Item 159
git_comments:

git_commits:

1. **summary:** Replace Hashmap to Object2IntOpenHashMap in OnHeapStringDictionary (#4568)
   message: Replace Hashmap to Object2IntOpenHashMap in OnHeapStringDictionary (#4568) * Replace Hashmap to Object2IntOpenHashMap in OnHeapStringDictionary * Use getInt instead of get method * Adjust measurment

github_issues:

github_issues_comments:

github_pulls:

1. **title:** Replace Hashmap to Object2IntOpenHashMap in OnHeapStringDictionary
   body: This PR replaces `Hashmap` to `Object2IntOpenHashMap` in OnHeapStringDictionary. https://issues.apache.org/jira/browse/PINOT-8

github_pulls_comments:

1. Can you please benchmark this map vs. HashMap? If I remember correctly, we benchmarked it and it is actually performing worse than the HashMap
2. Here are the results by using offheap dictionary, onheap dictionary with HashMap, and onheap dictionary with Object2IntOpenHashMap: offheap: ``` Total time: 12389 Total time for 10000000 lookups: 15996ms DictSize,TimeTaken(ms),SegmentSize,NumLookups,Min,Max,Mean,StdDev,Median,Skewness,Kurtosis,Variance,BufferSize 1000000,15996,514501118,10000000,1000.0,1000.0,1000.0,0.0,1000.0,NaN,NaN,0.0 Process finished with exit code 0 ``` onheap with HashMap: ``` Total time: 12250 Total time for 10000000 lookups: 237ms DictSize,TimeTaken(ms),SegmentSize,NumLookups,Min,Max,Mean,StdDev,Median,Skewness,Kurtosis,Variance,BufferSize 1000000,237,514501118,10000000,1000.0,1000.0,1000.0,0.0,1000.0,NaN,NaN,0.0 Process finished with exit code 0 ``` onheap with Object2IntOpenHashMap: ``` Total time: 12187 Total time for 10000000 lookups: 242ms DictSize,TimeTaken(ms),SegmentSize,NumLookups,Min,Max,Mean,StdDev,Median,Skewness,Kurtosis,Variance,BufferSize 1000000,242,514501118,10000000,1000.0,1000.0,1000.0,0.0,1000.0,NaN,NaN,0.0 Process finished with exit code 0 ``` As the results shows, there's no big difference on dictionary lookup between using these two hash maps.
3. @jackjlli Can you also benchmark the memory cost for these 2 maps? You can include the benchmark class in the pr.
4. # [Codecov](https://codecov.io/gh/apache/incubator-pinot/pull/4568?src=pr&el=h1) Report > :exclamation: No coverage uploaded for pull request base (`master@e98efcb`). [Click here to learn what that means](https://docs.codecov.io/docs/error-reference#section-missing-base-commit). > The diff coverage is `62.5%`. [![Impacted file tree graph](https://codecov.io/gh/apache/incubator-pinot/pull/4568/graphs/tree.svg?width=650&token=4ibza2ugkz&height=150&src=pr)](https://codecov.io/gh/apache/incubator-pinot/pull/4568?src=pr&el=tree) ```diff @@ Coverage Diff @@ ## master #4568 +/- ## == Coverage ? 64.44% Complexity ? 32 == Files ? 1072 Lines ? 55703 Branches ? 8130 == Hits ? 35896 Misses ? 17160 Partials ? 2647 ``` | [Impacted Files](https://codecov.io/gh/apache/incubator-pinot/pull/4568?src=pr&el=tree) | Coverage Δ | Complexity Δ | | |---|---|---|---| | [.../segment/index/readers/OnHeapStringDictionary.java](https://codecov.io/gh/apache/incubator-pinot/pull/4568/diff?src=pr&el=tree#diff-cGlub3QtY29yZS9zcmMvbWFpbi9qYXZhL29yZy9hcGFjaGUvcGlub3QvY29yZS9zZWdtZW50L2luZGV4L3JlYWRlcnMvT25IZWFwU3RyaW5nRGljdGlvbmFyeS5qYXZh) | `60.71% <62.5%> (ø)` | `0 <0> (?)` | | ------ [Continue to review full report at Codecov](https://codecov.io/gh/apache/incubator-pinot/pull/4568?src=pr&el=continue). > **Legend** - [Click here to learn more](https://docs.codecov.io/docs/codecov-delta) > `Δ = absolute <relative> (impact)`, `ø = not affected`, `? = missing data` > Powered by [Codecov](https://codecov.io/gh/apache/incubator-pinot/pull/4568?src=pr&el=footer). Last update [e98efcb...246c829] (https://codecov.io/gh/apache/incubator-pinot/pull/4568?src=pr&el=lastupdated). Read the [comment docs](https://docs.codecov.io/docs/pull-request-comments).
5. I've updated the test and here are the results comparing `Object2IntOpenHashMap` with `HashMap`. `Object2IntOpenHashMap`: ``` Total time for building segment: 22239 Total time for 10000000 lookups: 1876ms Memory usage: 2168980704 DictSize,TimeTaken(ms),SegmentSize,NumLookups,Min,Max,Mean,StdDev,Median,Skewness,Kurtosis,Variance,BufferSize 2000000,1876,1029251120,10000000,1000.0,1000.0,1000.0,0.0,1000.0,NaN,NaN,0.0 ``` ``` Total time for building segment: 22667 Total time for 10000000 lookups: 1901ms Memory usage: 2168993176 DictSize,TimeTaken(ms),SegmentSize,NumLookups,Min,Max,Mean,StdDev,Median,Skewness,Kurtosis,Variance,BufferSize 2000000,1901,1029251120,10000000,1000.0,1000.0,1000.0,0.0,1000.0,NaN,NaN,0.0 ``` ``` Total time for building segment: 22174 Total time for 10000000 lookups: 1880ms Memory usage: 2168980824 DictSize,TimeTaken(ms),SegmentSize,NumLookups,Min,Max,Mean,StdDev,Median,Skewness,Kurtosis,Variance,BufferSize 2000000,1880,1029251120,10000000,1000.0,1000.0,1000.0,0.0,1000.0,NaN,NaN,0.0 ``` `HashMap`: ``` Total time for building segment: 22343 Total time for 10000000 lookups: 2294ms Memory usage: 2248214504 DictSize,TimeTaken(ms),SegmentSize,NumLookups,Min,Max,Mean,StdDev,Median,Skewness,Kurtosis,Variance,BufferSize 2000000,2294,1029251120,10000000,1000.0,1000.0,1000.0,0.0,1000.0,NaN,NaN,0.0 ``` ``` Total time for building segment: 24396 Total time for 10000000 lookups: 2244ms Memory usage: 2248201544 DictSize,TimeTaken(ms),SegmentSize,NumLookups,Min,Max,Mean,StdDev,Median,Skewness,Kurtosis,Variance,BufferSize 2000000,2244,1029251120,10000000,1000.0,1000.0,1000.0,0.0,1000.0,NaN,NaN,0.0 ``` ``` Total time for building segment: 23854 Total time for 10000000 lookups: 2216ms Memory usage: 2248211648 DictSize,TimeTaken(ms),SegmentSize,NumLookups,Min,Max,Mean,StdDev,Median,Skewness,Kurtosis,Variance,BufferSize 2000000,2216,1029251120,10000000,1000.0,1000.0,1000.0,0.0,1000.0,NaN,NaN,0.0 ``` As shown above, `Object2IntOpenHashMap` uses less memory and it takes less time on lookups. For 2m rows of strings, reduced memory: ``` 2,248M - 2,168M = 80MB ``` reduced time for 10m lookups: ``` 2200ms - 1900ms = 300ms ```

github_pulls_reviews:

jira_issues:

1. **summary:** Memory waste and GC slowdown in Pinot-server due to many j.l.Integer instances
   description: I've recently analyzed a heap dump of Pinot-server with jxray ([www.jxray.com).|http://www.jxray.com%29./] One problem that I found is that 5.8% of used heap (by byte count) and ~18% (by object count) is wasted by {{java.lang.Integer}} instances. Thus they negatively impact both memory footprint of Pinot-server and most likely its GC time as well. The GC impact is due to the fact that these objects are very numerous (nearly 1/5 of all objects), and the time to collect objects that get promoted into the Old Gen is proportional to the number of these objects. The screenshot below shows where these objects are coming: the live ones are managed by {{OnHeapStringDictionary._unPaddedStringToIdMap}}, and the garbage ones likely originate in that table as well. !Screen Shot 2019-08-28 at 13.01.53.png! To fix this problem, it's enough to replace the {{java.util.HashMap}} above with the specialized {{fastutil Object2IntOpenHashMap.}}

jira_issues_comments: