Benchmarking between string method and hash-list method

All benchmarking uses n\*5 table as example. Exact attributes are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| student | name | age | gender | id |

Implementation:

Initialization:

Implemented as not piped I/O, which means I/O frequency equals to the number of insertion. This is implemented in such way to observe better difference since the result is quite undistinguishable under pipe I/O.

Search:

Search is implemented as I/O search, which is retrieving entry once a time and compare and repeat for n times. This prevent memory limit happened in loading all data to front and it is the only method that can be implemented in the same way in both schemas. This is, in general, a valid way to do this task as Redis encourages users to prefer I/O operation ober memory operation.

Sort:

Not implemented. I do not really know how to implement sort in string schema.

Memory allocation:

This value is retrieved from Redis built-in memory command.

Benchmarking for 10000:

(difference is colored to green to show String schema win and red to show hash-list win)

|  |  |  |  |
| --- | --- | --- | --- |
| Size=10000,t=ms | String schema | Hash-list schema | Difference |
| Initialize | 844.1108199357986 | 1116.585629940033 | +132% |
| Search | 1.4148859977722168 | 1.2410629987716675 |  |
| Memory allocation | 4.970944MB | 2.51272MB | -50% |

Benchmarking for 25000:

|  |  |  |  |
| --- | --- | --- | --- |
| Size=25000,t=ms | String schema | Hash-list schema | Difference |
| Initialize | 1940.4012439250946 | 2282.6092879772186 | +117% |
| Search | 1.713973045349121 | 1.6048020124435425 |  |
| Memory allocation | 11.184544MB | 4.948368MB | -44% |

Benchmarking for 50000:

|  |  |  |  |
| --- | --- | --- | --- |
| Size=50000,t=ms | String schema | Hash-list schema | Difference |
| Initialize | 3779.8733439445496 | 6289.282114028931 | +166% |
| Search | 1.5746580362319946 | 2.1750099658966064 |  |
| Memory allocation | 21.558464MB | 9.411328MB | -43% |

Benchmarking for 100000:

|  |  |  |  |
| --- | --- | --- | --- |
| Size=100000,t=ms | String schema | Hash-list schema | Difference |
| Initialize | 6941.113039970398 | 9562.68447291851 | +137% |
| Search | 2.4059540033340454 | 1.4262419939041138 |  |
| Memory allocation | 42.256944MB | 18.333088MB | -43% |

Conclusion:

Search operation time stamp looks to be really weird. Operation time stamp will be looked into with more optimization and implementation.

Hash-list method takes in general about 30%~50% more time to initialize. However, it saves about 40%~50% memory space comparing to string method. It is really hard to tell which is better at this point as both of them showing their unique advantage at this point.

Update:

Search operation is now correctly benchmarked. It looks like to have similar performance but with little variation depending on the data complexity.

Benchmarking for 10000:

(difference is colored to green to show String schema win and red to show hash-list win)

|  |  |  |  |
| --- | --- | --- | --- |
| Size=10000,t=ms | String schema | Hash-list schema | Difference |
| Initialize | 844.1108199357986 | 1116.585629940033 | +132% |
| Memory allocation | 4.970944MB | 2.51272MB | -50% |

Benchmarking for 25000:

|  |  |  |  |
| --- | --- | --- | --- |
| Size=25000,t=ms | String schema | Hash-list schema | Difference |
| Initialize | 1940.4012439250946 | 2282.6092879772186 | +117% |
| Memory allocation | 11.184544MB | 4.948368MB | -44% |

Benchmarking for 50000:

|  |  |  |  |
| --- | --- | --- | --- |
| Size=50000,t=ms | String schema | Hash-list schema | Difference |
| Initialize | 3779.8733439445496 | 6289.282114028931 | +166% |
| Memory allocation | 21.558464MB | 9.411328MB | -43% |

Benchmarking for 100000:

|  |  |  |  |
| --- | --- | --- | --- |
| Size=100000,t=ms | String schema | Hash-list schema | Difference |
| Initialize | 6941.113039970398 | 9562.68447291851 | +137% |
| Memory allocation | 42.256944MB | 18.333088MB | -43% |