Kafka设计解析：Replication工具

Topic Tool

$KAFKA_HOME/bin/kafka-topics.sh，该工具可用于创建、删除、修改、查看某个Topic，也可用于列出所有Topic。另外，该工具还可修改某个Topic的以下配置。

unclean.leader.election.enable
delete.retention.ms
segment.jitter.ms
retention.ms
flush.ms
segment.bytes
flush.messages
segment.ms
retention.bytes
cleanup.policy
segment.index.bytes
min.cleanable.dirty.ratio
max.message.bytes
file.delete.delay.ms
min.insync.replicas
index.interval.bytes
Replica Verification Tool

$KAFKA_HOME/bin/kafka-replica-verification.sh，该工具用来验证所指定的一个或多个Topic下每个Partition对应的所有Replica是否都同步。可通过topic-white-list这一参数指定所需要验证的所有Topic，支持正则表达式。

Preferred Replica Leader Election Tool

用途

有了Replication机制后，每个Partition可能有多个备份。某个Partition的Replica列表叫作AR

（Assigned Replicas），AR中的第一个Replica即为“Preferred Replica”。创建一个新的Topic或者给已有Topic增加Partition时，Kafka保证Preferred Replica被均匀分布到集群中的所有Broker上。理想情况下，Preferred Replica会被选为Leader。以上两点保证了所有Partition的Leader被均匀分布到了集群当中，这一点非常重要，因为所有的读写操作都由Leader完成，若Leader分布过于集中，会造成集群负载不均衡。但是，随着集群的运行，该平衡可能会因为Broker的宕机而被打破，该工具就是用来帮助恢复Leader分配的平衡。

事实上，每个Topic从失败中恢复过来后，它默认会被设置为Follower角色，除非某个Partition的Replica全部宕机，而当前Broker是该Partition的AR中第一个恢复回来的Replica。因此，某个Partition的Leader（Preferred Replica）宕机并恢复后，它很可能不再是该Partition的Leader，但仍然是Preferred Replica。

原理

1. 在ZooKeeper上创建/admin/preferred_replica_election节点，并存入需要调整Preferred Replica的Partition信息。
2. Controller一直Watch该节点，一旦该节点被创建，Controller会收到通知，并获取该内容。
3. Controller读取Preferred Replica，如果发现该Replica当前并非是Leader并且它在该Partition的ISR中，Controller向该Replica发送LeaderAndIsrRequest，使该Replica成为Leader。如果该Replica当前并非是Leader，且不在ISR中，Controller为了保证没有数据丢失，并不会将其设置为Leader。

用法

```
$KAFKA_HOME/bin/kafka-preferred-replica-election.sh --zookeeper www.iteblog.com:2181
```

在包含8个Broker的Kafka集群上，创建1个名为topic1，replication-factor为3，Partition数为8的Topic，使用如下命令查看其Partition/Replica分布。

```
$KAFKA_HOME/bin/kafka-topics.sh --describe --topic iteblog --zookeeper www.iteblog.com:2181
```

查询结果如下图所示，从图中可以看到，Kafka将所有Replica均匀分布到了整个集群，并且Leader也均匀分布:

```
Topic:iteblog PartitionCount:8 ReplicationFactor:1 Configs:
 Topic: iteblog Partition: 0 Leader: 1 Replicas: 1,3,4 Isr: 4,1,3
 Topic: iteblog Partition: 1 Leader: 2 Replicas: 2,4,5 Isr: 5,2,4
```

Topic: iteblog Partition: 2 Leader: 3 Replicas: 3,5,6 Isr: 5,3,6
Topic: iteblog Partition: 3 Leader: 4 Replicas: 4,6,7 Isr: 4,7,6
Topic: iteblog Partition: 4 Leader: 5 Replicas: 5,7,0 Isr: 5,7,0
Topic: iteblog Partition: 5 Leader: 6 Replicas: 6,0,1 Isr: 0,6,1
Topic: iteblog Partition: 6 Leader: 7 Replicas: 7,1,2 Isr: 7,1,2
Topic: iteblog Partition: 7 Leader: 0 Replicas: 0,2,3 Isr: 0,2,3

手动停止部分Broker，topic1的Partition/Replica分布如下图所示。从图中可以看到，由于Broker 1/2/4都被停止，Partition 0的Leader由原来的1变为3，Partition 1的Leader由原来的2变为5，Partition 2的Leader由原来的3变为6，Partition 3的Leader由原来的4变为7。

Topic:iteblog PartitionCount:8 ReplicationFactor:1 Configs:
Topic: iteblog Partition: 0 Leader: 3 Replicas: 1,3,4 Isr: 3
Topic: iteblog Partition: 1 Leader: 5 Replicas: 2,4,5 Isr: 5
Topic: iteblog Partition: 2 Leader: 6 Replicas: 3,5,6 Isr: 5,3,6
Topic: iteblog Partition: 3 Leader: 7 Replicas: 4,6,7 Isr: 7,6
Topic: iteblog Partition: 4 Leader: 5 Replicas: 5,7,0 Isr: 5,7,0
Topic: iteblog Partition: 5 Leader: 6 Replicas: 6,0,1 Isr: 0,6
Topic: iteblog Partition: 6 Leader: 7 Replicas: 7,1,2 Isr: 7
Topic: iteblog Partition: 7 Leader: 0 Replicas: 0,2,3 Isr: 0,3

再重新启动ID为1的Broker，topic1的Partition/Replica分布如下。可以看到，虽然Broker 1已经启动（Partition 0和Partition5的ISR中有1），但是1并不是任何一个Parititon的Leader，而Broker 5/6/7都是2个Partition的Leader，即Leader的分布不均衡——一个Broker最多是2个Partition的Leader，而最少是0个Partition的Leader:

Topic:iteblog PartitionCount:8 ReplicationFactor:1 Configs:
Topic: iteblog Partition: 0 Leader: 3 Replicas: 1,3,4 Isr: 3,1
Topic: iteblog Partition: 1 Leader: 5 Replicas: 2,4,5 Isr: 5
Topic: iteblog Partition: 2 Leader: 6 Replicas: 3,5,6 Isr: 5,3,6
Topic: iteblog Partition: 3 Leader: 7 Replicas: 4,6,7 Isr: 7,6
Topic: iteblog Partition: 4 Leader: 5 Replicas: 5,7,0 Isr: 5,7,0
Topic: iteblog Partition: 5 Leader: 6 Replicas: 6,0,1 Isr: 0,6,1
Topic: iteblog Partition: 6 Leader: 7 Replicas: 7,1,2 Isr: 7,1
Topic: iteblog Partition: 7 Leader: 0 Replicas: 0,2,3 Isr: 0,3

运行该工具后，topic1的Partition/Replica分布如下图所示。由图可见，除了Partition 1和Partition 3由于Broker 2和Broker 4还未启动，所以其Leader不是其Preferred Repliac外，其它所有Partition的Leader都是其Preferred Replica。同时，与运行该工具前相比，Leader的分配更均匀——一个Broker最多是2个Parittion的Leader，最少是1个Partition的Leader:

```
Topic:iteblog PartitionCount:8 ReplicationFactor:1 Configs:
 Topic: iteblog Partition: 0 Leader: 1 Replicas: 1,3,4 Isr: 3,1
 Topic: iteblog Partition: 1 Leader: 5 Replicas: 2,4,5 Isr: 5
 Topic: iteblog Partition: 2 Leader: 3 Replicas: 3,5,6 Isr: 5,3,6
 Topic: iteblog Partition: 3 Leader: 7 Replicas: 4,6,7 Isr: 7,6
 Topic: iteblog Partition: 4 Leader: 5 Replicas: 5,7,0 Isr: 5,7,0
 Topic: iteblog Partition: 5 Leader: 6 Replicas: 6,0,1 Isr: 0,6,1
 Topic: iteblog Partition: 6 Leader: 7 Replicas: 7,1,2 Isr: 7,1
 Topic: iteblog Partition: 7 Leader: 0 Replicas: 0,2,3 Isr: 0,3
```

启动Broker 2和Broker 4，Leader分布与上一步相比并未变化，如下图所示。：

```
Topic:iteblog PartitionCount:8 ReplicationFactor:1 Configs:
 Topic: iteblog Partition: 0 Leader: 1 Replicas: 1,3,4 Isr: 3,1,4
 Topic: iteblog Partition: 1 Leader: 5 Replicas: 2,4,5 Isr: 5,2,4
 Topic: iteblog Partition: 2 Leader: 3 Replicas: 3,5,6 Isr: 5,3,6
 Topic: iteblog Partition: 3 Leader: 7 Replicas: 4,6,7 Isr: 7,6,4
 Topic: iteblog Partition: 4 Leader: 5 Replicas: 5,7,0 Isr: 5,7,0
 Topic: iteblog Partition: 5 Leader: 6 Replicas: 6,0,1 Isr: 0,6,1
 Topic: iteblog Partition: 6 Leader: 7 Replicas: 7,1,2 Isr: 7,1,2
 Topic: iteblog Partition: 7 Leader: 0 Replicas: 0,2,3 Isr: 0,3,2
```

再次运行该工具，所有Partition的Leader都由其Preferred Replica承担，Leader分布更均匀——每个Broker承担1个Partition的Leader角色。

除了手动运行该工具使Leader分配均匀外，Kafka还提供了自动平衡Leader分配的功能，该功能可通过将auto.leader.rebalance.enable设置为true开启，它将周期性检查Leader分配是否平衡，若不平衡度超过一定阈值则自动由Controller尝试将各Partition的Leader设置为其Preferred Replica。检查周期由leader.imbalance.check.interval.seconds指定，不平衡度阈值由leader.imbalance.per.broker.percentage指定。

Kafka Reassign Partitions Tool

用途

该工具的设计目标与Preferred Replica Leader Election Tool有些类似，都旨在促进Kafka集群的负载均衡。不同的是，Preferred Replica Leader Election只能在Partition的AR范围内调整其Leader，使Leader分布均匀，而该工具还可以调整Partition的AR。

Follower需要从Leader Fetch数据以保持与Leader同步，所以仅仅保持Leader分布的平衡对整个集群的负载均衡来说是不够的。另外，生产环境下，随着负载的增大，可能需要给Kafka集群扩容。向Kafka集群中增加Broker非常简单方便，但是对于已有的Topic，并不会自动将其Partition迁移到新加入的Broker上，此时可用该工具达到此目的。某些场景下，实际负载可能远小于最初预期负载，此时可用该工具将分布在整个集群上的Partition重装分配到某些机器上，然后可以停止不需要的Broker从而实现节约资源的目的。

需要说明的是，该工具不仅可以调整Partition的AR位置，还可调整其AR数量，即改变该Topic的replication factor。

原理

该工具只负责将所需信息存入ZooKeeper中相应节点，然后退出，不负责相关的具体操作，所有调整都由Controller完成。

1. 在ZooKeeper上创建/admin/reassign_partitions节点，并存入目标Partition列表及其对应的目标AR列表。
2. Controller注册在/admin/reassign_partitions上的Watch被fire，Controller获取该列表。
3. 对列表中的所有Partition，Controller会做如下操作：

（1）、启动RAR - AR中的Replica，即新分配的Replica。（RAR = Reassigned Replicas，AR = Assigned Replicas）
（2）、等待新的Replica与Leader同步
（3）、如果Leader不在RAR中，从RAR中选出新的Leader
（4）、停止并删除AR - RAR中的Replica，即不再需要的Replica
（5）、删除/admin/reassign_partitions节点

用法

该工具有三种使用模式

（1）、generate模式，给定需要重新分配的Topic，自动生成reassign plan（并不执行）
（2）、execute模式，根据指定的reassign plan重新分配Partition
（3）、verify模式，验证重新分配Partition是否成功

下面这个例子将使用该工具将Topic的所有Partition重新分配到Broker 2,3 上，步骤如下：

1. 使用generate模式，生成reassign plan
指定需要重新分配的Topic（{"topics":[{"topic":"iteblog"}],"version":1}），并存入/tmp/topics-to-move.json文件中，然后执行如下命令

```
$KAFKA_HOME/bin/kafka-reassign-partitions.sh --zookeeper www.iteblog.com:2181
--topics-to-move-json-file /tmp/topics-to-move.json
--broker-list "2,3" --generate
```

结果如下图所示

```
Current partition replica assignment

{"version":1,"partitions":[{"topic":"iteblog","partition":2,"replicas":[3]},{"topic":"iteblog","partition":6,"replicas":[1]},{"topic":"iteblog","partition":7,"replicas":[2]},{"topic":"iteblog","partition":0,"replicas":[1]},{"topic":"iteblog","partition":8,"replicas":[3]},{"topic":"iteblog","partition":5,"replicas":[3]},{"topic":"iteblog","partition":4,"replicas":[2]},{"topic":"iteblog","partition":3,"replicas":[1]},{"topic":"iteblog","partition":1,"replicas":[2]}]}
Proposed partition reassignment configuration

{"version":1,"partitions":[{"topic":"iteblog","partition":6,"replicas":[3]},{"topic":"iteblog","partition":2,"replicas":[3]},{"topic":"iteblog","partition":7,"replicas":[2]},{"topic":"iteblog","partition":0,"replicas":[3]},{"topic":"iteblog","partition":8,"replicas":[3]},{"topic":"iteblog","partition":5,"replicas":[2]},{"topic":"iteblog","partition":4,"replicas":[3]},{"topic":"iteblog","partition":1,"replicas":[2]},{"topic":"iteblog","partition":3,"replicas":[2]}]}
```

2. 使用execute模式，执行reassign plan

将上一步生成的reassignment plan存入/tmp/reassign-plan.json文件中，并执行

```
$KAFKA_HOME/bin/kafka-reassign-partitions.sh --zookeeper www.iteblog.com:2181
--reassignment-json-file /tmp/reassign-plan.json --execute
```

```
Current partition replica assignment

{"version":1,"partitions":[{"topic":"iteblog","partition":2,"replicas":[3]},{"topic":"iteblog","partition":6,"replicas":[1]},{"topic":"iteblog","partition":7,"replicas":[2]},{"topic":"iteblog","partition":0,"replicas":[1]},{"topic":"iteblog","partition":8,"replicas":[3]},{"topic":"iteblog","partition":5,"replicas":[3]},{"topic":"iteblog","partition":4,"replicas":[2]},{"topic":"iteblog","partition":3,"replicas":[1]},{"topic":"iteblog","partition":1,"replicas":[2]}]}

Save this to use as the --reassignment-json-file option during rollback
```

Successfully started reassignment of partitions {"version":1,"partitions":[{"topic":"iteblog","partition":2,"replicas":[3]},{"topic":"iteblog","partition":6,"replicas":[3]},{"topic":"iteblog","partition":7,"replicas":[2]},{"topic":"iteblog","partition":0,"replicas":[3]},{"topic":"iteblog","partition":8,"replicas":[3]},{"topic":"iteblog","partition":4,"replicas":[3]},{"topic":"iteblog","partition":5,"replicas":[2]},{"topic":"iteblog","partition":3,"replicas":[2]},{"topic":"iteblog","partition":1,"replicas":[2]}]}

此时，ZooKeeper上/admin/reassign_partitions节点被创建，且其值与/tmp/reassign-plan.json文件的内容一致。

[zk: www.iteblog.com:2181(CONNECTED) 12] get /admin/reassign_partitions
{"version":1,"partitions":[{"topic":"iteblog","partition":2,"replicas":[3]},{"topic":"iteblog","partition":6,"replicas":[3]},{"topic":"iteblog","partition":7,"replicas":[2]},{"topic":"iteblog","partition":0,"replicas":[3]},{"topic":"iteblog","partition":8,"replicas":[3]},{"topic":"iteblog","partition":4,"replicas":[3]},{"topic":"iteblog","partition":5,"replicas":[2]},{"topic":"iteblog","partition":3,"replicas":[2]},{"topic":"iteblog","partition":1,"replicas":[2]}]}

3. 使用verify模式，验证reassign是否完成

执行verify命令

```
$KAFKA_HOME/bin/kafka-reassign-partitions.sh --zookeeper www.iteblog.com:2181 --reassignment-json-file /tmp/reassign-plan.json --verify
```

结果如下所示，从图中可看出iteblog的所有Partititon都根据reassign plan重新分配成功:

```
Status of partition reassignment:
Reassignment of partition [iteblog,4] completed successfully
Reassignment of partition [iteblog,0] completed successfully
Reassignment of partition [iteblog,2] completed successfully
Reassignment of partition [iteblog,6] completed successfully
Reassignment of partition [iteblog,7] completed successfully
Reassignment of partition [iteblog,5] completed successfully
Reassignment of partition [iteblog,3] completed successfully
Reassignment of partition [iteblog,8] completed successfully
Reassignment of partition [iteblog,1] completed successfully
```

接下来用Topic Tool再次验证。

bin/kafka-topics.sh --zookeeper www.iteblog.com:2181 --describe --topic iteblog

结果如下图所示，从图中可看出iteblog 的所有Partition都被重新分配到Broker 2/3，且每个Partition的AR与reassign plan一致。

Topic:iteblog PartitionCount:9 ReplicationFactor:1 Configs:
Topic: iteblog Partition: 0 Leader: 3 Replicas: 3 Isr: 3
Topic: iteblog Partition: 1 Leader: 2 Replicas: 2 Isr: 2
Topic: iteblog Partition: 2 Leader: 3 Replicas: 3 Isr: 3
Topic: iteblog Partition: 3 Leader: 2 Replicas: 2 Isr: 2
Topic: iteblog Partition: 4 Leader: 3 Replicas: 3 Isr: 3
Topic: iteblog Partition: 5 Leader: 2 Replicas: 2 Isr: 2
Topic: iteblog Partition: 6 Leader: 3 Replicas: 3 Isr: 3
Topic: iteblog Partition: 7 Leader: 2 Replicas: 2 Isr: 2
Topic: iteblog Partition: 8 Leader: 3 Replicas: 3 Isr: 3

这里附出执行上面命令之前iteblog的分布情况：

bin/kafka-topics.sh --zookeeper www.iteblog.com:2181 --describe --topic iteblog

Topic:iteblog PartitionCount:9 ReplicationFactor:1 Configs:
Topic: iteblog Partition: 0 Leader: 1 Replicas: 1 Isr: 1
Topic: iteblog Partition: 1 Leader: 2 Replicas: 2 Isr: 2
Topic: iteblog Partition: 2 Leader: 3 Replicas: 3 Isr: 3
Topic: iteblog Partition: 3 Leader: 1 Replicas: 1 Isr: 1
Topic: iteblog Partition: 4 Leader: 2 Replicas: 2 Isr: 2
Topic: iteblog Partition: 5 Leader: 3 Replicas: 3 Isr: 3
Topic: iteblog Partition: 6 Leader: 1 Replicas: 1 Isr: 1
Topic: iteblog Partition: 7 Leader: 2 Replicas: 2 Isr: 2
Topic: iteblog Partition: 8 Leader: 3 Replicas: 3 Isr: 3

需要说明的是，在使用execute之前，并不一定要使用generate模式自动生成reassign

plan，使用generate模式只是为了方便。事实上，某些场景下，generate模式生成的reassign plan并不一定能满足需求，此时用户可以自己设置reassign plan。

State Change Log Merge Tool

用途

该工具旨在从整个集群的Broker上收集状态改变日志，并生成一个集中的格式化的日志以帮助诊断状态改变相关的故障。每个Broker都会将其收到的状态改变相关的的指令存于名为state-change.log的日志文件中。某些情况下，Partition的Leader election可能会出现问题，此时我们需要对整个集群的状态改变有个全局的了解从而诊断故障并解决问题。该工具将集群中相关的state-change.log日志按时间顺序合并，同时支持用户输入时间范围和目标Topic及Partition作为过滤条件，最终将格式化的结果输出。

用法

```
bin/kafka-run-class.sh kafka.tools.StateChangeLogMerger
--logs /opt/kafka_2.11-0.8.2.1/logs/state-change.log
--topic iteblog --partitions 0,1,2,3,4,5,6,7
```