Rebalancing vBucket Mapping

Weikang Zhou

Couchbase

July 6, 2012

Table of contents

Notations

- N: number of vbuckets
- M: number of nodes $\mathbb{Z}_M = \{0, 1, \ldots, M-1\}$
- L: number of copies for each vBucket (*including* the active one)
- S: slave number for each node
- For instance, $N = 1024$, $M = 41$, $L = 4$, $S = 10$.

Mapping and replication matrix

Mapping A: $N \times L$ matrix of node IDs.

Replication matrix R: $M \times M$ matrix. A replication pair is a 2-tuple in A: $\Big(A(i,0), A(i,j)\Big), j = 1, \ldots, L-1$. $R(k,l)$ is the total number of occurrences of the replication pair (k,l) in the mapping A.

Optimization Objectives

0. *First order balance* The mapping A should be "balanced" in first order, i.e. having equal number of occurrences of each node.
1. *Second order balance* The replication pairs in A are either "balanced" or zero. That is, the elements in matrices R are either equal numbers for each row, or zero.
2. *Slave number balance* The number of slaves is balanced. That is, the number of non-zero entries in matrices R for each row is a fixed number.

Structure of R

- R completely characterizes A as far as all the constraints are concerned.
- Column sum of R: the number of times an element appears in the replica region of A.
- Column sums: either $\lfloor \dfrac{N \cdot (L-1)}{M} \rfloor$ or $\lfloor \dfrac{N \cdot (L-1)}{M} \rfloor + 1$
- The row sum of R divided by $(L-1)$: the number of active nodes in A: $\lfloor N/M \rfloor$ or $\lfloor N/M \rfloor + 1$.
- For each row, R will have S nonzero equal terms that sums to $\lfloor N/M \rfloor \cdot (L-1)$ or $(\lfloor N/M \rfloor + 1) \cdot (L-1)$.
- Each entry of R: $\left\lfloor \lfloor \dfrac{N}{M} \rfloor \cdot \dfrac{L-1}{S} \right\rfloor$, or $\left\lfloor \lfloor \dfrac{N}{M} \rfloor \cdot \dfrac{L-1}{S} \right\rfloor + 1$, or $\left\lfloor (\lfloor \dfrac{N}{M} \rfloor + 1) \cdot \dfrac{L-1}{S} \right\rfloor$, or $\left\lfloor (\lfloor \dfrac{N}{M} \rfloor + 1) \cdot \dfrac{L-1}{S} \right\rfloor + 1$.

Structure of R

If R satisfies:

1. R has row sum $\lfloor N/M \rfloor \cdot (L-1)$ or $(\lfloor N/M \rfloor + 1) \cdot (L-1)$;
2. Each entry in each row of R is balanced;
3. R has column sum $\lfloor \dfrac{N \cdot (L-1)}{M} \rfloor$ or $\lfloor \dfrac{N \cdot (L-1)}{M} \rfloor + 1$.

Then A satisfies all three constraints.

Construction of new A

1. First we determine the least frequent occurrence of the desired value to fill in each row, and disperse them as evenly as possible along the row.
2. We then use different offsets to avoid patterns over different rows. This will result in a R that has very close column sum. `[0R_disperse]`
3. Next we greedily balance R by search for possible exchange of two entries in the same row. `[1R_greedy]`
4. If R is still not balanced, we propagate extreme values toward its opposites. `[2R_balanced]`
5. Filling in A vertically in replica region for each active node.

Rebalance

- We need only to consider two simple cases where there has been no failover and we simply want to either add or delete nodes.
- If we need to simultaneously add and delete nodes, we match them first by letting the nodes to be deleted to be filled by the newly added nodes. What remains must be a simple addition or deletion.
- If rebalance occurs after failover, we consider the input as original balanced mapping before failover. The result is the same because failed nodes are removed.

Adding nodes

- A lower bounds on the movements: $\dfrac{N \cdot L}{M}(M - M_0)$.
- *Free* movements: if they have already been counted in the lower bound.
- We want to minimize the additional movements provided that the resulting mapping is balanced.
- First get a target replication matrix R_T for M. Our goal is to adjust A with minimal movements so that R becomes R_T. [3R_target, 4R_original]
- The algorithm uses three stages to achieve this goal:
  1. Make row sums of R equal to the row sums of target R_T; [5R_SmartARowMove]
  2. Make substitute to correct the slave nodes; [6R_substitute]
  3. Locally adjust A so that R is exactly the target R_T. [7R_TwoStageFinish]
- The example: $M_0 = 22, M = 27$, move = 1200, lower bound = 987

1: Get R for the current A. Generate a balanced mapping with (N, M, L, S) and retrieve its R to be our target R_T.
2: Get rid of rows in R with $R^R(i) > R_T^R(i)$.
3: **for** $i = 0$ to $M_0 - 1$ **do**
4:     Search for a best row of A starting with i and a best new node $M_0 \leq j < M$: change i to j until $R^R(i) \geq R_T^R(i)$
5: **end for**
6: **for** $i = 0$ to $M - 1$ **do**
7:     Let the slaves of i to be nodes $i + 1, \ldots, i + S \bmod M$
8:     If in R i used to have different slaves: substitute
9: **end for**
10: **for** $i = 0$ to $M - 1$ **do**
11:     Until we get R_T, search for k and j, where A has too many (i, k) pairs and too few (i, j) pairs and change k to j, use a second row if needed.
12: **end for**

Adding/Deleting nodes

1. Bottoleneck: the additional movements primarily result from the fact that balanced pairs among the old nodes are assigned to the new nodes in active positions, and as a consequence the replicas have to be changed individually.
2. Deleting nodes is similar, with different implementation details, such as renumbering of nodes.

Testing results

- For $N = 1024$, running time on average is 0.24 s.
- When evaluating performance, we consider the gap between actual *movements* it takes for rebalance with its theoretical *lower bound*.
- We want to minimize the ratio between the two and their absolute difference.
- $N = 1024, L = 2$, average absolute difference 59.86, average ratio 1.241
- $N = 1024, L = 4$, average absolute difference 155.1, average ratio 1.219

Testing results

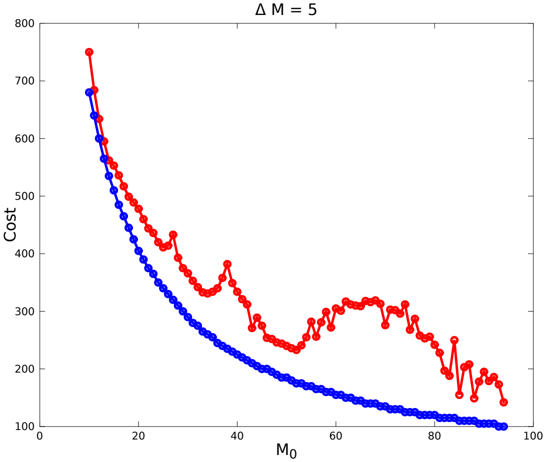


Figure: Gap between movements and lower bound. $\Delta M = 5, L = 2$.

Testing results

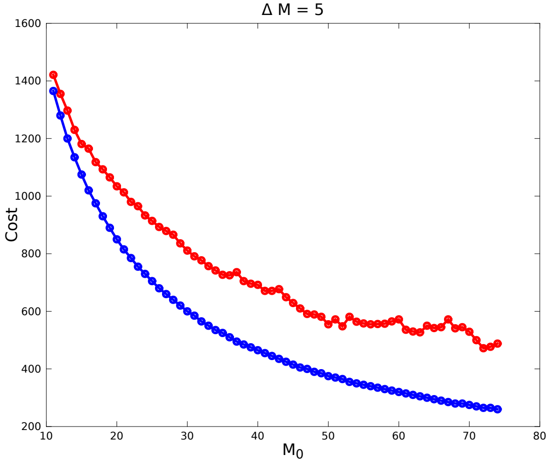
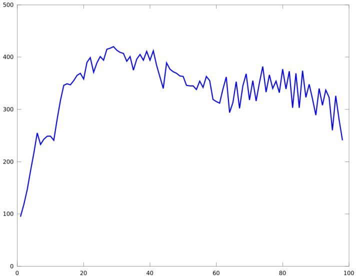
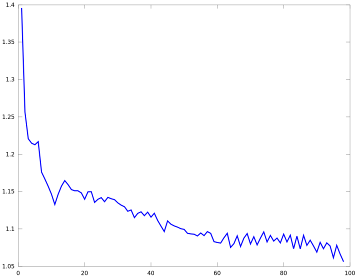


Figure: Gap between movements and lower bound. $\Delta M = 5, L = 4$.

Testing results



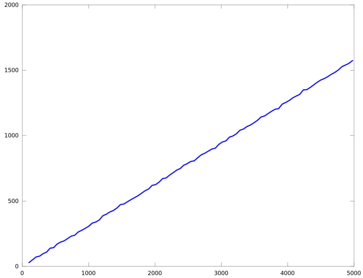
(a) Absolute difference



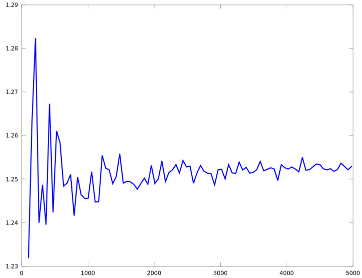
(b) Ratio

Figure: $M_0 = 20$ fixed, $\Delta M = 1$ changes from 1 to 98

Testing results



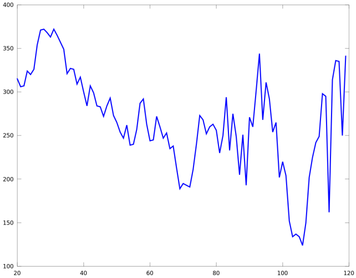
(a) Absolute difference



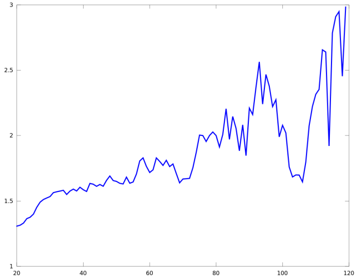
(b) Ratio

Figure: $M_0 = 15$, $\Delta M = 5$ fixed, N changes from 100 to 5000

Testing results



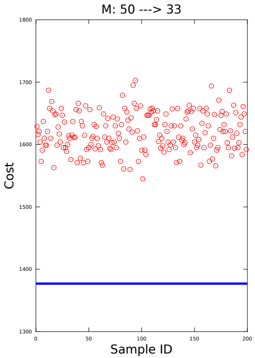
(a) Absolute difference



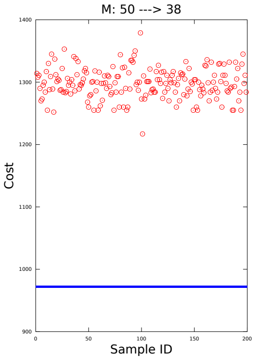
(b) Ratio

Figure: $\Delta M = -4$ fixed, M_0 changes from 20 to 119

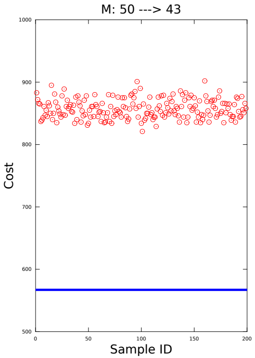
Testing results



(a) $\Delta M = -17$



(b) $\Delta M = -12$



(c) $\Delta M = -7$

Figure: $\Delta M = -17, -12, -7$, $M_0 = 50$, deleted nodes randomly generated

Testing results

1. Deleting movements is roughly the same with adding movements with the same (reversed) parameter.
2. Ratio is the worst when number of node is large but change is small. In this case the absolute difference is roughly the same but the lower bound decreases quickly.
3. The absolute difference grows to be a constant as M fixed but ΔM increases.
4. As N grows, the ratio stays roughly the same and cost grows linearly.
5. When delete nodes: the choice of nodes to delete will have a moderate impact on movements. (It helps when the nodes to be deleted are contiguous and it hurts when they are dispersed.)

Next steps

1. Global search of optimal: reduce the optimization gap.
2. Theoretical considerations: refine the lower bound.
3. Consider other constraints: nodes on shelves, nodes with different versions, heterogeneous nodes...