# **Couchbase performance testing**

## **Purpose**

This is an effort to validate the feasibility and performance of storing MET data in a Couchbase document database, measure and evaluate the performance of some basic meteorological queries against this dataset, and to learn what are best practices and potential pitfalls.

## **Prerequisites**

1. JAVA - at least version 8
2. python3 (for loading)
   * PyMysql
   * numpy
   * pandas
   * python-dateutil
   * docutils
   * lxml
   * Couchbase (sdk v3.0 +)
3. A Couchbase server
4. On the Couchbase server a Collection named mdata
5. In the mdata collection a bucket named mdata
6. Couchbase python sdk v3 installation guide.. <https://docs.couchbase.com/python-sdk/current/hello-world/start-using-sdk.html> Couchbase cbq - Couchbase command line query tool.
   * See <https://docs.couchbase.com/server/current/tools/cbq-shell.html>
7. METviewer (code for using mv\_mysql.sql script to initialize METviewer databases)
8. METdbload installed (for loading MariaDB)
9. VXingest (for loading Couchbase) See <https://github.com/NOAA-GSL/VxIngest> for the GitHub repository.
10. Environment variables defined
    * export PATH="/opt/couchbase/bin:$PATH"
    * This is where the Couchbase utilities, specifically cbq live. See <https://docs.couchbase.com/server/current/tools/cbq-shell.html>
    * The expectation is that jq, a mysql client, and java are also in the PATH where the tests are being run.

* export JAVA\_HOME="$(/usr/libexec/java\_home)"
* export MV\_HOME=/Users/randy.pierce/IdeaProjects/METviewer
* export METdb=/Users/randy.pierce/METdb
* export CouchbasePerformanceTest=/Users/randy.pierce/WebstormProjects/CouchbasePerformanceTest

### **BASH utilities used.**

* jq <https://stedolan.github.io/jq/>

## **directories**

* bin - contains executables for loading and miscellaneous operations.
* load - load specs that are used by both the sql and cb load scripts.
* test\_cases - test case scripts. These are bash executables that query and massage output data to produce comparable results from different data bases.

## **methodology**

### **Overview**

The intent was to incrementally load redundant datasets with different subsets (subset is a top-level keyword in our test Couchbase document schema) and compare query times for a rational set of meteorological data queries. The test suite queried an instance of MariaDB, a single node Couchbase server, and a multi-node Couchbase cluster. The MariaDB queries accomplished with the mysql command line client, and the Couchbase queries with the cbq command line client. No attempt has been made to qualify the Couchbase client SDK, as that is beyond the scope of this evaluation.

Part way into the evaluation it became clear the size of the data did not have a significant impact on a query that returned a specific subset of data, as long as the query is a key value pair query, or a proper index for the query has been applied, and the data has been distributed across multiple nodes. See table 6. This table shows the difference in querying a single subset versus four subsets i.e. four times the data on the three node cluster. The differences are between 0.6 and 9.3 percent, depending on the test. The average difference is less than 3%. This is for a data set difference of four times. The test with the largest returned data set wasn't the one with the biggest difference. This indicates that the size of the data queried wasn't as much of a factor as the initial assumption. The test results verify the Couchbase documentation which explains that if the data indexing has been done properly, with large datasets distributed across multiple nodes, the data access time can be largely consistent or even improved when querying our meteorological data. This [document](https://info.couchbase.com/rs/302-GJY-034/images/High_Performance_With_Distributed_Caching_Couchbase.pdf) more fully explains how using the Couchbase architecture to distribute data across multiple nodes keeps data access latency small.

#### *Indexing*

It is beyond the scope of this evaluation to research the optimal indexing strategies. The "Couchbase Query Advisor" was used to obtain hints about indexes for the test queries, and those were applied, but it should be realized that better optimization could be obtained with better indexes and more thoughtful application of indexing strategies.

#### *Key Value queries*

From this Couchbase [documentation...](https://docs.couchbase.com/server/5.0/data-access/data-access-intro.html#:~:text=Couchbase%20provides%20multiple%20ways%20of,key%20to%20the%20item%20stored.)

Due to their simplicity, KV operations execute with extremely low latency, often sub-millisecond. While the Query service is accessed by a defined query language (N1QL), the KV store is accessed using simple CRUD (Create, Read, Update, Delete) APIs, and thus provides a simpler interface when accessing documents

The interesting part of using KV queries is determining how the keys will be composed. The key is the ID field, and we discuss it in the following section. There is no higher performance way to query a document store than using a KV operation.

This implies that to best utilize a document store for meteorological data, queries could be made most performant by restructuring them into KV operations whenever possible.

Joins should be avoided, and in the case of METviewer data it is easily accomplished by using a schema that incorporates header data in the document along with all the associated data that corresponds to that header.

By making the keys algorithmically derivable it is possible to programmatically convert queries to KV operations. The test cases intend to provide examples of this.

### 

### **Document Schema**

The two most important discoveries in this exercise turn out to be the importance of the document schema in Couchbase, and the importance of converting long-running N1QL queries into key value pair queries based on the ID field - an artifact of the schema. The basic document schema looks like this…

{

"id": “DD::V01::SAL1L2::mv\_gfs\_grid2obs\_vsdb1::ECM::G2/NHX::HGT::ECM::P1000::1301616000",

"type": "DataDocument",

"dataType": "VSDB\_V01\_SAL1L2",

"subset": "mv\_gfs\_grid2obs\_vsdb1",

"dataFile\_id": "DF\_id",

"datasource\_id": "DS\_id",

"version": "V01",

"model": "ECM",

"geoLocation\_id": "G2/NHX",

"obtype": "ECM",

"fcst\_valid\_beg": "2011-04-01T00:00:00Z",

"fcst\_valid\_epoch": 1301616000,

"fcst\_var": "HGT",

"fcst\_units": null,

"fcst\_lev": "P1000",

"data": [

{

"fcst\_lead": "00",

"total": "3456.",

"fabar": "-0.175314694E+02",

"oabar": "-0.175314694E+02",

"foabar": "0.484113130E+04",

"ffabar": "0.484113130E+04",

"ooabar": "0.484113130E+04",

"mae": "None",

"fcst\_init\_beg": "2011-04-01T00:00:00Z",

"fcst\_init\_epoch": 1301616000,

"fcst\_valid\_beg": "2011-04-01T00:00:00Z",

"fcst\_valid\_epoch": 1301616000

},

{

"fcst\_lead": "24",

"total": "3456.",

"fabar": "-0.173110400E+02",

"oabar": "-0.175314694E+02",

"foabar": "0.483493080E+04",

"ffabar": "0.486662484E+04",

"ooabar": "0.484113130E+04",

"mae": "None",

"fcst\_init\_beg": "2011-03-31T00:00:00Z",

"fcst\_init\_epoch": 1301529600,

"fcst\_valid\_beg": "2011-04-01T00:00:00Z",

"fcst\_valid\_epoch": 1301616000

},

...

]

}

...

The important things to notice about the schema are...

There is an ID field which is algorithmically derived from a subset of the top-level fields. This allows for converting what might be an N1QL sql type query with a predicate clause like  
SELECT r.mdata.geoLocation\_id as vx\_mask, data.fcst\_init\_beg, data.fcst\_valid\_beg, data.fcst\_lead, r.mdata.model, r.mdata.fcst\_lev, r.mdata.fcst\_var, data.total, data.fabar, data.oabar, data.foabar, data.ffabar, data.ooabar

FROM (

SELECT \*

FROM mdata

WHERE model == "GFS"

AND dataType == "VSDB\_V01\_SAL1L2"

AND subset == "mv\_gfs\_grid2obs\_vsdb1"

AND fcst\_var == "HGT"

AND fcst\_valid\_beg BETWEEN "2018-01-01T00:00:00Z" AND "2018-01-04T18:00:00Z") AS r

UNNEST r.mdata.data AS data

WHERE data.fcst\_lead IN ['00', '06', '12', '18', '24', '30', '36', '42', '48', '54', '60', '66', '72', '78', '84', '90', '96', '102', '108', '114', '120', '126', '132', '138', '144', '150', '156', '162', '168', '174', '180', '186', '192', '198', '204', '210', '216', '222', '228', '234', '240', '252', '264', '276', '288', '300', '312', '324', '336', '348', '360', '372', '384']

ORDER BY data.fcst\_valid\_beg, data.fcst\_init\_beg, r.mdata.fcst\_lev, data.fcst\_lead, r.mdata.geoLocation\_id;

into something like this...  
SELECT r.mdata.geoLocation\_id as vx\_mask, data.fcst\_init\_beg, data.fcst\_valid\_beg, data.fcst\_lead, r.mdata.model, r.mdata.fcst\_lev, r.mdata.fcst\_var, data.total, data.fabar, data.oabar, data.foabar, data.ffabar, data.ooabar

FROM

(SELECT \*

FROM mdata USE KEYS [

"DD::V01::SAL1L2::mv\_gfs\_grid2obs\_vsdb1::GFS::G2::HGT::GFS::P1000::1514764800",

"DD::V01::SAL1L2::mv\_gfs\_grid2obs\_vsdb1::GFS::G2::HGT::GFS::P250::1514764800",

...

]) AS r

UNNEST r.mdata.data AS data

WHERE data.fcst\_lead IN ['00', '06', '12', '18', '24', '30', '36', '42', '48', '54', '60', '66', '72', '78', '84', '90', '96', '102', '108', '114', '120', '126', '132', '138', '144', '150', '156', '162', '168', '174', '180', '186', '192', '198', '204', '210', '216', '222', '228', '234', '240', '252', '264', '276', '288', '300', '312', '324', '336', '348', '360', '372', '384']

ORDER BY data.fcst\_valid\_beg, data.fcst\_init\_beg, r.mdata.fcst\_lev, data.fcst\_lead, r.mdata.geoLocation\_id;

* with the keys derived algorithmically and inserted into the query. This is from example test\_2 and resulted in a key value query time 20% of the predicate based query time.  
  In a program it would be possible to use a native API call from the Couchbase SDK to make this query even more performant because the data would be returned to the client program as a native data structure.
* There is a data section that is essentially an array of all the fcst\_lead times that are available for this particular data set at this particular valid time. This data representation could be an ordered array or a map, it doesn't matter, N1QL is capable of returning the data in a low impedance manner regardless.
* This schema has the effect of combining all the data that would be associated with a given stat header in METviewer into a single document in Couchbase. This does several things.
  1. It has the natural effect of data compression.
  2. It has the effect of reducing query time because documents get indexed and having more related data in a single document means fewer documents get indexed.
  3. Reducing the index size reduces the server memory requirements.

### **Data**

The raw data that gets loaded into MariaDB and Couchbase is composed of several years of VSDB data (The data goes from Nov 5th, 2006 through Nov 30, 2019) for GFS and ECM. There are 95,411 files which average about 1.5 MB each consisting of SL1L2, SAL1L2, VL1L2, and VAL1L2 data.

The original data set resides on hera in /scratch1/NCEPDEV/global/Fanglin.Yang/stat/vsdb\_data and was copied to the local server adb-cb1 and rearranged in order to make it easier to load with MetDBLoad and VXingest. Having the data local minimized the effects of a network mount point on loading times. This data set has years of accumulated VSDB files and is continually being appended by EMC, which is why it was chosen. The rearranged data lives in /public/retro/pierce/vsdb\_data which is a GSL cummulo mount point as well as on adb-cb1.

This is the load\_val section of the mv\_load spec.

```

<load\_val>

<field name="stattype">

<val>anom</val>

<val>pres</val>

<val>sfc</val>

<val>grid2obs</val>

</field>

<field name="cycle">

<val>00Z</val>

<val>06Z</val>

<val>12Z</val>

<val>18Z</val>

</field>

<field name="model">

<val>gfs</val>

<val>ecm</val>

</field>

</load\_val>

```

This load spec represents 95,411 data files and about 142GB of raw data on disk.

To make the actual dataset larger in the Couchbase database the data gets loaded into different subsets in the same bucket of the Couchbase server. This is analogous to putting the same data into different databases in a METviewer MariaDB database server.

Initially, there was an intent to add redundant data to the cluster incrementally and run the tests to measure the query time degradation due to having more data, but after doing this once or twice it became obvious any negative effects due to adding more data get limited by network transfer time (if asking for more data) and the effectiveness of the indexing, not the size of the dataset. With proper indexing in place or using key value queries, the effects of extra data were minimal. It was decided to abandon the initial approach and to concentrate instead on the differences between SQL, a single Couchbase node, and a Couchbase cluster, as well as the types and structure of the test queries.

For the single node Couchbase server two redundant copies get loaded, subset mv\_gfs\_grid2obs\_vsdb and mv\_gfs\_grid2obs\_vsdb1 This resulted in 63,576,906 documents which represents 190,822 raw vsdb files in the single node Couchbase server for a database disk space of 132GB and a raw disk space of 284GB. This is data compression (due to schema not actual compression) of around 36%.

The cluster got loaded with twice as much data in order to see how it impacted the query times. The cluster had 128,342,347 documents representing 381,644 files and which consumed 578GB of disk database space representing 764GB of raw disk. This is a data reduction of 25%. The lower data compression rate, relative to the raw data, is probably due to this data being more redundant. There are four copies of the same data which means that the header section got replicated 4 times for the data set, relative to the raw data.

Because of the Couchbase schema each Couchbase document represents more than one data record. Each document has all the records for a given valid time qualified by the model, the variable, like this...

"dataType": "VSDB\_V01\_SAL1L2",

"subset": "mv\_gfs\_grid2obs\_vsdb1",

"version": "V01",

"model": "ECM",

"geoLocation\_id": "G2/NHX",

"obtype": "ECM",

"fcst\_valid\_beg": "2011-04-01T00:00:00Z",

"fcst\_valid\_epoch": 1301616000,

"fcst\_var": "HGT",

"fcst\_lev": "P1000",

The field subset allows redundancy. For example, one set of data has the subset "mv\_gfs\_grid2obs\_vsdb", the next "mv\_gfs\_grid2obs\_vsdb1", "the next mv\_gfs\_grid2obs\_vsdb2" etc.

### **Data Loading**

#### *MariaDB*

The data gets loaded to mysql with bin/run\_metdbload\_mysql.sh and/or with bin/run\_mvload\_mysql.sh. Before loading data, a database must be preconfigured with the METviewer/sql/mv\_mysql.sql script.

For load comparisons multiple user mariaDB databases can be used.

#### *Couchbase*

The data gets loaded to a Couchbase bucket with bin/run\_cb.sh which is a wrapper for the run\_cb\_threads.py program. (<https://github.com/NOAA-GSL/VxIngest>)

### 

### **Tests**

After the data gets loaded you can run the tests in the test\_cases directory. The test cases are sets of tests representing queries that were captured from METviewer for real plots appropriate for this dataset. The tests are shell scripts and are in the test\_cases directory of this repository. Each test has this usage...

./test....sh -s server -S subset(or database) [-p(prints prologue)]

Each test case i.e. test\_n\_.... has multiple flavours which are delineated by the part of the name after the first 'test\_'. The output of running a test goes into the output directory and retains the first part of the name i.e. test\_n\_..... and appends .out. These outputs get transformed in the test script to enable them to be compared. The JSON output from Couchbase N1QL and key value queries get converted to tabular format so that they can be compared to the SQL output. The outputs and intermediate transformation files get retained. When a given test finishes there may be several '.out' files that start with the same 'test\_n\_'. These test outputs should successfully compare.

These are the test flavours...

* sql.sh uses a sql server query
* cached\_sql.sh attempts to use sql server caching
* epoch\_cb.sh uses an epoch predicate in an N1QL query to qualify all the date fields in the date range.
* iso\_cb.sh attempts to use an ISO date predicate in an N1QL query to qualify a date range.
* keys\_cb.sh attempts to derive keys to do a key value query against the document store.

Each test\_n where the n is the same should return the same data. The output data gets massaged in each test script to produce an output that can be compared. Each test script will store its output in the output directory. The output data files contain performance data such as query time.

Each test script takes one of these parameters.

* -s server - specifies the mariaDB or CB server
* -S subset - specifies the Couchbase subset or the sql database
* -p prints out the profile for the test
* -h prints the test script help

The script "run\_all\_tests.sh" runs the entire test suite and produces formatted results on std\_out.

##Test Descriptions

##Test Results

## **Test Descriptions**

#### *test\_1\_sql.sh*

This is a basic test to determine the ability to query header and data fields and to qualify header predicate values in a subselect, using an inclusive range for fcst\_valid\_beg values, 2018-01-01 00:00:00 through 2018-01-04 18:00:00 and to to further qualify the data portion with a set of fcst\_leads 0 through 384, for model 'GFS', fcst\_lev="P1000", and domain "G2/NHX". This test is for sql MariaDB.

#### *test\_1\_cached\_sql.sh*

This test differs from test\_1\_sql.sh in that it is for cached sql. To be sure it uses cached data, it is for a different date range. This test query is for MariaDB with sql.

#### *test\_1\_epoch\_cb.sh*

This test differs from test\_1\_sql.sh in that it is a N1QL query against Couchbase using an array of epochs to qualify the fcst\_valid\_beg dates.

#### *test\_1\_iso\_cb.sh*

This test differs from test\_1\_sql.sh in that it is a N1QL query against Couchbase using an array of ISO dates to qualify the fcst\_valid\_beg dates.

#### *test\_1\_iso\_range.sh*

This test differs from test\_1\_sql.sh in that it is a N1QL query against Couchbase using a range of ISO dates to qualify the fcst\_valid\_beg dates eg. "BETWEEN '2018-01-01T00:00:00Z' AND '2018-01-04T18:00:00Z'". This test is for Couchbase.

#### *test\_1\_matchcached\_cb.sh*

This is an N1QL query similar to test\_1\_sql\_cached.sh for purposes of verifying the results. This test is for Couchbase.

#### *test\_2\_sql.sh*

This is a basic test to determine the ability to query header and data fields and to qualify one header predicate value in a subselect, leaving others to default to all possible values, and using an inclusive range for fcst\_valid\_beg ISO values, and to to further qualify the data portion with a specific set of fcst\_leads. It uses a query copied from a METviewer plot that returns data fields qualified with a subselect that specifies a valid begin time array of date elements 2019-01-01 00:00:00 through 2019-03-10 00:00:00

for gfs with domains "G2/NHX" and "G2/SHX" with fcst\_var "HGT" and all the fcst\_lev. The valid begin timestamps specified in an array. This is an SQL test against MariaDb.

#### *test\_2\_cached\_sql.sh*

This test differs from test\_2\_sql.sh in that it is for cached sql. To be sure it uses cached data it is for a different date range.

#### *test\_2\_cb.sh*

This test differs from test\_2\_sql.sh in that is is an N1QL query against Couchbase.

#### *test\_2\_keys\_cb.sh*

This test differs from test\_2\_sql.sh in that is is an N1QL query against Couchbase and the query is a key value query with keys derived and contained in an array like ...

[ "DD::V01::SAL1L2::mv\_gfs\_grid2obs\_vsdb1::GFS::G2/NHX::HGT::GFS::P500::1546300800"

....]

#### *test\_2\_matchcached\_cb.sh*

This test differs from test\_2\_cached\_sql.sh in that is is an N1QL query against Couchbase for the same valid\_beg date range that matches the sql test\_2 cached test.

#### *test\_3\_sql.sh*

This is a basic test query copied from a METviewer plot that returns data fields qualified with a subselect that specifies a valid begin time array of date elements 2019-01-01 00:00:00 through 2019-03-10 00:00:00

for gfs with domains "G2/NHX" and "G2/SHX" with fcst\_var "HGT" and fcst\_lev "P500". The valid begin timestamps specified in an array. This test differs from test\_2 by specifying a single fcst\_lev. This is an SQL test against MariaDb.

#### *test\_3\_iso\_cb.sh*

This test differs from test\_3\_sql.sh in that it is a N1QL query against Couchbase using an array of ISO dates to qualify the fcst\_valid\_beg dates.

#### *test\_3\_iso\_range\_cb.sh*

This test differs from test\_3\_sql.sh in that it is a N1QL query against Couchbase using a range of ISO dates to qualify the fcst\_valid\_beg. dates e.g. "BETWEEN '2019-01-01T00:00:00Z' AND '2019-03-10T00:00:00Z'".

#### *test\_3\_keys\_cb.sh*

This is a basic test query copied from a METviewer plot that returns data fields qualified with a subselect that specifies a valid begin time range 2019-01-01T00:00:00Z through 2019-03-10T00:00:00Z for gfs with domains "G2/NHX" and "G2/SHX" with fcst\_var "HGT" and fcst\_lev "P500". The query is a key value query with keys derived and contained in an array like ...

[ "DD::V01::SAL1L2::mv\_gfs\_grid2obs\_vsdb1::GFS::G2/NHX::HGT::GFS::P500::1546300800"

....]

This is a Couchbase query.

#### *test\_4\_sql.sh*

This is a basic test query copied from a METviewer plot that returns data fields qualified with a subselect that specifies a dataType == "VSDB\_V01\_SAL1L2", valid begin time array of date elements 2019-01-01 00:00:00 through 2019-03-10 00:00:00

for gfs with domains "G2/NHX" and "G2/SHX" with fcst\_var "HGT" and fcst\_lev "P500, and P750". The valid begin timestamps specified in an array. This test differs from test\_3 in that it specifies multiple levels. This is an SQL test against MariaDb.

#### *test\_4\_iso\_cb.sh*

This N1QL Couchbase test differs from test\_3 in that it specifies multiple levels. This test differs from test\_4\_sql.sh in that the valid begin timestamps are specified in an array of ISO dates.

#### *test\_4\_iso\_range\_cb.sh*

This N1QL Couchbase test differs from test\_3 in that it specifies multiple levels. This test differs from test\_4\_sql.sh in that the valid begin timestamps are specified in a range of ISO dates.

#### *test\_4\_keys\_cb.sh*

This test differs from test\_4\_iso\_range\_cb in that the query is a key value query with keys derived and contained in an array like ...

[ "DD::V01::SAL1L2::mv\_gfs\_grid2obs\_vsdb1::GFS::G2/NHX::HGT::GFS::P500::1546300800"

....]

This is an N1QL Couchbase query.

#### *test\_5\_sql.sh*

This is a basic test query copied from a METviewer plot that returns data fields qualified with a subselect that specifies a dataType == "VSDB\_V01\_SAL1L2", valid begin time array of date elements 2019-01-01 00:00:00 through 2019-03-10 00:00:00 for gfs with domains "G2/NHX" and "G2/SHX" with fcst\_var "HGT" and fcst\_lev "'P10', 'P20', 'P30', 'P50', 'P70', 'P100', 'P150', 'P200', 'P250', 'P300', 'P400', 'P500', 'P700', 'P850', 'P925', 'P1000'". The valid begin timestamps specified in an array. This test differs from test\_3 in that it specifies many levels and only one fcst lead time. Levels 'P10', 'P20', 'P30', 'P50', 'P70', 'P100', 'P150', 'P200', 'P250', 'P300', 'P400', 'P500', 'P700', 'P850', 'P925', 'P1000' This is an SQL test against MariaDb.

#### *test\_5\_iso\_range\_cb.sh*

This test differs from test\_5\_sql.sh by specifying the valid begin timestamps in a range of ISO dates.

#### *test\_5\_keys\_cb.sh*

This test differs from test\_5\_iso\_range\_cb in that the query is a key value query with keys derived and contained in an array like ...

[ "DD::V01::SAL1L2::mv\_gfs\_grid2obs\_vsdb1::GFS::G2/NHX::HGT::GFS::P500::1546300800"

....]

This is an N1QL Couchbase query.

#### *test\_6\_sql.sh*

This is a basic test query copied from a METviewer plot that returns data fields qualified with a subselect that specifies a dataType == "VSDB\_V01\_VAL1L2" valid begin time array of date elements 2019-01-01 00:00:00 through 2019-03-10 00:00:00 for model 'GFS' with domains "G2/NHX" and "G2/SHX" with fcst\_var "WIND", and level 'P500', This test differs from test\_5 in that it specifies only one level and a different variable. This is an SQL test against MariaDb.

#### *test\_6\_iso\_cb.sh*

This test differs from test\_6\_sql.sh in that the valid begin timestamps are specified in an array of ISO dates.

#### *test\_6\_iso\_range\_cb.sh*

This test differs from test\_6\_sql.sh by specifying the valid begin timestamps in a range of ISO dates.

#### *test\_6\_keys\_cb.sh*

This test differs from test\_6\_iso\_range\_cb in that the query is a key value query with keys derived and contained in an array like ...

[ "DD::V01::VAL1L2::mv\_gfs\_grid2obs\_vsdb1::GFS::G2/NHX::WIND::GFS::P500::1546300800"

....]

#### *test\_9\_sql.sh*

This test differs from test\_6\_sql.sh by querying a different data set 'line\_data\_sal1l2' and a different variable 'HGT'.

#### *test\_9\_keys\_cb.sh*

This test differs from test\_6\_iso\_range\_cb in that the query is a key value query with keys derived and contained in an array like ...

[ "DD::V01::SAL1L2::mv\_gfs\_grid2obs\_vsdb1::GFS::G2/NHX::HGT::GFS::P500::1546300800"

....]

## **Test Results**

#### *These results came from the file 20200825:18:02:22-vsdb2.out.*

This file is the output of a test run "run\_all\_tests.sh" that was run on a standalone server, adb-cb1 which is identical in hardware and software to any of the three nodes of a Couchbase cluster, adb-cb4.

The tests are querying a single node Couchbase server on adb-cb1, a three node Couchbase cluster on adb-cb4 - which is the entry point for the cluster, and a single instance of MariaDB also running on adb-cb1.

The clients for the queries were an instance of mysql, and an instance of cbq (similar to mysql but part of the Couchbase server tools).

Some verbose details of this output got truncated for purposes of identifying results. The truncated and semi-processed results are in the file 20200825:18:02:22-vsdb2.out.times.txt. I used a spreadsheet to process those results to obtain the following tables. Each number represents the number of milliseconds it took to process the query part of the test.

There were two tests that were aimed at discovering if using SQL cache made a huge difference on these queries. The results indicate that for these queries, not a lot of difference, so I omitted those queries from the other tests.

#### *Exclusions*

Not every test performs every action. For example test\_9 only does a key value test trying to show the difference of doing a large key value query against a single node versus against a cluster.

Several other tests skip the epoch based query because test\_1 was clearly demonstrating that using epochs is faster and this result is pretty intuitive, actually.

#### *What is source code controlled*

The actual query data is not in source code management as the queries were excessively large, and the total data size for these 7 tests was nearly 500 MB. However the tests results are easily reproducible against the data set. We do not anticipate destroying the data set in the near future, and the raw data is available from hera so that it can always be re-ingested. The load directory has the necessary load\_spec xml files, and the bin directory has scripts that can be used to perform the data ingest if necessary.

**NOTE:** *One of these values (CBC iso test 6 - 34598) is consistently long-running against the cluster, but not the single node. This is not yet understood. The problem can be observed across multiple test runs.* Further investigation required.

##### All values are query execution time in milliseconds

##### table 1 - Tabulated results sql versus single cluster versus multi-node cluster

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TEST | SQL | SQL cached | CBS epoch | CBS iso | CBS ISO range | CBS keys | CBC epoch | CBC iso | CBC ISO range | CBC keys |
| 1 | 51 | 51 | 47 | 107 | ---- | ---- | 63 | 214 | ---- | ---- |
| 2 | 9940 | 9860 | ---- | ---- | 3778 | **729** | ---- | ---- | 5120 | **738** |
| 3 | 1170 | ---- | ---- | 1649 | 1207 | **1158** | ---- | 4150 | 1264 | **1238** |
| 4 | 1160 | ---- | ---- | 2004 | 1187 | **1175** | ---- | 4435 | 1306 | **1255** |
| 5 | 4260 | ---- | ---- | 11195 | 1498 | **1803** | ---- | 11268 | 1594 | **1987** |
| 6 | ---- | ---- | ---- | 1711 | 1240 | 1310 | ---- | 34599 | 1398 | 1244 |
| 9 | ---- | ---- | ---- | ---- | ---- | 1146 | ---- | ---- | ---- | 1139 |
| AVG | 3,408 | 5,185 | 47 | 3,333 | 1,782 | **1,220** | 63 | 10,933 | 2,136 | **1,267** |

##### table 2 - Tabulated results sql versus sql cached.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TEST | SQL | SQL cached | difference | percent difference |
| 1 | 510 | 510 | 0 | 0 |
| 2 | 9940 | 9860 | 80 | 0.8113590264 |

##### table 3 - Tabulated results ISO date N1QL versus epoch

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TEST | CBS epoch | CBS iso | difference | percent | CBC epoch | CBC iso | difference | percent |
| 1 | 47 | 107 | 60 | 56.07 | 63 | 214 | 151 | 70.56 |

##### 

##### table 4 - Tabulated results ISO N1QL versus ISO range

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TEST | CBS iso | CBS ISO range | difference | percent | CBC iso | CBC ISO range | difference | percent |
| 3 | 1649 | 1207 | 442 | 26.80 | 4150 | 1264 | 2886 | 69.54 |
| 4 | 2004 | 1187 | 817 | 40.76 | 4435 | 1306 | 3129 | 70.55 |
| 5 | 11195 | 1498 | 9697 | 86.61 | 11268 | 1594 | 9674 | 85.85 |
| 6 | 1711 | 1240 | 471 | 27.52 | 34599 | 1398 | 33201 | 95.95 |

##### table 5 - Tabulated results sql versus ISO range N1QL versus key/value

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TEST | SQL | CBS ISO range | difference to sql | percent diff sql |
| 2 | 9940 | 3778 | 6162 | 61.99 |
| 3 | 1170 | 1207 | -37 | -3.16 |
| 4 | 1160 | 1187 | -27 | -2.32 |
| 5 | 4260 | 1498 | 2762 | 64.83 |
| TEST | SQL | CBC keys |  |  |
| 2 | 9940 | 738 | 9202 | 92.57 |
| 3 | 1170 | 1238 | -68 | -5.81 |
| 4 | 1160 | 1255 | -95 | -8.18 |
| 5 | 4260 | 1987 | 2273 | 53.35 |

##### table 6 - Tabulated results N1Ql kv query single node versus cluster

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TEST | CBS keys | CBC keys | CBC - CBS | percent |
| 2 | 729 | 738 | 9 | 1.21 |
| 3 | 1158 | 1238 | 80 | 6.46 |
| 4 | 1175 | 1255 | 80 | 6.37 |
| 5 | 1803 | 1987 | 184 | 9.26 |
| 6 | 1310 | 1244 | -66 | -5.3 |
| 9 | 1146 | 1139 | -7 | -0.61 |

##### table 7 - Tabulated results sizes in bytes

|  |  |  |
| --- | --- | --- |
| TEST | record count | size in bytes |
| test\_1\_cached\_sql.sh | 848 | 1.55E+05 |
| test\_1\_sql.sh | 848 | 1.53E+05 |
| test\_2\_cached\_sql.sh | 16960 | 3.10E+06 |
| test\_2\_sql.sh | 16960 | 3.10E+06 |
| test\_3\_sql.sh | 28718 | 5.20E+06 |
| test\_4\_sql.sh | 28718 | 5.20E+06 |
| test\_5\_sql.sh | 2168 | 4.10E+05 |
| test\_6\_sql.sh | 28718 | 5.80E+06 |
| test\_9\_sql.sh | 28718 | 5.20E+06 |
| test\_1\_epoch\_cb.sh | 848 | 3.82E+05 |
| test\_1\_iso\_cb.sh | 848 | 3.82E+05 |
| test\_1\_matchcached\_cb.sh | 848 | 3.82E+05 |
| test\_2\_cb.sh | 16960 | 7.63E+06 |
| test\_2\_keys\_cb.sh | 16960 | 7.63E+06 |
| test\_2\_matchcached\_cb.sh | 16960 | 7.63E+06 |
| test\_3\_iso\_cb.sh | 28718 | 1.29E+07 |
| test\_3\_iso\_range\_cb.sh | 28718 | 1.29E+07 |
| test\_3\_keys\_cb.sh | 28718 | 1.29E+07 |
| test\_4\_iso\_cb.sh | 28718 | 1.29E+07 |
| test\_4\_iso\_range\_cb.sh | 28718 | 1.29E+07 |
| test\_4\_keys\_cb.sh | 28718 | 1.29E+07 |
| test\_5\_iso\_cb.sh | 2168 | 9.77E+05 |
| test\_5\_iso\_range\_cb.sh | 2168 | 9.77E+05 |
| test\_5\_keys\_cb.sh | 2168 | 9.77E+05 |
| test\_6\_iso\_cb.sh | 27636 | 1.48E+07 |
| test\_6\_iso\_range\_cb.sh | 27636 | 1.48E+07 |
| test\_6\_keys\_cb.sh | 27636 | 1.48E+07 |
| test\_9\_keys\_cb.sh | 27636 | 1.24E+07 |

## **Observations**

### **Feasibility**

The feasibility of converting a METviewer or METexpress query to a query that is suitable for a Couchbase document store is demonstrated. METviewer tends to create queries that

utilize an array of explicit date strings, and by applying an algorithm using the predicate values these lists of dates can readily be converted to a list of keys for this proposed schema. These keys can be used for a key value query into the Couchbase database and this new query will result in the best overall performance possible for Couchbase, and that performance will generally well exceed the performance of the original query against the MariaDB METviewer database.

Alternatively, there will be many cases where the original query can be slightly modified into an N1QL query, changing the date strings into epochs, and this will result in a query with acceptable performance. METviewer usually requires a join between data and header tables and these joins need to be reduced. It is also advantageous to form the query with a subselect before applying the predicate. It is most performant to form the subselect with a date range.

There is a distinct advantage with respect to the scalability. Where the MariaDB database performance degrades with increasing data volume, the Couchbase database can be scaled horizontally by adding nodes and rebalancing and the query times do not suffer much at all, as long as the schema is rational, no joins are used, subselects are used, and key value queries are used either for everything or for long running queries.

Please refer to table 1. This table contains a summary of the test results. The key value queries have been bolded. In all cases the Couchbase key value times are either less (some are a factor of 10 smaller) or approximate to the SQL times.

Please refer to table 6. This table illustrates the difference between the queries against a single node Couchbase server and the three node Couchbase server with four times the data. The percent difference is always less than 10% and in all but one case less than 6.5%. In two cases the cluster was actually faster than the single node.

### **Correlation to Couchbase best practices**

Many of these results corroborate Couchbase best practices. For example this document [Couchbase Querying Data](https://docs.couchbase.com/server/5.0/architecture/querying-data-and-query-data-service.html) clearly states that using key value queries is the most performant type of query.

This method is restrictive as you can use this method only if you have the document keys in your hand. However, it is the fastest way to retrieve data.

By using this document schema we can derive the keys algorithmically, so we'll always have the keys (or the ability to create them) "on hand". This depends on using a dependable algorithm for ingesting the data as well as retrieving it. I suggest that for MET style data (and I suspect all meteorological verification data) this technique is very relevant. **The algorithms that are to be used for creating the keys should be part of the metadata for the document store.** The test cases themselves do provide examples of deriving keys algorithmically. The algorithms (at least for this dataset) are simple concatenations.

### **Key Value queries verses N1QL with predicate clauses**

The tests demonstrate (see table 5) that the quickest queries use keys. However, there were two tests (3 and 4) where it was close. There will be times where the dataset can be adequately qualified with a subselect or other means whereby a straight N1QL query based on predicates will be sufficient. It will depend on the data. But there were never times where the straight SQL or N1QL query was dramatically better than the key value query. That said there will be many times that a predicate based N1QL query is desired and performant enough.

### **Uncached SQL versus cached SQL**

See table 2. There was not a noticeable difference between cached and uncached SQL queries. This might be because the datasets retrieved were quite large in these tests or it might be that with a better understanding of tuning the cached SQL queries would demonstrate better performance.

### **ISO dates versus epochs.**

With these tests epochs are always faster than querying by ISO dates. I suspect that will always be true, because epochs are ordinal integers. The tests indicate that the difference is substantive, see table 3. Epoch based queries are more than twice as fast. Considering that almost all meteorological verification data is date and time based, but never sub-second times it makes good sense to base queries on epochs. **NOTE:** That does not mean that it isn't a good idea to ingest ISO dates into the documents, as there may be good reasons for doing that, just don't qualify the query by them. Always have epochs along with the dates to qualify the query.

### **Ranges versus arrays of dates**

Ranges are faster than explicit arrays of dates. See table 4. For tests 3 and 6 the range queries were nearly 4 times faster. When given a choice of formulating a query a date/time range with epochs might be the faster way to go. This does mean that interval processing and processing holes in the data might require extra code in the data processing. That is an application decision, but ranges appear to be faster.

### **Data scaling**

The addition of more data to the cluster did not negatively affect performance. See table 6. The CBC server (cluster) has four times the data of the CBS (single node) server yet showed a negligible drop in performance. On two test cases, 6 and 9, the performance was even (although very marginally) better.

## **Conclusion**

The use of Couchbase (or really any document oriented database) is quite amenable to meteorological verification data. The expeditious use of key value queries, epochs for timestamps, subselect clauses, and ranges for dates, coupled with an appropriately designed schema that includes header data with an associated data structure, can be very performant even as data scales up.