Summary

Requirements:

In the project, a user would like to have a script to do shards assignment in a balanced way. The rule of thumb would be to allocate these shards given their size among nodes in a balanced way. Also replicas should also be distributed among the cluster too.

In summary, the allocation should meet the following criteria:

- The shards will be distributed among the cluster with the optimal minimum difference between each node
- When there is no shard needs to be assigned, the program will stop and exit
- When there is no node that has any available space, the program will stop and exit
- When there is a node that is full, it should be removed from the node list and will not be considered for the following allocation
- When there is no node that has any available space, the program will stop and exit
- The number of replica should not larger than nodes.size()-1
- Any replica of the shard should not be in the same node with the shard
- Any two replicas of the same shard should not be assigned to the same node

Implementation:

To get the optimal solution for this problem, I solved it with binary search. Two assumptions are made:

1. No node will hold way too many shards than others (ideally they will hold shards evenly in terms of numbers per node too)
2. If the first node gets its closest shard locally, then the shard that is closest to the node followed by it would also be optimal.

The entire workflow is as follows:

1. Rule out node that doesn't have enough space and shard whose size is larger than any available space among nodes
2. Calculate the estimated usage of the cluster if all shards are allocated
3. Sort the nodes in descending order and then iterate over them
4. For each node, find the shard that is closest to its estimated usage by using binary search
5. Repeat the previous step, until no more shard can be allocated or all the nodes run out of its usage

Distribute replica:

1. Check if args.replica is larger than the number of nodes-1, if so, raise ValueError
2. Iterate over shards and then iterate over each replica of the same shard
3. Sort nodes except the node which holds the current shard in descending order while sort replica based on their size in ascending order (the greedy strategy, the node which has a larger available space will take larger shard)
4. Use Mod to get the index of the node where the replica will be put

Pros and Cons

Pros:

- The time complexity of binary search is O(logm), if there are n nodes,m shards, the time complexity is O(nlogm) which is acceptable
- Although it is not the way to find the best solution, given m is often way larger than n, any solution (backtracking etc) that tries to find the best split of the shard set and then find the solution based on it will use more time than it. So it is still acceptable

Cons:

- It is made on some assumptions
- It can return optimal solution but not the best solution
- The distribution of replica doesn't follow the same rule and it is not the optimal solution

Other Solutions

We can also define weight and balance nodes according to weight on each node. This is also the solution that elasticsearch is using. Based on the code here, It can be done in steps:

- Allocate shards to its optimal one
- rebalance

Allocate:

- Define weight:
    - sum = shard_factor + disk_factor
    - weight_shard = node.num_shards - avg_num_shards; theta0 = shard_factor/sum
    - weight_disk = node.used_space - avg_used_space; theta1=disk_factor /sum
    - Weight = weight_shard*theta0+weight_disk*theta1
- Use the same strategy to find the optical node for every unassigned shard

Rebalance

- For every collection, calculate the weight for every node and then find the maximum delta of weight (weight[hi] - weight[lo])

- If the maximum delta is larger than threshold, do rebalance for this collection, move shard from highest load to lowest load and calculate weight again, stop rebalance until the maximum delta is smaller than threshold

Pros and Cons

Pros

- It can return better results after rebalance and it takes the number of shards and the usage of disk into account the same time
- The assumption that is made previously is not always true, and it can return better results on the condition when shards are not splitted in the similar size
- Given the dish usage is often large number in bytes, the weight definition will make it easier to compute and lower the impact of different unit of measurements

Cons

- Setting the prior number for thread, the shard factor and the disk factor will have some impact on the final results.