Balance优化测试

## 优化目的

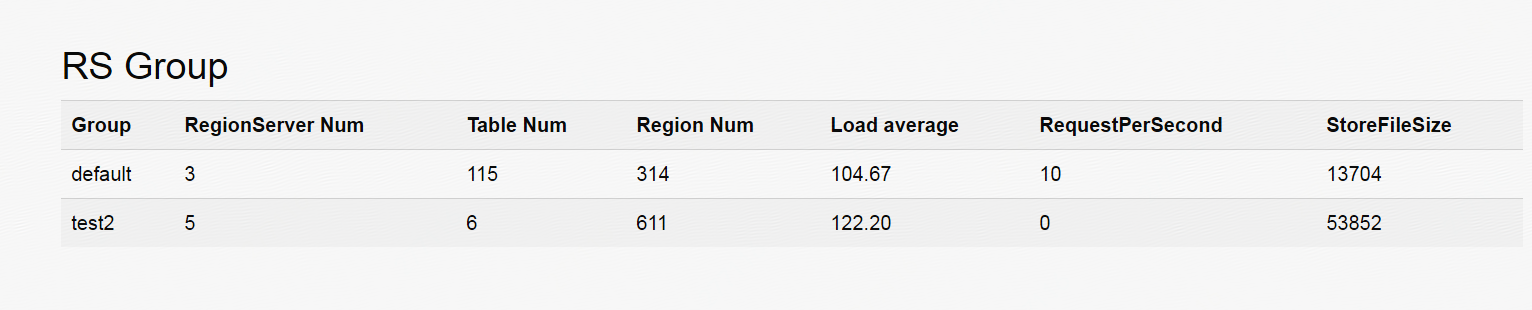
线上集群当前Balance策略仅考虑Regionserver粒度的负载均衡，此次优化目的是在保证在RegionServer均衡的情况下，综合考虑表均衡、数据本地性等因素，能更快地触发Balance实现更好的均衡，以满足更多业务场景。

## 测试环境

测试集群：STARFISH，8台Regionserver

硬件配置：32core+256g,1T\*12 SSD

集群配置：2个rsgroup：default（3台RS，多小表）；test2（5台RS，多大表）



压测配置：两台发压机ycsb进行单条2k大小的顺序写压测，单台起50client线程。（为更快split，将集群配置hbase.hregion.max.filesize大小调整为5G）

## 优化流程

约定前提：压测前，集群是均衡的。

### 重写Balance触发函数

当前Balance触发机制仅考虑regionserver粒度的负载均衡（regioncount）,未考虑表region分布倾斜、数据本地性倾斜等场景，在相应场景无法及时触发Balance机制，影响线上业务读写效率，增大集群部分节点压力。

现在重写Balance触发函数，将影响Balance的因素均纳入考虑范围，完成后进行压测测试。

#### 多小表场景

此种场景下，压测表split分裂成16个region后首次触发Balance，表region在3台regionserver上的分布如下：[4,10,2],三台regionserver的负载分布情况如下：[110,109,110]。

#### 多大表场景

此种场景下， 压测表split分裂成44个region后首次触发Balance，表region在5台regionserver上的分布如下：[34,3,1,2,4]，5台regionserver的负载分布情况如下：[131,131,130,131,131]。

#### 存在的问题

单表写入量很大的场景下，集群需要很长时间去触发Balance，在多大表场景下更甚；触发Balance后，各regionserver能达到良好的负载均衡，但表region分布有较大倾斜。

### 优化Balance算法

根据上述情况可以发现，单region表大量写入会发生长时间的表数据倾斜，才会触发Balance阀值，增大单点压力，同时Balance后存在较大的表倾斜，因此需要优化Balance算法，能更及时地感知并触发Balance机制，并达到表region均衡。

#### 多小表场景

此种场景下，压测表split分裂成5个region后首次触发Balance，表region在3台regionserver上的分布如下：[1,2,2],三台regionserver的负载分布情况如下：[106,106,106]。观察后续Balance触发、执行情况均符合预期。

#### 多大表场景

此种场景下，压测表split分裂成6个region后首次触发Balance，压测表表region在5台regionserver上的分布如下：[1,1,1,1,2]，5台regionserver的负载分布情况如下：[123,124,123,123,123]。观察后续Balance触发、执行情况均符合预期。

#### 存在的问题

运行一段时间，HMaster产生长GC（数十秒），引起HMaster主从切换。

#### 优化总结

* HMaster调大内存（5g->20g），采用CMS的GC策略
* 优化后Balance触发时机至少提快3倍
* 优化后Balance执行效果接近全局最优解，符合优化预期

### 稳定性验证

集群各项指标均正常，TP性能均在正常范围。

### 主要优化点&解决问题

1. 重写balance触发函数
2. 修复获取表在单节点最大region数不准确的问题
3. 修复locality picker不选择低本地性region进行balance的问题
4. 减小balance在内存的开销
5. 修复rs负载相同时，picker总是选择第一个region而不是random的问题
6. 重写locality代价函数，优化本地性计算算法
7. 优化Balance权重算法
8. 重写表倾斜代价函数，优化swap逻辑

## 后续解决问题

* 解决热region问题
* 增加相应监控告警机制