Balancing Kafka on JBOD

At Automattic we run a diverse array of systems and as with many companies Kafka is the glue that ties them together; letting us to shuffle data back and forth. Our experience with Kafka have thus far been fantastic, it's stable, provides excellent throughput, and the simple API makes it trivial to hook any of our systems up to it. In fact it's been so popular that we've been steadily piping more and more data through it over the past year. Now we're starting to run out of disk space necessitating an expansion of the cluster.

The expansion plan was pretty straight forward. Spin up some more brokers, have them join the same cluster, then when they are all up simply use Kafka's provided `kafka-reassign-partitions.sh` script to rebalance all topics across all brokers.

We then thought, with more available capacity why not also do a rolling upgrade of our cluster to get on the latest distro across all our servers? And seeing as we run most topics with a replication factor of 2 so it seemed better to use the reassign script to move data off each server as we upgrade as to not have partitions run with only a single copy of the data at any time during the upgrade process.

The Problem

The expansion and rolling restart method seemed like a a good plan however after the upgrade we discovered that while the reassign script balances partitions between servers, those servers do not balance each topic between the configured `log.dirs`. This lead to major imbalances on each broker causing some disks to be virtually empty while other disks to be almost entirely full. A quick search showed others have also run into similar issues with no good way of fixing it.

Why Did It Happen?

The problem stems from the fact that we run our Kafka cluster in JBOD mode, meaning we don't RAID or combine the disks on the hardware or OS level. Instead we mount all disks then configure Kafka's `log.dirs` to use all drives on the server. When run this way Kafka distributes partitions assigned to each broker evenly across all log dirs available to it.

> *Each new partition that is created will be placed in the directory which currently has the fewest partitions.*
> –Kafka Docs

Unfortunately, we have some quite unbalanced topics. Some topics are huge on the order of TBs, with long retention periods. Then there are other topics that perhaps are not yet fully productionized with only a couple MB or KB in each partition. By simply balancing the *number of partitions* across `log.dirs` we end up with some disks with mostly large partitions and some with mostly small partitions.

Hack Partition Reassignment

There is currently some talk about trying to make JBOD support in Kafka better but we don't want to run with such unbalanced disk use nor did we want to bring brokers down to try and hack the metadata in order to move files around in the interim. Luckily how Kafka brokers assign partitions and moves data during partition reassignment operations is well

defined which means we can force each broker to balance its data through the use of partition reassignment if we control it very carefully. The basic concept is pretty simple:

1. **Ensure the topic we want to distribute has the right number of partitions.** In order to achieve an even distribution of a topic we need to make sure we have enough partitions to evenly distribute in the first place. This means we'll need to make sure that the `number of partitions × replication factor` is a multiple of the `number of brokers × number of disks` on each broker. If this is not done we will not end up with exactly the same number of partitions per disk across all brokers and disks.
2. **Balance partition counts across disks on all brokers.** We want every disks to have the same number of partitions on each broker before we move anything. Doing so ensures that when new partitions are created they will be round robined across every available disk on each broker evenly. When a broker is assigned a new partition Kafka assigns that partition to one of the directories configured in `log.dirs` that has the fewest partitions. Without first balancing partition counts across disks these assignment operations will not assign the topic being moved across disks evenly.
3. **Move one topic at a time but move all partitions of that topic together.** When Kafka reassigns partitions the new partition is created on the broker it's being moved to and then synced with the leader. By moving every replica of every partition of a single topic together Kafka will be forced to create a new copy of each partition thus distributing all those partitions evenly.

Tips and Hints

The easiest way to make sure partition counts are balanced across all disks of all brokers is to simply create a new topic and manually assign the right number of partitions to each broker. To do this go to each of your brokers and count up how many partitions you have on each disk then figure out how many more partitions you need on each broker to bring the partition count of the broker up to be the `max partitions on any one disk × number of disks`. For example if you have 3 disks and with 10 partitions on disk A and 6 partitions on disk B & C you'll need 8 more partitions to bring the total partition count of that broker up to 30.

Once you have all the counts simply create a new topic with the number of new partitions needed for each broker. Continuing with the above example if you've determined that broker "1" needs 8 more partitions and broker "2" needs 5 more partitions you would issue the following command to create a new topic with 13 partitions, each with a single replica.

```
$ ./kafka-topics.sh \
    --zookeeper zk1.xyu.io:2181,zk2.xyu.io:2181/kafka \
    --create --topic xyu_gap_filler \
    --replica-assignment 1,1,1,1,1,1,1,1,2,2,2,2,2
```

Once this filler topic has been created the number of partitions per disk on each broker will be evened out.

Forcing a reassignment of every partition and replica for a topic is a bit harder to accomplish. First you must make sure that the current number of replicas and number of future replicas is less then the total number of brokers. (This is necessary because all brokers that you are assigning a partition to must not already be assigned that partition.) If this is not done then only some portion of partitions for a topic will actually be moved and it's likely one of the partitions being moved will end up getting moving to the same disk as a partition not being moved leading to a unbalanced distribution once again.

With dozens of partitions and thousands of possible permutations trying to assign all partitions so that they are both balanced across all brokers and are not assigned to a broker that already contains a replica of it was not something I wanted to do by hand so I wrote a Python script to generate the necessary assignments for me.

```
#!/usr/bin/env python

import argparse
```

```
import json
import random
import sys

def get_topic( args, data ):
    topic = data['partitions'][0]['topic']
    for partition in data['partitions']:
        if topic != partition['topic']:
            raise ValueError( 'Rebalance only works on one topic at a time' )
    return topic

def get_brokers( args, data ):
    brokers = set()
    if args.brokers:
        for broker in args.brokers.split( ',' ):
            brokers.add( int( broker ) )
    else:
        for partition in data['partitions']:
            for broker in partition['replicas']:
                brokers.add( broker )
    return brokers

def get_target_replicas( args, data ):
    if args.replicas:
        replicas = args.replicas
    else:
        replicas = get_current_replicas( args, data )
    return replicas

def get_current_replicas( args, data ):
    num_replicas = set()
    for partition in data['partitions']:
        num_replicas.add( len( partition['replicas'] ) )
    return max( num_replicas )

def get_broker_list( data, replicas, brokers ):
    broker_list = []
    per_broker = len( data['partitions'] ) * replicas / len( brokers )
    for broker in brokers:
        broker_list += [ broker ] * per_broker
    return broker_list

def rebalance( data, replicas, broker_list, verbose ):
    random.shuffle( broker_list )
    new_partitions = []
    for partition in data['partitions']:
        i = 0
        new_partition = partition.copy()
        new_partition['replicas'] = []
        while True:
            # Ran out of brokers that can be assigned
            if len( broker_list ) == i:
                return None

            # Don't use if matches any existing broker
            if broker_list[i] in partition['replicas']:
                i += 1
                continue

            # Don't use if matches already selected new broker
            if broker_list[i] in new_partition['replicas']:
                i += 1
                continue

            new_partition['replicas'] += [ broker_list[i] ]
```

```
            broker_list = broker_list[0:i] + broker_list[1+i:]
            i=0

            # Found all brokers for this partition
            if len( new_partition['replicas'] ) == replicas:
                break

        if verbose:
            print "Partition " + str( partition['partition'] ) + "\t" + str( partition['replicas'] ) + " -> " + str( new_partition['replicas'] )

        new_partitions += [ new_partition ]

    return new_partitions

# CLI args
parser = argparse.ArgumentParser()
parser.add_argument(
    "assignment", type=str,
    help="current partition replica assignment (JSON)"
)
parser.add_argument(
    "--brokers", type=str,
    help="override brokers to use"
)
parser.add_argument(
    "--replicas", type=int,
    help="override number of replicas after rebalance"
)
parser.add_argument(
    "-v", "--verbose",
    help="increase output verbosity", action="store_true"
)
args = parser.parse_args()
data = json.loads( args.assignment )

# Check we have enough brokers to handle rebalance
if get_target_replicas( args, data ) + get_current_replicas( args, data ) > len( get_brokers( args, data ) ):
    raise ValueError( 'Cannot rebalance, not enough brokers' )

topic = get_topic( args, data )
replicas = get_target_replicas( args, data )
brokers = get_brokers( args, data )
broker_list = get_broker_list( data, replicas, brokers )

# Try to assign 100 times then giveup
print "Attempting to rebalance " + topic
for i in range( 0, 1800 ):
    if args.verbose:
        print "Try " + str( i )
    else:
        sys.stdout.write('.')
        if 74 == i%75:
            print ""
    new_partitions = rebalance( data, replicas, broker_list, args.verbose )
    if None != new_partitions:
        break

if None == new_partitions:
    raise RuntimeError( 'Could not rebalance partitions' )

print "\nDone!\n"

print json.dumps(
    { "version" : 1, "partitions" : new_partitions },
    separators=( ',', ':' )
```

```
134 )
```

kafka-rebalance.py hosted with ❤ by GitHub view raw

To use this script you must first get the current partition replica assignment from Kafka for the topic that you are trying to move using the `kafka-reassign-partitions.sh` tool:

```
$ ./kafka-reassign-partitions.sh \
    --zookeeper zk1.xyu.io:2181,zk2.xyu.io:2181/kafka \
    --broker-list '1,2,3' \
    --topics-to-move-json-file topic.json \
    --generate
```

The param specified by `broker-list` does not actually matter as we don't care about the plan generated by the tool. We're just using this to print out the current assignment in JSON format.

The `topic.json` file should look something like the following. (Remember, only move one topic at a time otherwise balance is not guaranteed!)

```
{
  "version": 1,
  "topics": [
    { "topic": "my_sample_topic" }
  ]
}
```

The `kafka-reassign-partitions.sh` script will generate two JSON strings, one showing the current partition assignment and one showing a proposed partition assignment. Ignore the proposed assignment and instead pass the current assignment JSON string to `kafka-rebalance.py` to get a new proposed assignment, one that guarantees all partitions will relocate.

```
$ ./kafka-rebalance.py \
    '{"version":1,"partitions":[{…},{…}]}'
```

This is a pretty convoluted way to rebalance topics across all disks on each Kafka broker however it's pretty safe as this method does not involve any metadata hacking nor does it necessitate bring any brokers down. Hopefully Kafka will become better at balancing topics across disks when run in JBOD mode by itself in the future.

SHARE:

RELATED

Building a Faster ETL Pipeline with Flume, Kafka, and Hive
2015-07-13
In "Big Data"

Log Analysis With Hive
2016-01-04
In "Big Data"

Elasticsearch StatsD Plugin
2014-08-11
In "Big Data"

2016-02-29 xyu Big Data Apache, Kafka, Opensource