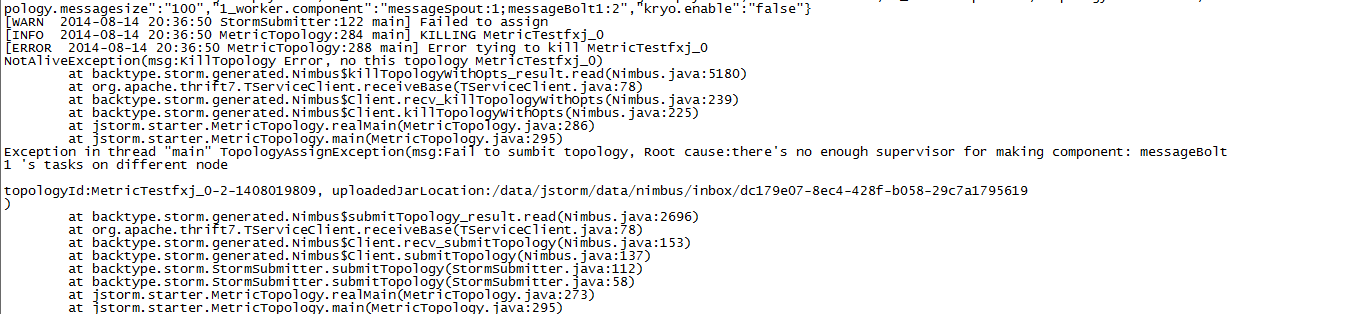


conf\_task是map结构，该语句限制了bolt-task必须再不同supervisor上

* 测试目标：

无法进行调度分配，因为conf\_task和worker=1相互矛盾

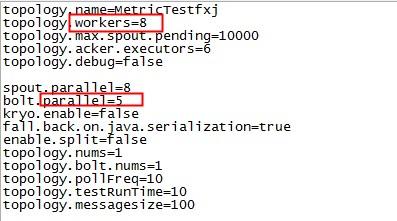
* 结果

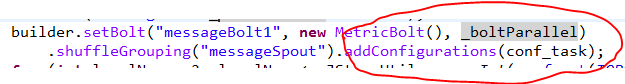


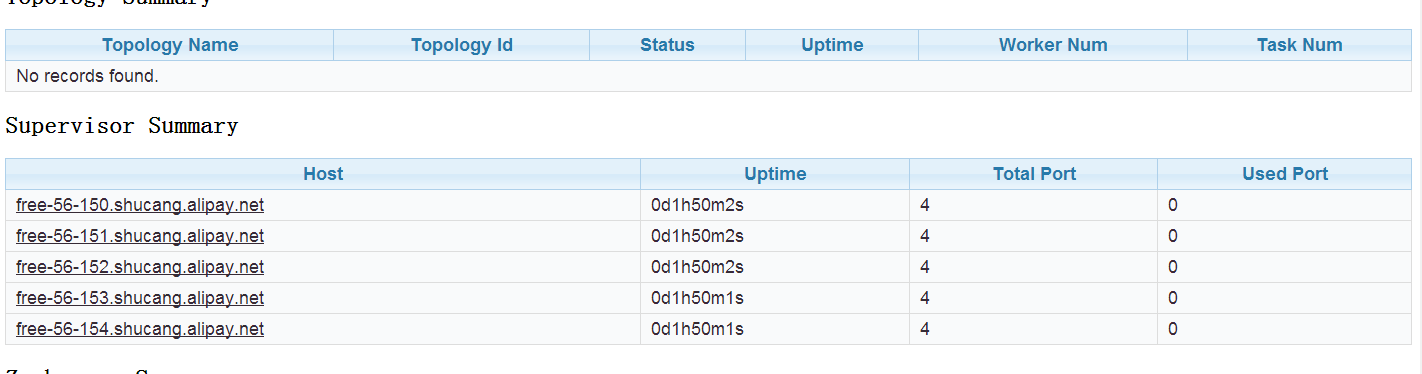
符合预期

## 6.2普通例子2

* 配置：







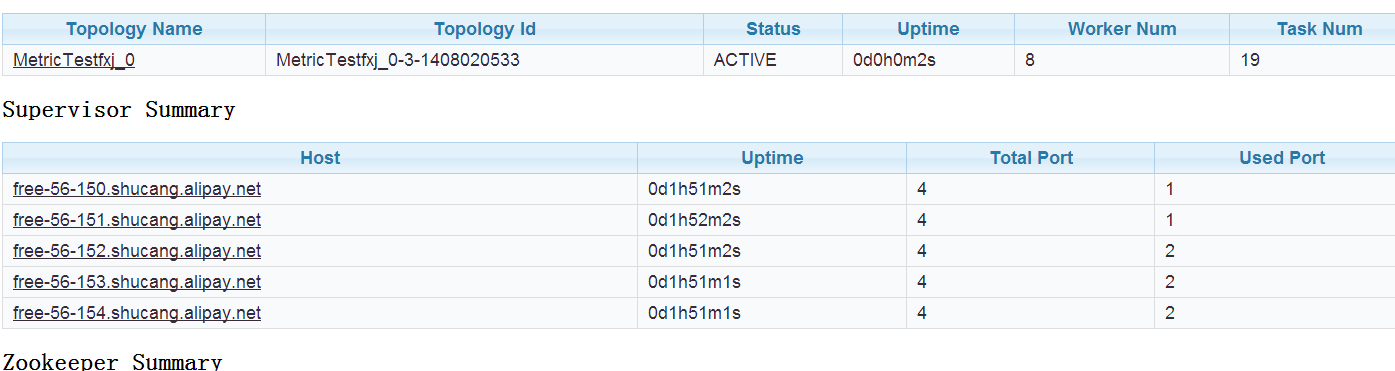
conf\_task是map结构，该语句限制了bolt-task必须再不同supervisor上；

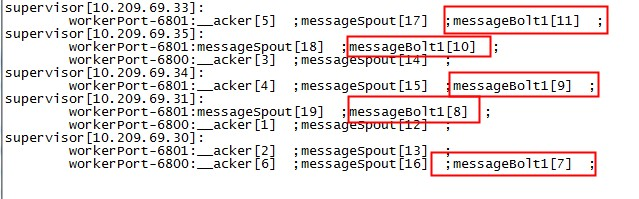
这里只有5个suprvisor

* 测试目标：

bolt-task将被分配到不同supervisor上去

* 结果

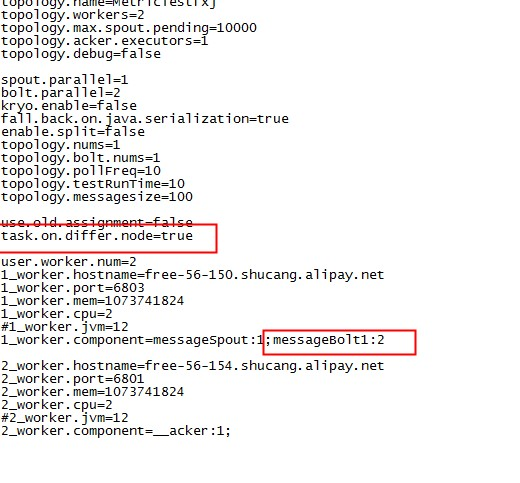




符合预期

## 6.3普通例子3

配置：

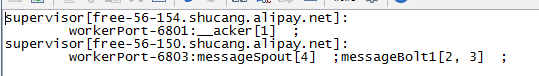


自定义任务调度和 task.on.differ.node设置相矛盾

* 测试目标：

当自定义任务调度和 task.on.differ.node设置相矛盾时，优先满足自定义任务调度；

* 结果



符合预期

# 7 测试supervisor.use.ip=true调度

## 7.1 常规测试

* **配置：**task.on.differ.node=false，topology\_nums=1, topology\_level=1, workers=10

spout.parallel=10, bolt\_0.parallel=15 acker.executors=10

机1端口数量=4、机1端口数量=4、机1端口数量=4、机1端口数量=4、

机1端口数量=4

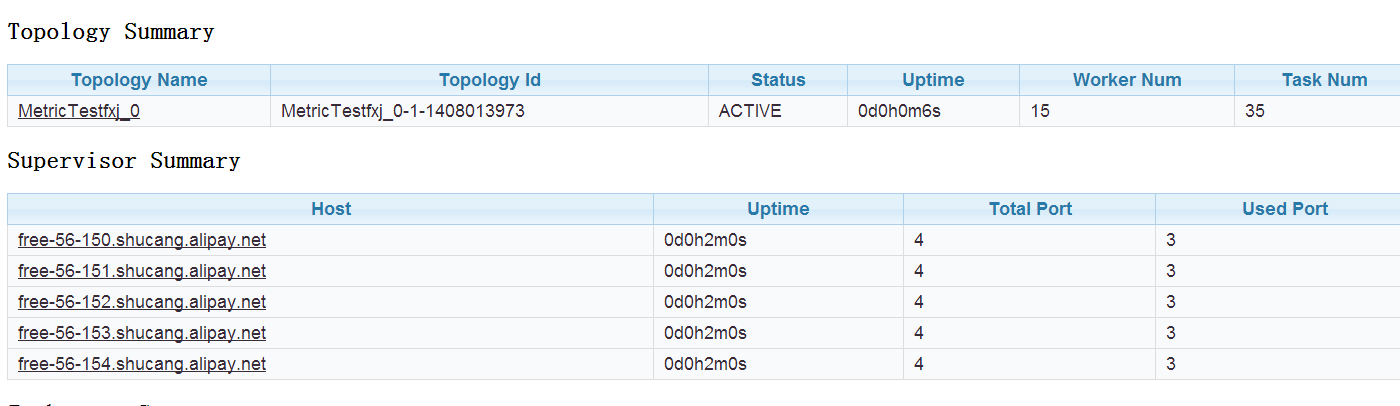
* **测试目标**

**①**以worker为维度，尽量将worker平均分配到各个supervisor上；

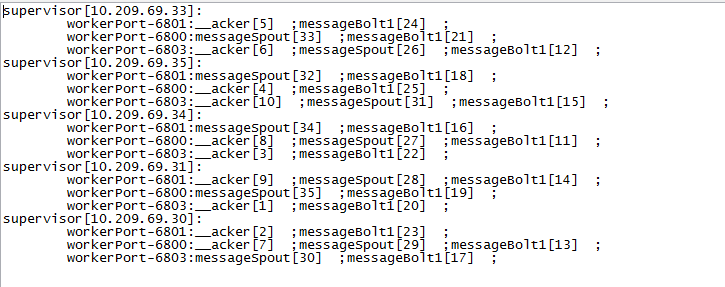
②保证同类型task尽量均分到不同worker和supervisor的基础上，其次保证每一个worker和supervisor的task数量均衡，最后保证直接相关的task放到一个进程上去；

* **测试结果:**

Worker被一一分配到supervisor上



Task按规定分配到work上去



**合理**

## 7.2 old Assignment 测试

* **配置：**task.on.differ.node=false，topology\_nums=1, topology\_level=1, workers=10

spout.parallel=10, bolt\_0.parallel=15 acker.executors=10

机1端口数量=4、机1端口数量=4、机1端口数量=4、机1端口数量=4、

机1端口数量=4 临时增加一个机器机1端口数量=4

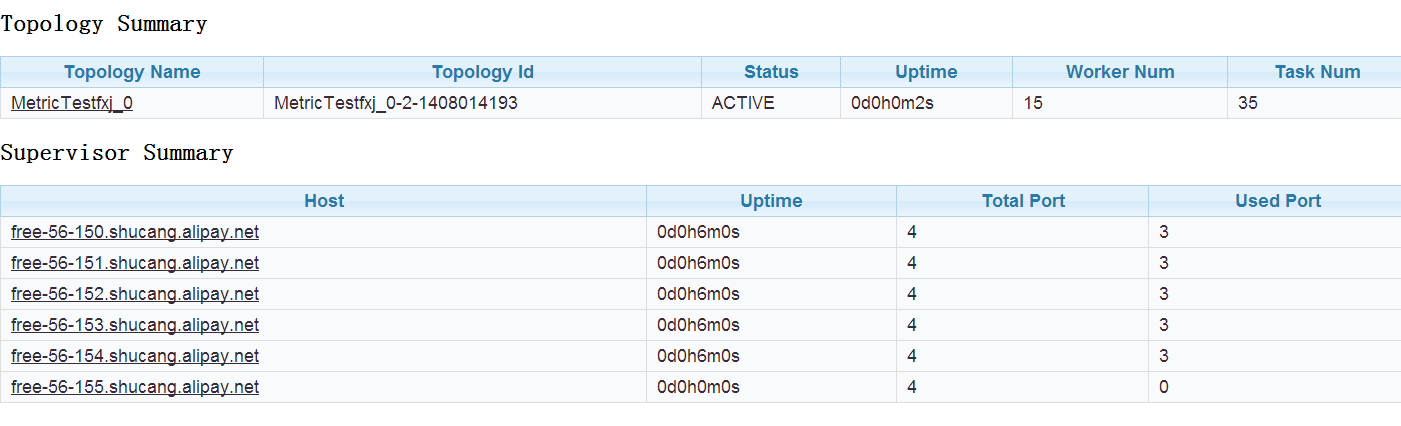
* **测试目标**

**①**以worker为维度，尽量将worker平均分配到各个supervisor上；

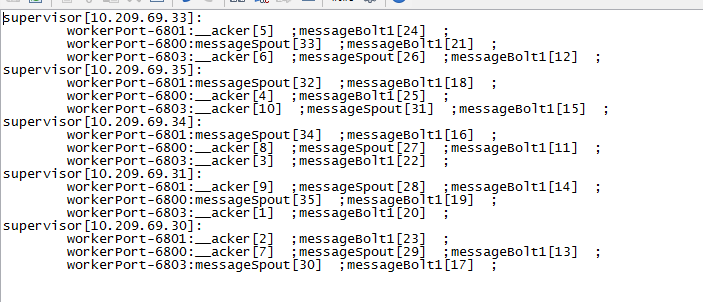
②保证同类型task尽量均分到不同worker和supervisor的基础上，其次保证每一个worker和supervisor的task数量均衡，最后保证直接相关的task放到一个进程上去；

* **测试结果:**

Worker被一一分配到supervisor上



Task按规定分配到work上去



合理

**注意：**

当old Assignment中的supervisor对应的hostname，和supervisor.use.ip=true不匹配时，这时采用old Assignment生产的任务调度结果将与预期不符。（意思是说old Assignment中的supervisor对应的hostname如果是域名的话，而配置了supervisor.use.ip=true后，现在的supervisor上的hostname是ip，这样就不匹配了）

## 7.3自定义任务调度（略）

一切正常

**注意：**

当supervisor.use.ip=true，用户自定义任务调度的hostname也应对应ip，如果用的是域名的话，如下图所示：



这样设置用户自定义的任务调度不符合预期。