Effect of Unused Columns on Query Performance

Max Ganz II @ Redshift Observatory

9th August 2021

Abstract

Redshift is column-store and as such in principle queries are unaffected by the unused columns in the tables being queried. In practise, accessing any single column is unaffected by the number of columns, but, tentatively, it looks like accessing multiple columns shows that the more columns are present between a column and the final column in the table, the slower it is to access the column, and so the more columns are present in a table, the slower access becomes for all columns. The slowdown, even when there are 1600 columns, is very small for `dc2.large` and `ra3.xlplus`, but is much larger for `ds2.xlarge`. However, for normally sized tables, up to say 100 columns, even on `ds2.xlarge` the slowdown is very small.

Contents

Introduction

Redshift is a column-store database and as such tables are broken up into their constituent columns and each column is stored separately on disk.

A query typically will use only some rather than all of the columns in a table, and naturally the more columns are used, the slower the query, since more data has to be read from disk and more work has to be done to perform materialization, which is the task of joining the separated columns back up into contiguous rows for return to the client.

However, there is a question as to whether or not the columns which are not used by a query have a performance impact on the query. If this is so, then the more columns a table has, even though they are not used by a query, the more harmful for performance.

In principle those extra columns, being unused, should have no impact on query performance. The question is whether or not this is true in practise.

Test Method

The basic method is that we create a test table, and issue a test query on the table, and measure the time taken for the query to execute.

There are four test queries.

Each test query is issued five times, with the slowest and fastest results being discarded.

The test table is dropped, created and repopulated for every iteration, and for every test query; which is to say, all the test tables are identical, but each table only ever has one query issued on it.

The test table is varied, by the number of columns (2, 800, and 1600) and the number of blocks per column, per slice (1 and 8), with the five iterations of the four queries repeated on every variant of the table.

The table has `key` distribution. The `even` has off-by-one bugs when it comes to distribution rows, so cannot be used, as you end up with an uneven number of blocks on the different slices.

The value stored in the distribution key column, and the number of rows, are chosen to ensure each slice has an equal number of exactly completely full blocks. Validation checks are present in the test code to ensure each slice does in fact has the correct number of blocks with the correct number of rows.

For all columns the data type is `bigint` and `not null` is specified, encoding is `raw`, and the table is fully vacuumed and analyzed. Result caching is disabled, and the analysis threshold set to 0.

The tests described above are performed on three clusters, all two node, a `dc2.large`, a `ds2.xlarge` and an `ra3.xlplus`.

How long a query took to execute is taken directly from the system tables, and so mark the times internal to Redshift as to when the query execution started and ended (queuing and compile times are excluded).

The four test queries are;

1. find the `sum()` of the first column
2. find the `sum()` of the last column
3. find the `sum()` of the first column, when the second column, with one subtracted from its value, is greater than -10

4. find the `sum()` of the first column, when the last column, with one subtracted from its value, is greater than -10

The purpose of the “subtract one” is to defeat use of the Zone Map, and the comparison, greater than -10, ensures all rows are used in the `sum()`. This is because every column in a row has the same value, and that value is always 0 or greater.

These tests then measure how long it takes to `sum()` the first column, and the last column, and to access pairs of columns (first and second, first and last), as we vary the number of columns, the amount of data, and the node type.

Results

See Appendix A for the Python `pprint` dump of the results dictionary.

The X-axis is the number of exactly full blocks per column, per slice, in the table.

The Y-axis is the number of columns in the table.

The individual results are "mean/standard deviation" of the duration of the test queries in seconds.

As ever, five trials occurred, with the slowest and fastest being discarded.

Test duration was 14,375 seconds.

Performance Test

dc2.large, 2 nodes, first column

	1	8
2	0.007/0.000	0.013/0.000
800	0.008/0.000	0.012/0.000
1600	0.008/0.000	0.013/0.000

dc2.large, 2 nodes, last column

	1	8
2	0.011/0.003	0.012/0.000
800	0.008/0.000	0.013/0.000
1600	0.008/0.001	0.013/0.000

dc2.large, 2 nodes, first and second column

	1	8
2	0.012/0.003	0.018/0.002
800	0.009/0.000	0.025/0.000

	1	8
1600	0.009/0.001	0.026/0.001

dc2.large, 2 nodes, first and last column

	1	8
2	0.009/0.001	0.019/0.001
800	0.009/0.000	0.017/0.000
1600	0.008/0.000	0.018/0.002

ds2.xlarge, 2 nodes, first column

	1	8
2	0.006/0.000	0.010/0.001
800	0.006/0.000	0.010/0.001
1600	0.007/0.000	0.012/0.001

ds2.xlarge, 2 nodes, last column

	1	8
2	0.006/0.000	0.011/0.002
800	0.007/0.000	0.010/0.000
1600	0.007/0.000	0.011/0.001

ds2.xlarge, 2 nodes, first and second column

	1	8
2	0.006/0.000	0.014/0.001
800	0.007/0.000	0.091/0.002
1600	0.007/0.000	0.145/0.014

ds2.xlarge, 2 nodes, first and last column

	1	8
2	0.007/0.000	0.015/0.001
800	0.008/0.000	0.015/0.000
1600	0.008/0.001	0.014/0.001

ra3.xlplus, 2 nodes, first column

	1	8
2	0.008/0.002	0.009/0.001
800	0.006/0.000	0.008/0.001
1600	0.006/0.000	0.009/0.001

ra3.xlplus, 2 nodes, last column

	1	8
2	0.005/0.000	0.008/0.001
800	0.006/0.000	0.009/0.001
1600	0.006/0.000	0.008/0.001

ra3.xlplus, 2 nodes, first and second column

	1	8
2	0.006/0.000	0.011/0.002
800	0.006/0.001	0.019/0.000
1600	0.007/0.000	0.019/0.000

ra3.xlplus, 2 nodes, first and last column

	1	8
2	0.006/0.001	0.011/0.001
800	0.006/0.000	0.012/0.001
1600	0.007/0.000	0.011/0.001

Discussion

The first point of note is that for all tests with one block per column per slice, there are no significant differences in performance, for any of the SQL test queries, no matter how many columns there are, no matter which node type is used.

However, when we move to the eight block tests, differences do emerge.

The query times are of course longer, because eight blocks are being read per column per slice. What matters is not how long queries take as such, but differences in how long the queries take as the number of columns (and node types) vary.

Looking first at `dc2.large`, we see that when accessing the first column on its own, or the last column on its own, the number of columns makes no difference.

However, we have next the two test queries which access a pair of columns. As with increasing the number of blocks, these queries will be slower - they are reading two columns not one - but as before, we are only looking at the difference in performance as we vary the number of columns.

What we find is that when accessing the first and second column, moving from two columns to 800 or 1600 results in a slowdown, from 0.018 seconds to 0.025 and 0.026 seconds, respectively. This is very slight, but it seems to be a real change.

However, when looking at accessing the first and last column, we again see no significant change in performance as the number of columns change.

Very tentatively then, since we have so little evidence so far, it looks like the last columns in a table are accessed most quickly, and the first most slowly.

Moving on to `ds2.xlarge`, as before, accessing the first or last column makes no difference, and accessing first and last shows no significant change.

Now, as before, accessing first and second shows a slowdown, but the slowdown is far more pronounced with this node type. Query time goes from 0.014, to 0.091, to 0.145 seconds. That's a lot.

Finally, moving to `ra3.xlplus`, we see the same pattern, but with a change in performance for first and second on much the same order as for `dc2.large`.

Conclusions

When a query accesses a single column only, the number of columns in a table has no effect on performance.

When a query accesses the first and last columns, we also see the number of columns in a table has no effect on performance.

However, when a query accesses the first and second columns, we see the number of columns in a table *does* have an effect on performance. As the number of columns increases, the query becomes slower.

This slowdown effect is very small on `dc2.large` and `ra3.xlplus`, with the query duration going from about 0.011 second for 2 columns, to about 0.019 seconds for 1600 columns, but much larger on `ds2.xlarge`, which goes from 0.014 seconds to 0.145 seconds.

Tentatively, we can think to generalize this to say that the more columns there are between an accessed column and the final column in the table, the slower the column is to access, and so the more columns are in the table, the slower accessing a column becomes, because there are increasingly more columns between the column and the final column in the table.

In other words, the closer a column is to the end of the table, the faster it is to access, and the closer a column is to the start of the table, the slower.

The difference in performance is very small on `dc2.large` and `ra3.xlplus`, even with the worst case of 1600 columns, but looks considerable on `ds2.xlarge`, although having said that, it's a pretty rare table which exceeds even 30 columns, and for tables of this size, the slowdown even on `ds2.xlarge` is going to be very small.

Unexpected Findings

When you investigate Redshift, there are *always* unexpected findings.

1. The distribution style `even` appears to have off-by-one errors. When populating four slices with one block each, where each block has 130,994 rows (the maximum number of `bigint` with raw encoding per block), `even` will not give you one block per slice, but two slices with one block which is short one record, and the other two slices both have two blocks, one completely full and the next with one record.

   It won't matter in normal use, but it seems indicative of a lack of testing or testing but a lack of care to fix such errors, and it was a problem for me with this testing work, where I need to exactly control blocks per slice. It took some hours to implement my own solution using `key` based distribution, and this solution has unavoidable drawbacks compared to using `even`.

2. Not quite an unexpected finding, since I knew this already from earlier work, but I'm listing it here as I had to work around this issue and I would have discovered it here if I'd not already known; the `slice_num()` function appears very rarely to give the wrong slice number.

   I want to use `slice_num()` because where I'm using `key`, I need to know which slice a given distribution key value will go to, so I can deliberately put rows on each slice.

   Where `slice_num()` is unreliable, what has to be done instead is write a single row, then check `stv_blocklist` to find the column number, and then truncate the table, and then repeat until a value is found for each slice.

   So to work around one bug, I had to work around another bug.

   You can't help but feel at times Redshift is a bit of a clunker.

3. I realised something which actually came into being a little while ago; when the devs introduced automatic sortkey selection, and made this the default, it is not longer possible to keep unsorted tables.

   To be more exact, you can create unsorted tables, but because such tables are created by not specifying a sorting type, and now not specifying a sorting type has Redshift set the sorting type to `auto`, which gives Redshift

Figure 1: Redshift in the eyes of the dev team

Figure 2: The rest of us

carte blanche to change the sorting type, when you make an unsorted table, it can stop being an unsorted table at any time.

This breaks existing systems, because deliberately using unsorted tables is a perfectly valid design choice (with small tables and many slices, they save a *lot* of disk space), and it also breaks my test work, because I've been using unsorted tables as they're simpler to work with by not having an unsorted segment.

I have very strong feelings about this. The devs should *never* repeat ***never*** break existing systems - let alone doing so *silently* - and they should *never* get in the way of system developers and admin running their own cluster.

The problem is I suspect that the devs don't *know* when they're making breaking changes; this isn't the first time. I suspect all their work is based on fleet telemetry, and they're not actually in touch with users at all.

Further Questions

1. What happens with queries accessing three columns, ideally the first, second and third columns? do we see a further slowdown over the slowdown for the first and second columns, on the order we'd expect for the extra column, the third column, which should be the next slowest column to access.

Credits

This white paper was published on r/aws, leading to a review by one of the moderators, which identified two flaws in the test method. These were corrected, the test re-run, the white paper updated with the results, and then republished.

Revision History

v1

- Initial release.

v2

- Metadata changes. No content changes.

v3

After posting in `r/aws`, a subreddit mod reviewed the white paper and pointed out flaws in the test method. As a result, the following changes were made;

- Test method now for each test specifies the test table size in blocks per column per slice, rather than total number of rows. The blocks are exactly full.
- The test table is now dropped, created and populated between every test SQL query, to help defeat caching.
- The extra work create tables has made a full test run take many hours, so the set of numbers of columns to test has been reduced to 2, 800 and 1600.

v4

- No content changes; path to image file(s) changed.

v5

- Changed to Redshift Research Project (AWS have a copyright on "Amazon Redshift").

v6

- Added "About the Author". made site name in title a link, and made each chapter start a new page.

v7

- Web-site name changed to “Redshift Observatory”.
- Updated links from redshiftresearcproject.org to redshift-observatory.ch.

v8

- Removed “About The Author”.
- Added Slack join URL.

Appendix A : Raw Data Dump

Note these results are completely unprocessed; they are a raw dump of the results, so the original, wholly unprocessed data, is available.

```
{'proofs': {'dc2.large': {2: {}}, 'ds2.xlarge': {2: {}}, 'ra3.xlplus': {2: {}}}},
 'tests': {'dc2.large': {2: {1: {2: {'first': [0.009831000000000001,
                                                0.007121,
                                                0.007106,
                                                0.007927,
                                                0.006986],
                                      'first and last': [0.009111,
                                                         0.007817,
                                                         0.008855,
                                                         0.013065,
                                                         0.007816],
                                      'first and second': [0.009239,
                                                           0.021148,
                                                           0.015663,
                                                           0.010932,
                                                           0.008843],
                                      'last': [0.013651,
                                               0.006955,
                                               0.007586,
                                               0.05611,
                                               0.012056]},
                                  800: {'first': [0.007494,
                                                  0.007512,
                                                  0.007573,
                                                  0.008072,
                                                  0.00712],
                                        'first and last': [0.008437,
                                                           0.008456,
                                                           0.008418,
                                                           0.008853,
                                                           0.009487],
                                        'first and second': [0.00826,
                                                             0.008568,
                               0.008834,
                               0.008985,
                               0.019111],
          'last': [0.017022,
                   0.007516,
                   0.007351,
                   0.007447,
                   0.007659]},
    1600: {'first': [0.007455,
                     0.008847,
                     0.007901,
                     0.00799,
                     0.011692],
           'first and last': [0.009078,
                              0.008211,
                              0.00825,
                              0.008508,
                              0.007939],
           'first and second': [0.009466,
                                0.011271,
                                0.008583,
                                0.008835,
                                0.010071],
           'last': [0.007188,
                    0.007826,
                    0.009507,
                    0.010988,
                    0.007804]}},
8: {2: {'first': [0.013255,
                  0.012184,
                  0.01709,
                  0.012838,
                  0.012969],
        'first and last': [0.028075,
                           0.01886,
                           0.016114,
                           0.020439,
                           0.016953],
        'first and second': [0.02386,
                             0.020475,
                             0.017573,
                             0.016527,
                             0.016873],
        'last': [0.012078,
                 0.012822,
                 0.011611,
                 0.012327,
                 0.012086]},
    800: {'first': [0.012186,
                    0.012387,
                                      0.012328,
                                      0.013211,
                                      0.012549],
                            'first and last': [0.016758,
                                               0.017129,
                                               0.017125,
                                               0.018622,
                                               0.017386],
                            'first and second': [0.025634,
                                                 0.025044,
                                                 0.027201,
                                                 0.024404,
                                                 0.024922],
                            'last': [0.01231,
                                     0.015804,
                                     0.013119,
                                     0.012416,
                                     0.013035]},
                      1600: {'first': [0.012601,
                                       0.014621,
                                       0.012467,
                                       0.012254,
                                       0.012759],
                             'first and last': [0.020315,
                                                0.016714,
                                                0.016645,
                                                0.021115,
                                                0.016532],
                             'first and second': [0.035006,
                                                  0.024816,
                                                  0.026592,
                                                  0.026567,
                                                  0.02428],
                             'last': [0.013417,
                                      0.015566,
                                      0.013112,
                                      0.012518,
                                      0.012926]}}}},
'ds2.xlarge': {2: {1: {2: {'first': [0.010581,
                                     0.006096,
                                     0.005874,
                                     0.006064,
                                     0.005705],
                           'first and last': [0.007069,
                                              0.007114,
                                              0.00696,
                                              0.021457,
                                              0.006111],
                           'first and second': [0.006654999999999999,
                                                0.006148,
                             0.008511,
                             0.005904,
                             0.005982],
        'last': [0.005403,
                 0.005576,
                 0.00568,
                 0.005537,
                 0.005331]},
    800: {'first': [0.009515,
                    0.006787,
                    0.00573,
                    0.006183,
                    0.006404],
          'first and last': [0.007479,
                             0.006345,
                             0.007722,
                             0.007325,
                             0.009618],
          'first and second': [0.007499,
                               0.007612,
                               0.007331,
                               0.007663,
                               0.006303],
          'last': [0.006621,
                   0.006717,
                   0.008064,
                   0.006663,
                   0.006469]},
    1600: {'first': [0.006767,
                     0.008588,
                     0.006473,
                     0.006884,
                     0.007529],
           'first and last': [0.006403,
                              0.006559,
                              0.008803,
                              0.007665,
                              0.00828],
           'first and second': [0.006299,
                                0.007696,
                                0.007924,
                                0.008282,
                                0.006877],
           'last': [0.006611,
                    0.006213,
                    0.006902,
                    0.009258,
                    0.007416]}},
8: {2: {'first': [0.010172,
                  0.010276,
                  0.009098,
                  0.011165,
                  0.008997],
        'first and last': [0.014892,
                           0.012017,
                           0.012556,
                           0.01606,
                           0.019128],
        'first and second': [0.012179,
                             0.01791,
                             0.012364,
                             0.015408,
                             0.01507],
        'last': [0.014448,
                 0.009436,
                 0.010099,
                 0.009261,
                 0.013047]},
    800: {'first': [0.010728,
                    0.011137,
                    0.009314,
                    0.00939,
                    0.010986],
          'first and last': [0.015952,
                             0.012699,
                             0.016202,
                             0.01528,
                             0.015233],
          'first and second': [0.087814,
                               0.093193,
                               0.093329,
                               0.090595,
                               0.087897],
          'last': [0.01018,
                   0.009748,
                   0.009193,
                   0.009391,
                   0.009604]},
    1600: {'first': [0.011818,
                     0.015312,
                     0.012704,
                     0.011397,
                     0.011389],
           'first and last': [0.015404,
                              0.012765,
                              0.015671,
                              0.012296,
                              0.012886],
           'first and second': [0.092846,
                                0.158659,
                                                       0.157882,
                                                       0.126101,
                                                       0.150604],
                                  'last': [0.011752,
                                           0.009684,
                                           0.009571,
                                           0.011116,
                                           0.011377]}}}},
    'ra3.xlplus': {2: {1: {2: {'first': [0.009755,
                                         0.010415,
                                         0.005871,
                                         0.008958,
                                         0.005723],
                               'first and last': [0.009507,
                                                  0.00601,
                                                  0.006902,
                                                  0.005113,
                                                  0.005115],
                               'first and second': [0.00786500000000023,
                                                    0.00534,
                                                    0.005758,
                                                    0.005071,
                                                    0.005595],
                               'last': [0.005239,
                                        0.004884,
                                        0.005239,
                                        0.005269,
                                        0.006372]},
                           800: {'first': [0.005928,
                                           0.005915,
                                           0.005506,
                                           0.005278,
                                           0.006179],
                                 'first and last': [0.006394,
                                                    0.006603,
                                                    0.00621,
                                                    0.006334,
                                                    0.007127],
                                 'first and second': [0.005769,
                                                      0.005586,
                                                      0.006847,
                                                      0.008837,
                                                      0.005641],
                                 'last': [0.004933,
                                          0.005238,
                                          0.007724,
                                          0.005784,
                                          0.005889]},
                           1600: {'first': [0.006202,
                                            0.005788,
                         0.005208,
                         0.005904,
                         0.006407],
               'first and last': [0.005717,
                                  0.006912,
                                  0.007767,
                                  0.008512,
                                  0.007253],
               'first and second': [0.006739,
                                    0.006878,
                                    0.005955,
                                    0.006739,
                                    0.007418],
               'last': [0.00646,
                        0.006753,
                        0.005414,
                        0.00586,
                        0.005476]}},
    8: {2: {'first': [0.009762,
                      0.009675,
                      0.010701,
                      0.007362,
                      0.007175],
            'first and last': [0.011825,
                               0.01177,
                               0.011457,
                               0.009661,
                               0.009611],
            'first and second': [0.01092,
                                 0.014204,
                                 0.013526,
                                 0.009901,
                                 0.009765],
            'last': [0.00701,
                     0.006871,
                     0.010002,
                     0.008035,
                     0.008293]},
        800: {'first': [0.012611,
                        0.009181,
                        0.007319,
                        0.008433,
                        0.007569],
              'first and last': [0.013254,
                                 0.012883,
                                 0.011015,
                                 0.012974,
                                 0.009578],
              'first and second': [0.018516,
                                   0.018681,
                                                                  0.019176,
                                                                  0.018859,
                                                                  0.019516],
                                             'last': [0.009171,
                                                      0.007369,
                                                      0.007532,
                                                      0.011004,
                                                      0.009196]},
                                       1600: {'first': [0.009756,
                                                        0.007568,
                                                        0.010311,
                                                        0.007383,
                                                        0.008999],
                                              'first and last': [0.010471,
                                                                 0.010031,
                                                                 0.012248,
                                                                 0.011519,
                                                                 0.01256],
                                              'first and second': [0.018623,
                                                                   0.019316,
                                                                   0.019296,
                                                                   0.018557,
                                                                   0.019013],
                                              'last': [0.007297,
                                                       0.007603,
                                                       0.011624,
                                                       0.008758,
                                                       0.007641]}}}}},
     'versions': {'dc2.large': {2: 'PostgreSQL 8.0.2 on i686-pc-linux-gnu, '
                                   'compiled by GCC gcc (GCC) 3.4.2 20041017 (Red '
                                   'Hat 3.4.2-6.fc3), Redshift 1.0.29551'},
                  'ds2.xlarge': {2: 'PostgreSQL 8.0.2 on i686-pc-linux-gnu, '
                                    'compiled by GCC gcc (GCC) 3.4.2 20041017 (Red '
                                    'Hat 3.4.2-6.fc3), Redshift 1.0.29551'},
                  'ra3.xlplus': {2: 'PostgreSQL 8.0.2 on i686-pc-linux-gnu, '
                                    'compiled by GCC gcc (GCC) 3.4.2 20041017 (Red '
                                    'Hat 3.4.2-6.fc3), Redshift 1.0.29551'}}}
```

Redshift Observatory Slack

I’ve started up a Redshift Slack.

Join URL is;

https://join.slack.com/t/redshiftobservatory/shared_invite/zt-2vm3deqis-hc6h4GMDcG6Gs7~IECQNuQ