Telemetry Database Query Performance Review

SophosLabs - Network Security Group

Michael Shannon
Vulnerability Research Manager, SophosLabs
michael.shannon@sophos.com

Christopher Benninger
Linux Deep Packet Inspection Developer, SophosLabs
chris.benninger@sophos.com

Revision 1.0 - March 4, 2014

Table of Contents

- Executive Summary
- Introduction
    - Goals and Justification
    - Hypothesis
    - Disclaimer
- Test Methodology
    - Environment
    - Databases
    - Installation
    - Test Procedure
    - Schema and Data
        - Important note on Indexes
- Test Results
    - Test 1
    - Test 2
    - Test 3
    - Test 4
    - Test 5
    - Test 6
    - Test 7
    - Test 8
    - Test 9
    - Data Size
- Conclusions
- Sources

Executive Summary

Our available datastores were unable to service our proposed workload with suitable response times without significant development cost and complexity. After some research, columnar (aka: tabular datastores) were identified as potential modern solutions. We tested 3 available column-stores against a sample of our existing workload. These tests concluded that the columnar datastores in question provide significantly better performance with respect to query response time. Additional benefits of these solutions include substantially less development and maintenance and far less overall system complexity.

Introduction

This document is an overview of work done to attempt to rank some commonly used analytically-focused database technologies against our currently used infrastructure and traditional data-storage approaches. We will attempt to improve our current systems based on the work done here which will enable us to make an informed decision as to which technology to adopt in the future.

Goals and Justification

The purpose of this work is to determine if there is a better technology than the current infrastructure offers for our particular workload. The workload we are interested in is an analytical one with the focus on determining "big-picture" views and facts, that can support a working data set in the terabytes, without sacrificing write performance. In our ideal solution, ad-hoc queries will not require any changes of the datastore in order to provide a result inline with current user expectations of a responsive web app.

Queries on our current infrastructure often time out, and those that don't can take tens of minutes to complete. To enhance the performance of known queries, additional views of the data are created by a mixture of temporary tables and summary tables. Developing and maintaining these tables are costly and can result in unnecessary system complexity. This hinders maintenance, future development and auditing. As a result, development is required for implementation of a new query, and thus data mining using ad-hoc queries is impractical.

Some organizations traditionally handle these workloads using an OLAP middleware. As our current infrastructure doesn't provide an OLAP-like service, and because the industry has come out with a number of *non-OLAP alternative tools* specifically designed for this workload, we are investigating these OLAP alternatives.

Hypothesis

Based on other industry and academic research regarding column-stores vs row-stores our expectation is that for the workload presented, our queries should perform roughly one order of magnitude better than our current infrastructure given the same hardware and dataset.

Disclaimer

This investigation was performed with a very limited budget and time, by two individuals who are not DBAs, and do not have prior experience with the columnar-datastores tested. No effort was put into performance-enhancement of the test systems. These test are in no way meant to be exhaustive.

Test Methodology

Environment

CPU: 8-core Intel Core i7-2600 CPU @ 3.40GHz

Memory: 16GB

HDD: 500GB: 7200 RPM disk

OS: Debian Jessie (testing)

Databases

The databases chosen for review are based on an earlier review done by our team whose goal was to determine the most commonly used and more performant technologies available at the time. The concluded list consisted of the following technologies:

- InfiniDB - http://infinidb.co
- MonetDB - http://www.monetdb.org
- HP Vertica - http://www.vertica.com

As a baseline:

- MySQL - http://www.mysql.com
- MySQL_DB6 - This is a live machine in our infrastructure. We have added this wherever possible to provide yet another reference point for our MySQL configuration although it is not something to base decisions on, as it is highly optimized, and on totally different hardware

Some others we considered but later decided against testing are as follows:

- SybaseIQ - http://www.sybase.ca - Disqualified as a result of licensing costs
- Datomic - http://www.datomic.com - Postponed due to time constraints, query model vastly different. Future plans for testing tentative.
- InfoBright - https://www.infobright.com - Very similar to InfiniDB, time constraints meant we picked the more popular.

Test Procedure

Our tests consisted of either a) a single query run 5 times, or b) A single query run 5 times with different parameters 5 times each. The test procedure is intended to determine how a repeatedly run query's performance changes when it is repeated identically, and when the parameters change slightly. Each database is restarted before each query so we can see performance on a dry cache as well as a full one.

We chose a subset of queries from our currently existing internal-application with which to run our tests.

Schema and Data

Current storage requirements are: receiving and storing large growing and erratic volumes of message submissions, in combination with daily status reports on resource usage of deployed nodes. The rate and volume of the message submissions are unpredictable. All submissions (message and status) will need to be stored for 3 or more years.

We have traditionally stored submissions in de-normalized, 'flat', *general-purpose traditional row stores* (GPTRS) such as MySQL or PostgreSQL.

The schema used in our test is as follows for MySQL:

```
CREATE TABLE messages_original.status_submission (
       device_uuid      binary(16),
       hostname         VARCHAR(256),
       engine_version   VARCHAR(11),
       data_version  VARCHAR(11),
       license          CHAR(32),
       apache_timestamp TIMESTAMP,
       memory_used      INTEGER,
       memory_total     INTEGER,
       swap_total       INTEGER,
       swap_used        INTEGER,
       cpu_load_total   NUMERIC(20,17),
       cpu_count        INTEGER
);
```

```
CREATE TABLE messages_original.message_submission (
      device_uuid        binary(16),
      sid                INTEGER,
      apache_timestamp   TIMESTAMP,
      reported_timestamp TIMESTAMP,
      origin_ip         CHAR(15)
);
```

For the other DB's in our list do not benefit from having an indexed binary field, so the 'device_uuid' field was left as a VARCHAR as follows:

```
CREATE TABLE messages_original.status_submission (
      device_uuid      VARCHAR(40),
      ...
);
CREATE TABLE messages_original.message_submission (
      device_uuid      VARCHAR(40),
      ...
);
```

Data used in the tests was a snapshot of the data from an internal datastore. This data consists of 96 million message records and 12 thousand status submissions.

Important note on Indexes

It is commonly understood that MySQL does not perform well when key columns are not indexed. As a result, we added an index for each field in the "message" table. However, we did not create any composite indexes (indexes which are on multiple fields) as the permutations of these would result in unfeasibly large index data as the DB grows.

It is also noted that other DB's do not rely on indexes, therefore we have not added indexes to any DB's in our test other than MySQL.

Test Results

Test 1

Summary:
What is the total number of message submissions on a given day?

Example SQL Query:
```
SELECT COUNT(*)
FROM messages_original.message_submission
WHERE apache_timestamp BETWEEN '2014-01-01' AND '2014-01-02';
```

Process:
As there are no changing parameters for this query, we run it 5 times with no changes.

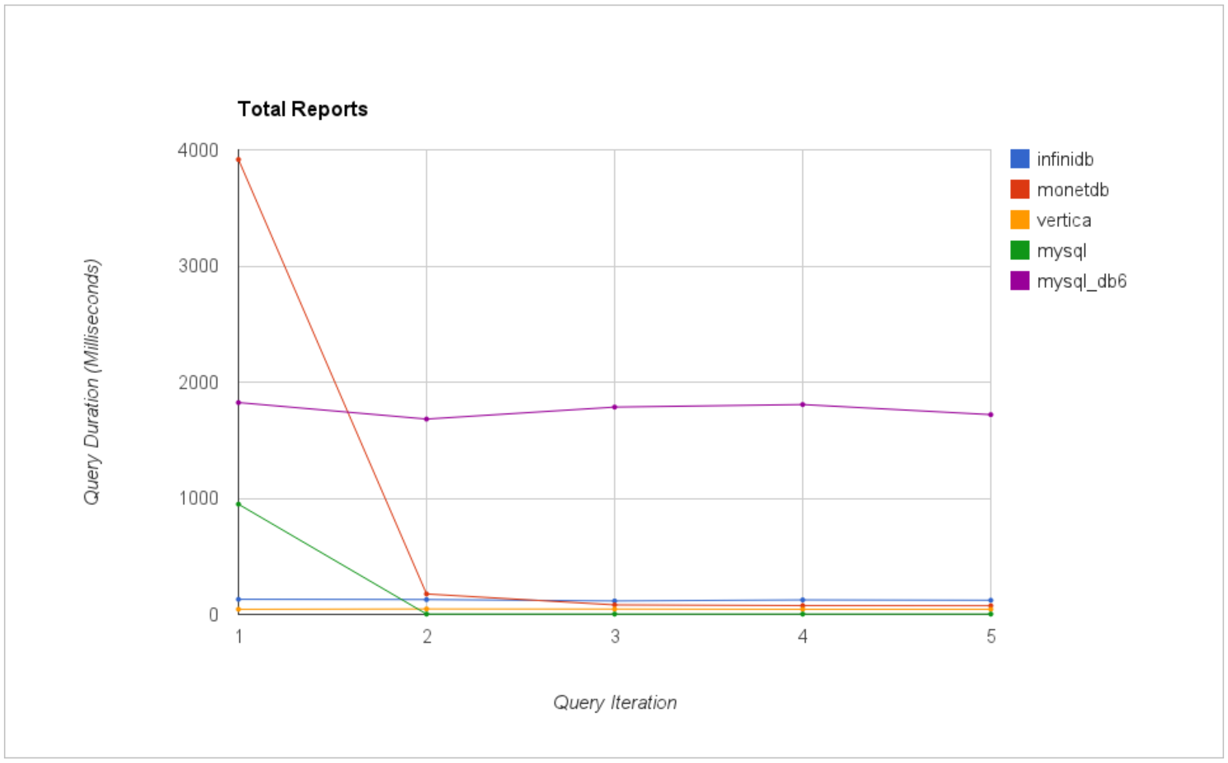

Figure 1.1 - Total Message Submissions

Explanation

- MonetDB performed the worst in this case, likely due to it’s implementation being based on MySQL in the backend and without having an index in place. However later iterations improved with it’s query cache.
- MySQL performed poorly as well comparatively, although it’s query cache sped up later repetitions.
- MySQL_DB performed nearly twice as poorly as MySQL, and as it has the query cache disabled, we do not see an improvement over time. This is unsurprising however as this is a live database with more data than our test data.
- The clear winners here are InfiniDB and Vertica, which were significantly faster in the initial query with nearly exact performance on later iterations.

Test 2

Summary:
What is the total number of status submissions submitted on a given day from a specific UUID?

Example SQL Query:
```
SELECT COUNT(*) FROM messages_original.message_submission
WHERE apache_timestamp BETWEEN '2014-01-01' AND '2014-01-02'
AND device_uuid=UNHEX('<UUID>');
```

Process:
This query was run with 5 different UUIDs (chosen as the 5 most frequent) each 5 times.

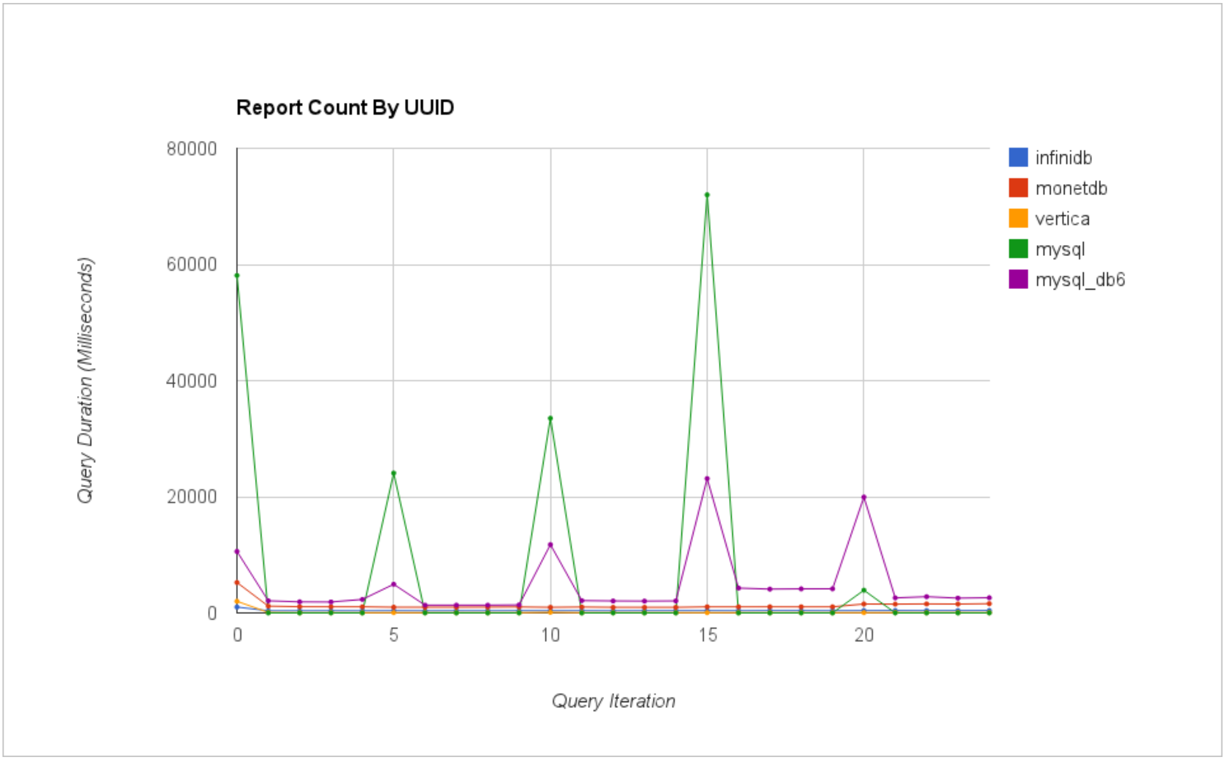

Figure 2.1 - Status Submission Count By UUID

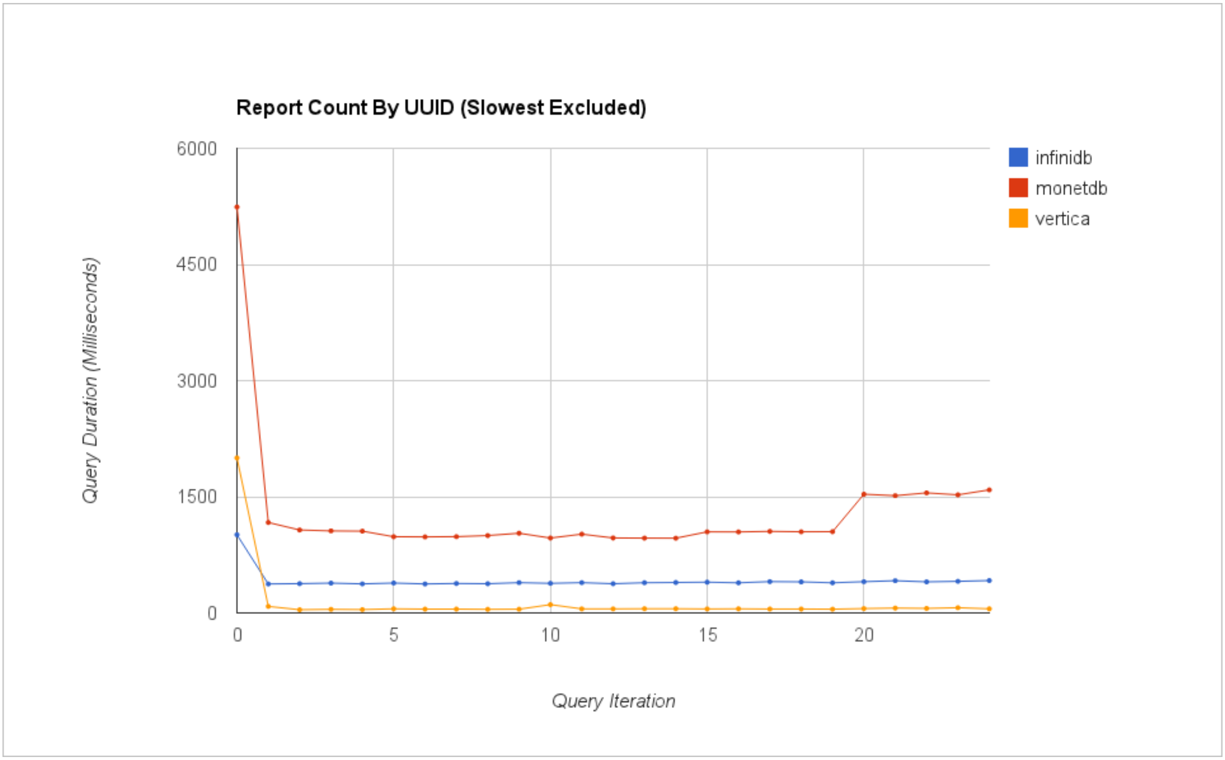

Figure 2.2 - Status Submission Count By UUID (Slowest Excluded)

Explanation

- Figure 2.1 shows the significant query time range spread in this particular test.
- The spike at the beginning is fairly consistent across all DBs, as is expected after a fresh restart.
- MySQL and MySQL_DB6 performed poorly for any query which was not an exact match in the query cache.
- All three column stores performed well, remaining with a near flat performance curve between queries with Vertica being the lowest by a decent margin.

Test 3

Summary:
What is the total number of message submissions on a given day involving a specific SID?

Example SQL Query:
```
SELECT COUNT(*) FROM messages_original.message_submission
WHERE apache_timestamp BETWEEN '2014-01-01' AND '2014-01-02'
AND sid=<sid>;
```

Process:
This query was run with 5 different SIDs (chosen as the 5 most frequent) each 5 times.

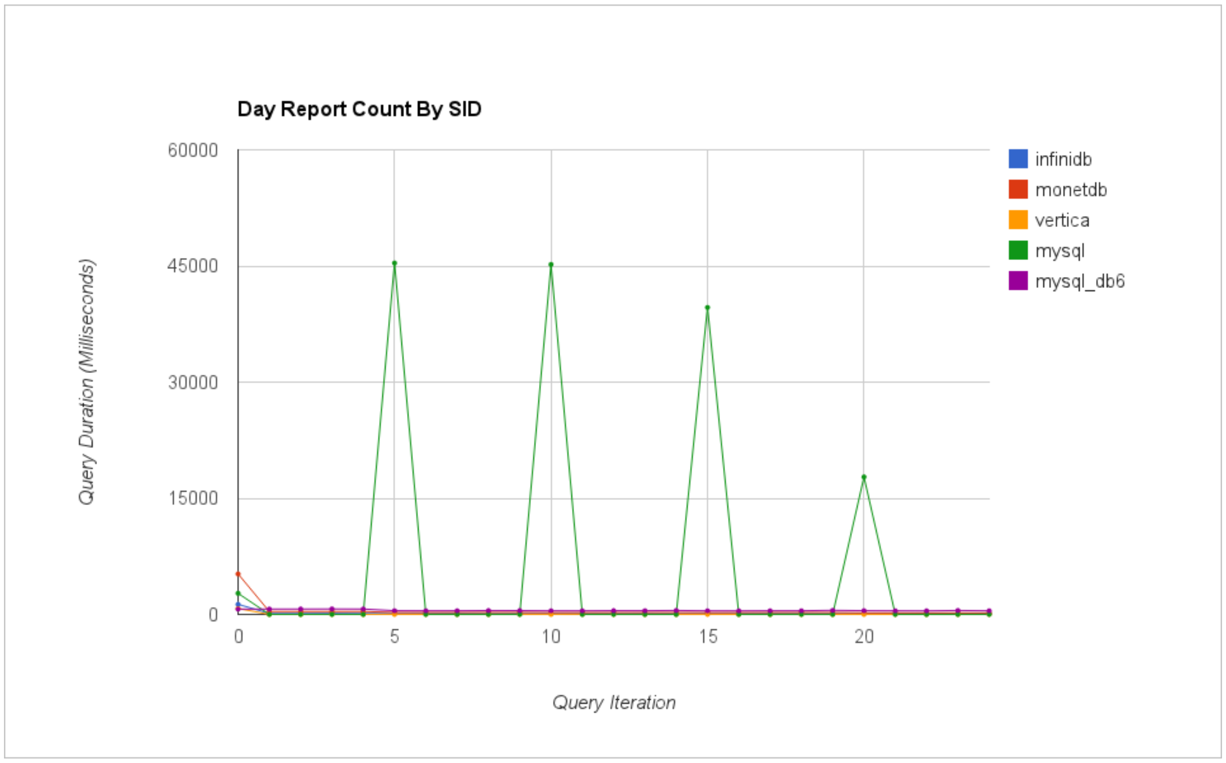

Figure 3.1 - Day Message Submission Count By SID

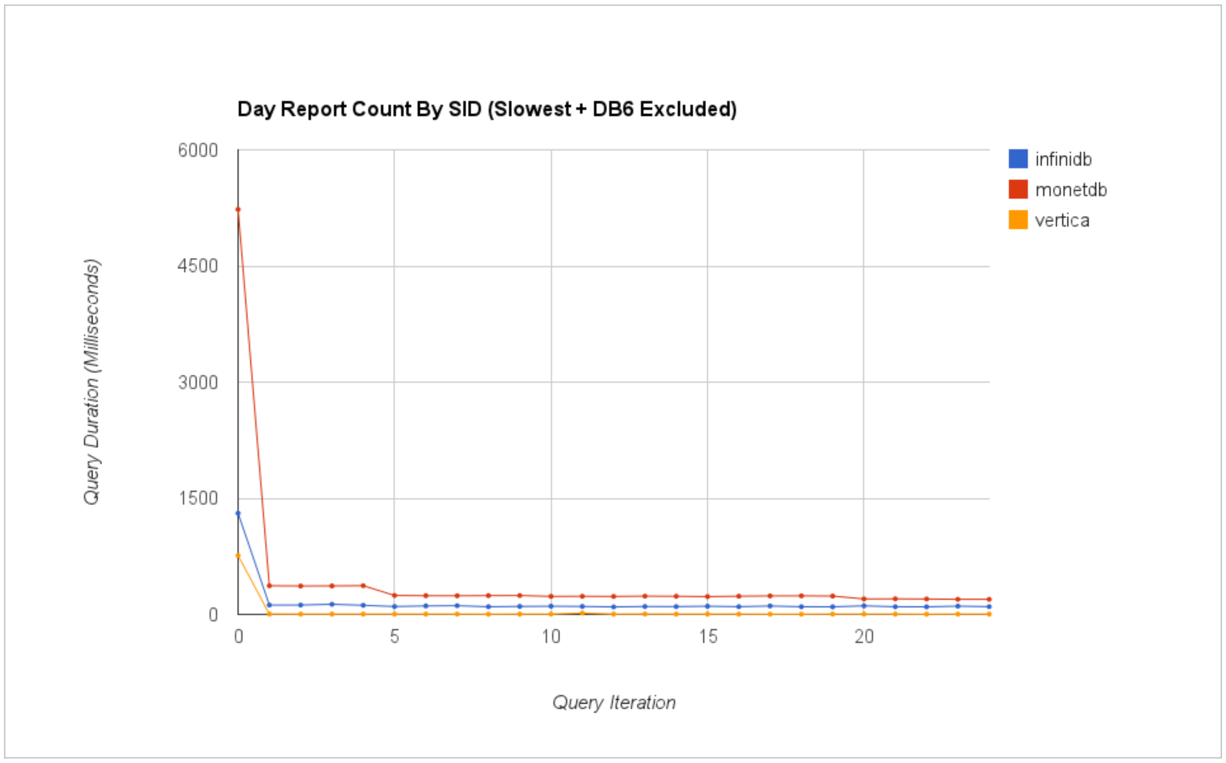

Figure 3.2 - Day Message Submission Count By SID (Slowest + DB6 Excluded)

Explanation

- Figure 3.1 we see MySQL query duration is very high up on it's own with all others including MySQL_DB6 along the bottom axis.
- After some analysis, it was determined that MySQL_DB6 has a composite index for Timestamp+SID, which explains the performance in this test, vs the previous one.

Test 4

Summary:
For a month, rank SIDs by number of occurrences on a granularity of a day.

Example SQL Query:

```
SELECT DATE(apache_timestamp) as logdate,sid,COUNT(sid) as count
FROM messages_original.message_submission
WHERE apache_timestamp BETWEEN '2013-12-16' AND '2014-01-16'
GROUP BY logdate,sid
ORDER BY logdate,count DESC;
```

Process:
As there are no changing parameters for this query, we run it 5 times with no changes.

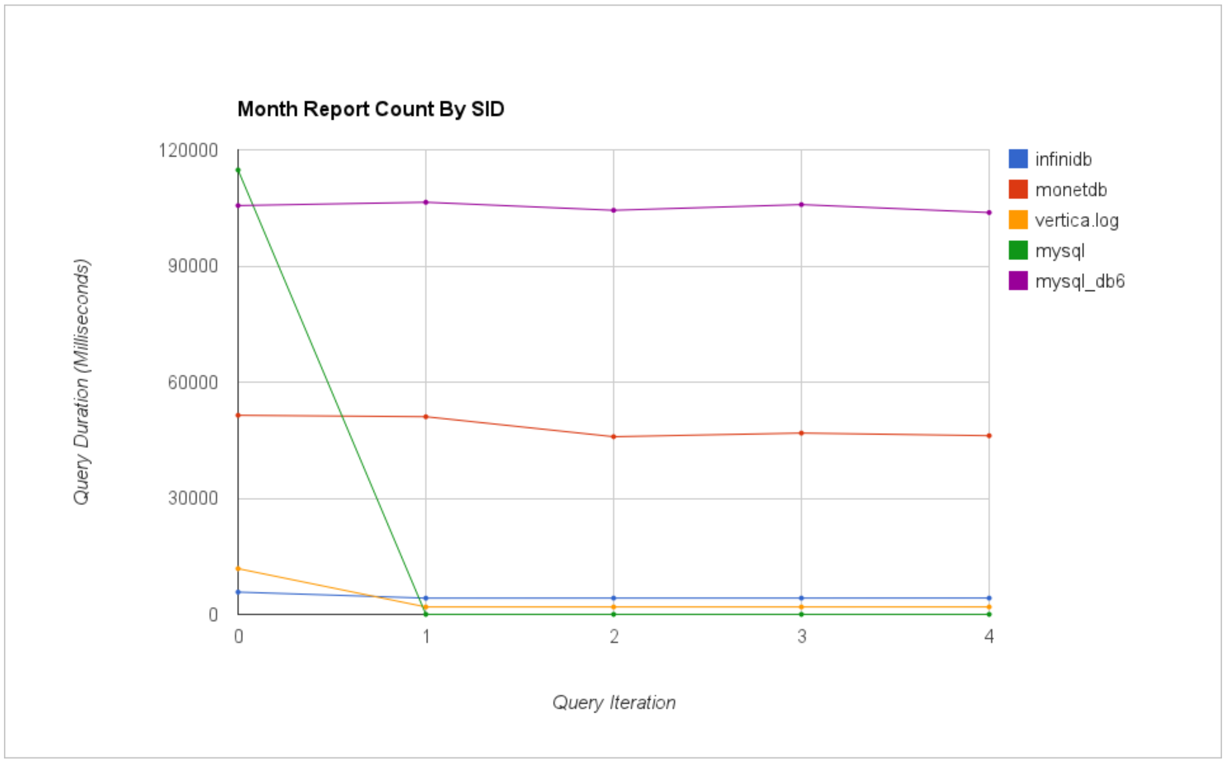

Figure 4.1 - Month Message Submission Count By SID

Explanation

- In Figure 4.1 we see that infiniDB and Vertica vastly outperform all others for this test, including MonetDB

Test 5

Summary:
For a month, rank SIDs by number of occurrences, reported by a specific UUID on a granularity of a day.

Example SQL Query:
```
SELECT DATE(apache_timestamp) as date,sid,COUNT(sid) as count
FROM messages_original.message_submission
WHERE message_submission.device_uuid = UNHEX('<UUID>')
AND apache_timestamp BETWEEN '2013-12-01' AND '2014-01-01'
GROUP BY date,sid
ORDER BY count DESC;
```

Process:
This query was run with 5 different UUIDs (chosen as the 5 most frequent) each 5 times.

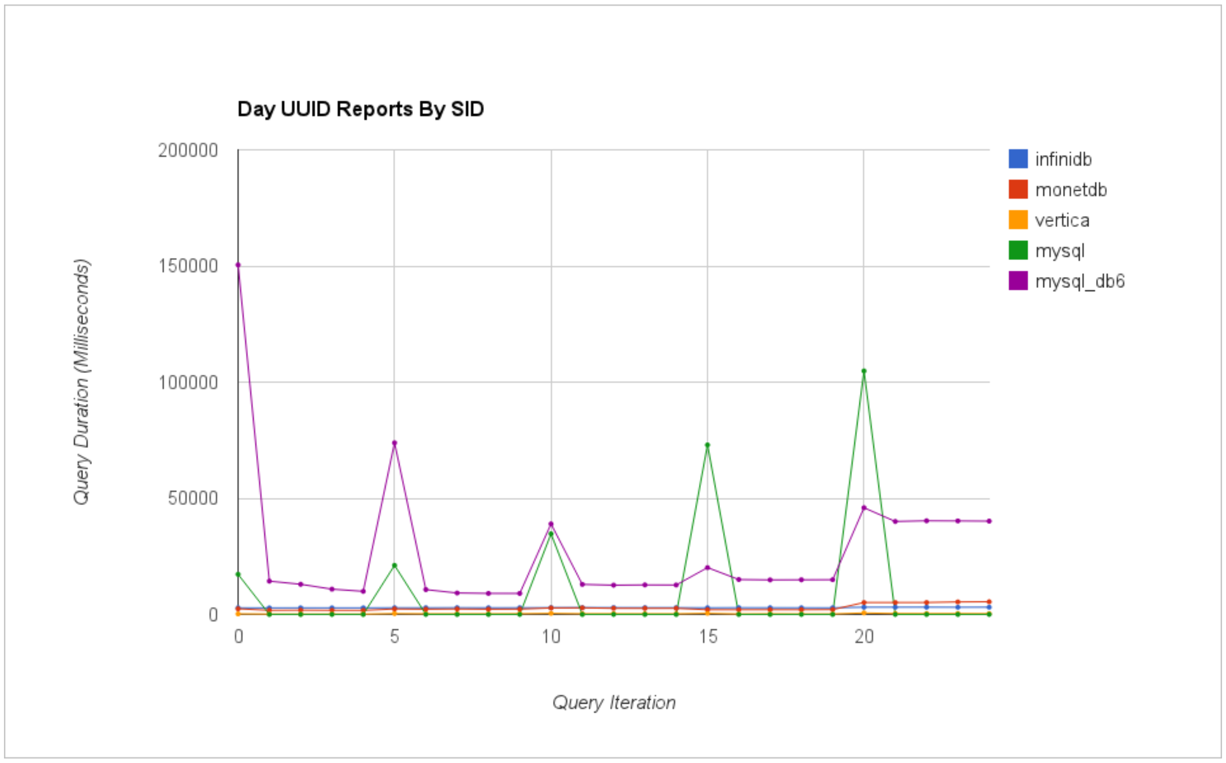

Figure 5.1 - Day UUID Message Submissions By SID

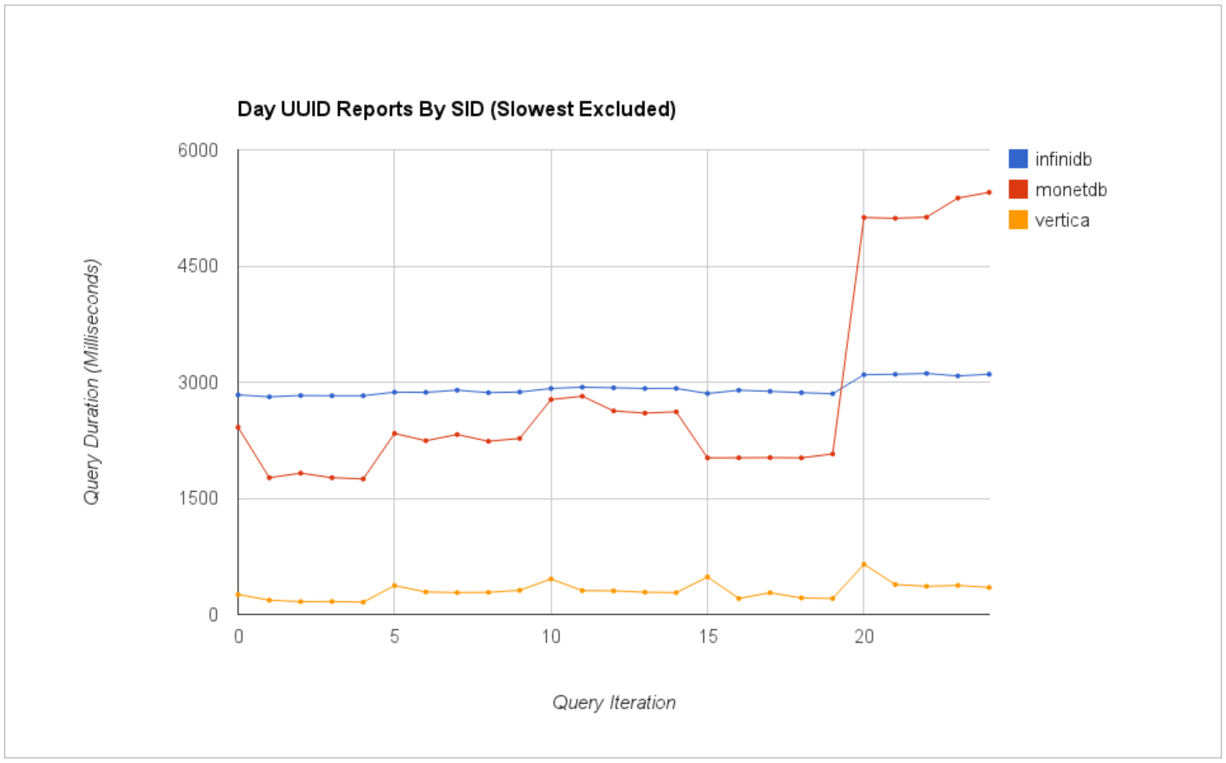

Figure 5.2 - Day UUID Message Submissions By SID (Slowest Excluded)

Explanation

- In Figure 5.1 we see that MySQL and MySQL_DB6 perform nearly two order of magnitude worse than any of the column-stores.
- In Figure 5.2 we can see Vertica outperforms other column-stores by a wide margin.

Test 6

Summary:

For a month, rank SIDs by number of occurrences for each engine_version on a granularity of a day

Example SQL Query:

```
SELECT DATE(message_submission.apache_timestamp) as date,sid,COUNT(*) as
count,engine_version
FROM messages_original.message_submission
JOIN messages_original.status_submission ON
  messages_original.status_submission.device_uuid =
  messages_original.message_submission.device_uuid
WHERE message_submission.apache_timestamp BETWEEN '2013-12-17' AND '2014-01-17'
```

```
GROUP by date, sid, engine_version
ORDER BY engine_version, count DESC
```

Process:
As there are no changing parameters for this query, we run it 5 times with no changes.

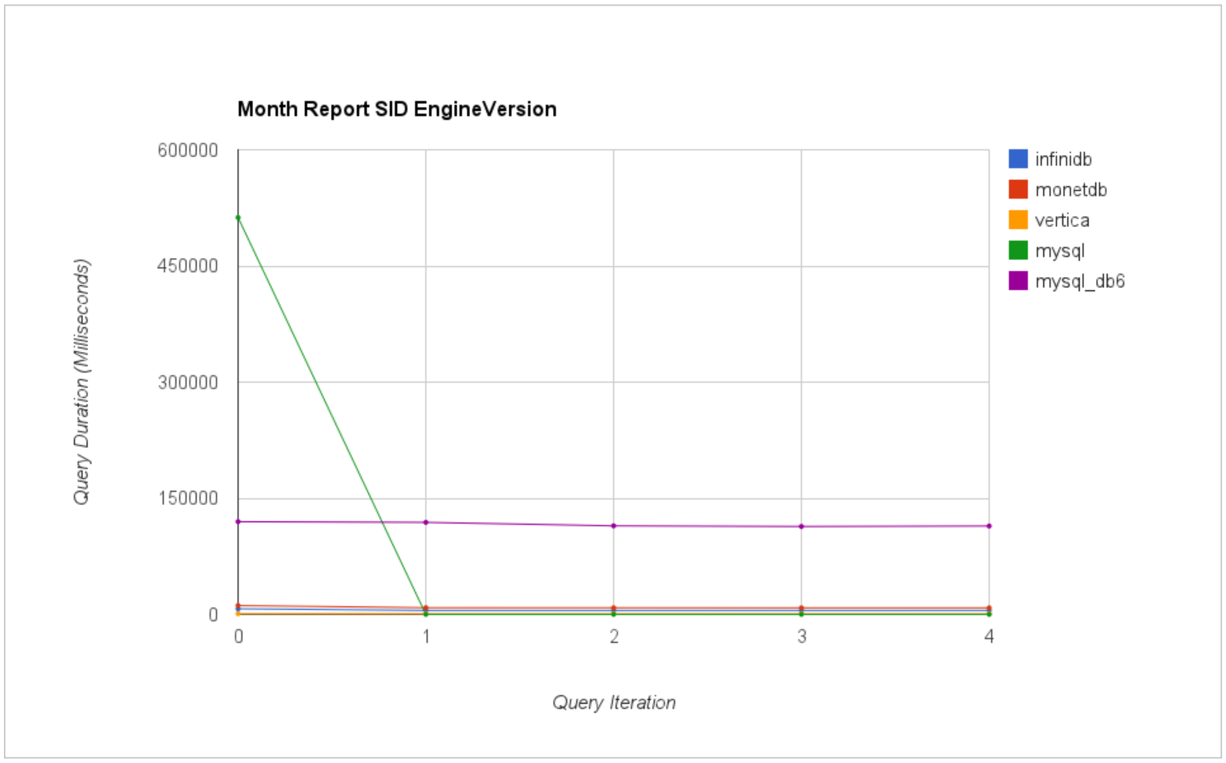

Figure 6.1 - Month Report SID EngineVersion

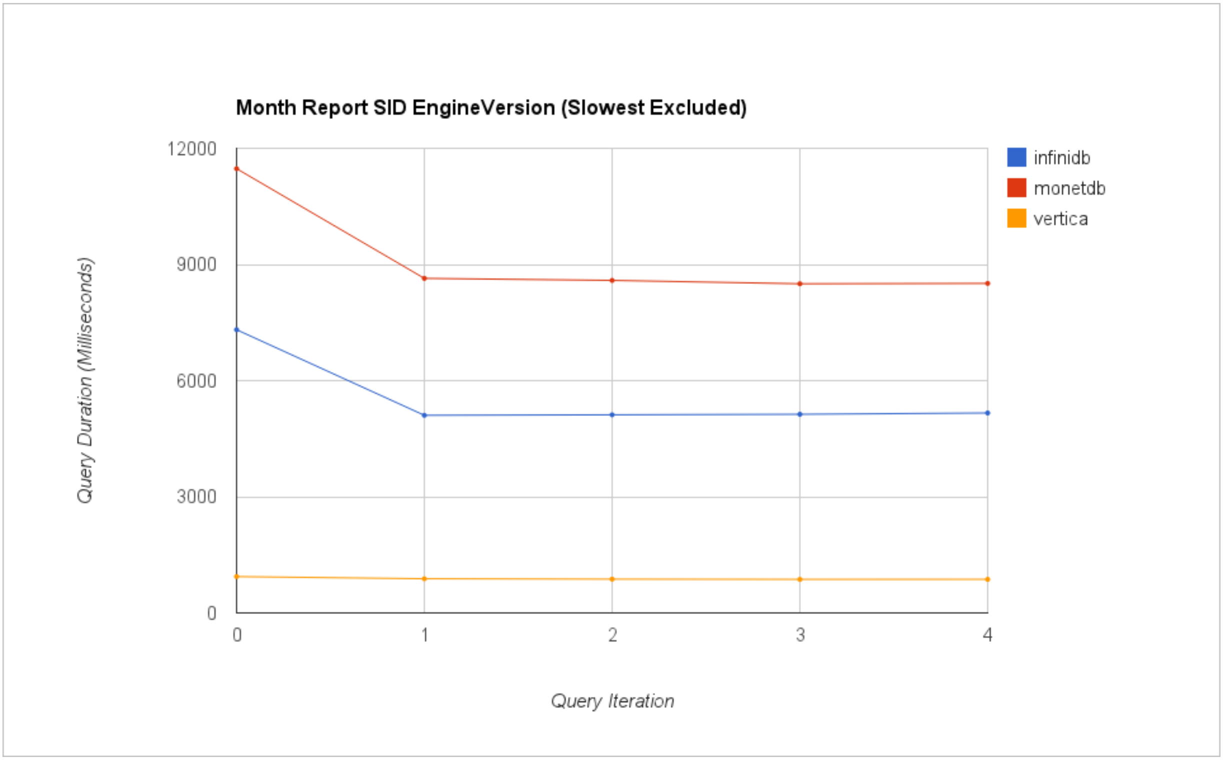

Figure 6.2 - Month Report SID EngineVersion (Slowest Excluded)

Explanation

- In Figure 6.1 we see that MySQL and MySQL_DB6 perform nearly two order of magnitude worse than any of the column-stores.
- In Figure 6.2 we see a fairly linear relationship, with Vertica being the clear winner

Test 7

Summary:

For a month, count the number of unique UUIDs submitting messages, on a day granularity.

Example SQL Query:

```
SELECT DATE(apache_timestamp) as logdate,COUNT(distinct device_uuid) as devices
FROM messagas_original.message_submission
WHERE apache_timestamp BETWEEN '2013-12-16' AND '2014-01-16'
GROUP BY logdate;
```

Process:

As there are no changing parameters for this query, we run it 5 times with no changes.

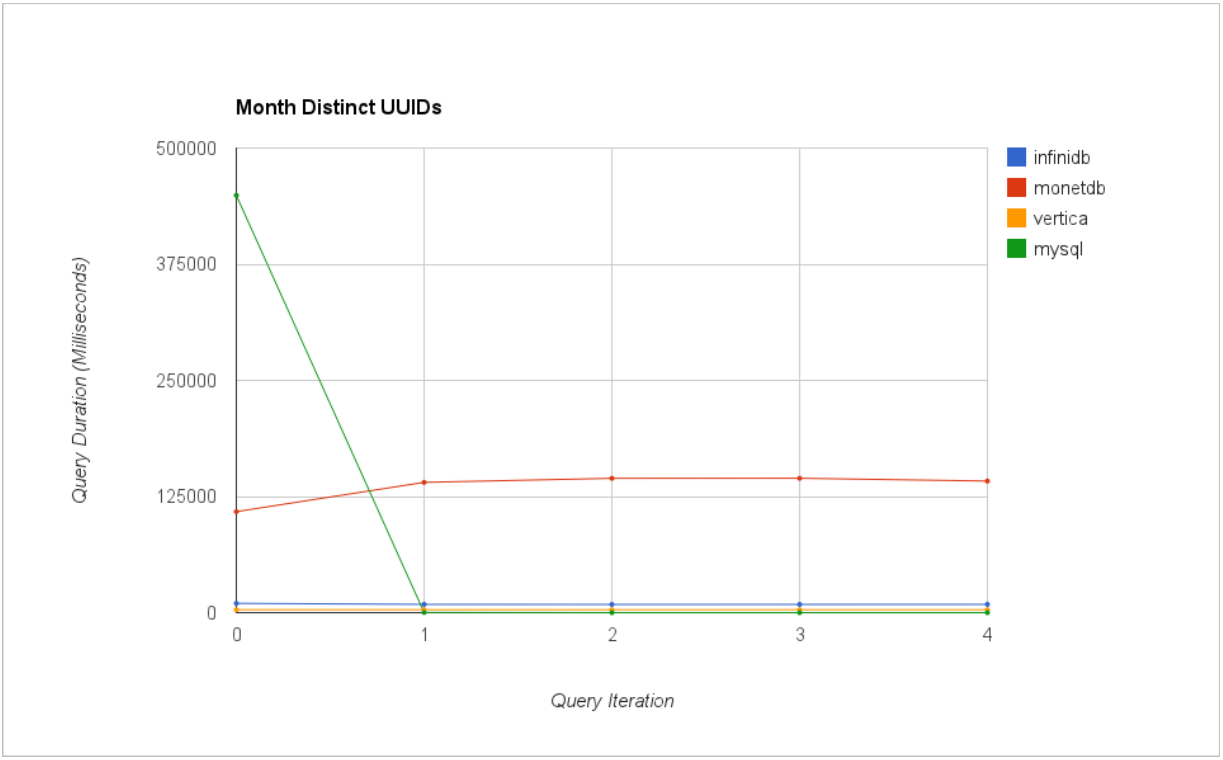

Figure 7.1 - Month Distinct UUIDs

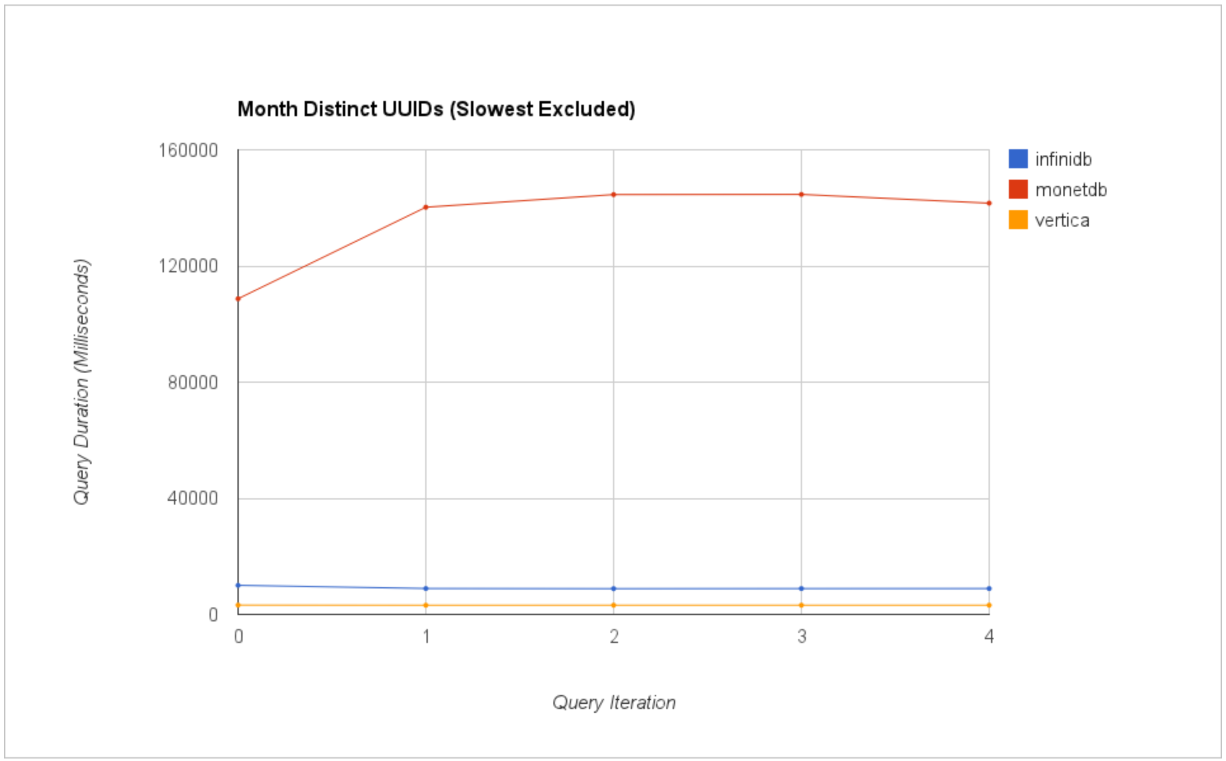

Figure 7.2 - Month Distinct UUIDs (Slowest Excluded)

Explanation

- In Figure 7.1 we see that MonetDB performs better, but on the same scale as MySQL for this test.
- In Figure 7.2 we see again the same pattern of InfiniDB and Vertica performing with similar characteristics, with Vertica having a significant edge.

Test 8

Summary:
For a month, count the number of unique UUIDs sending submissions involving a specific SID, on a day granularity.

Example SQL Query:
```
SELECT DATE(apache_timestamp) as logdate,COUNT(distinct device_uuid) as devices
FROM messages_original.message_submission
WHERE apache_timestamp BETWEEN '2013-12-16' AND '2014-01-22' AND sid=<SID>
GROUP BY logdate;
```

Process:
This query was run with 5 different UUIDs (chosen as the 5 most frequent) each 5 times.

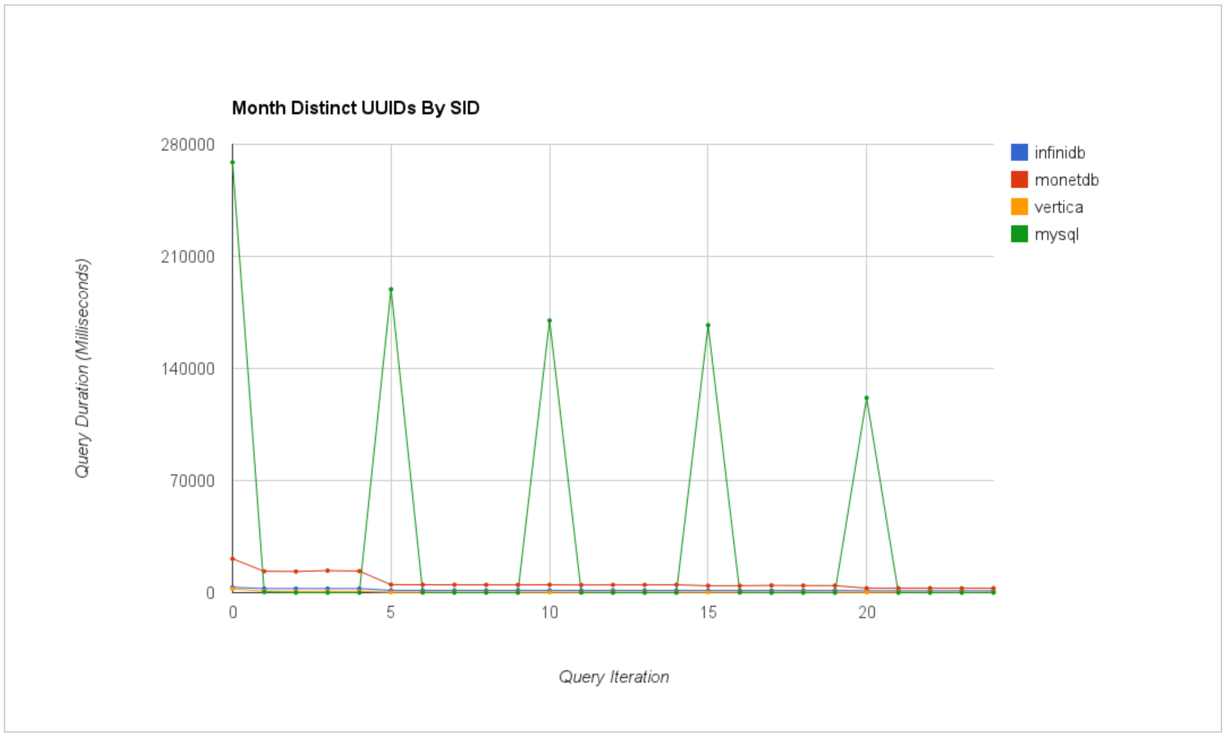

Figure 8.1 - Month Distinct UUIDs By SID

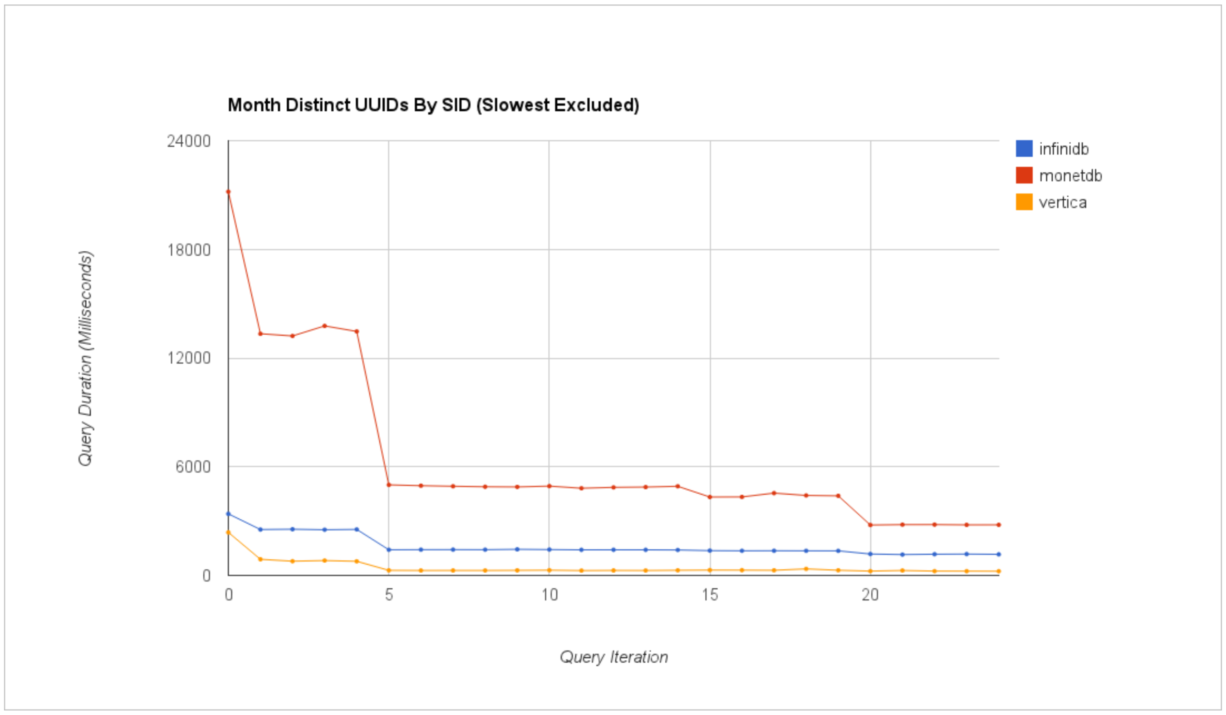

Figure 8.2 - Month Distinct UUIDs By SID (Slowest Excluded)

Explanation

- In Figures 8.1 and 8.2 we see the same patterns as observed in previous tests.

Test 9

Summary:

For a month, show the number of distinct UUIDs submitting messages, and what data version they reported most recently before each individual submission.

Note: This is a tricky one but is a query we use often to determine how rapidly data updates have propagated through our nodes. In addition, Vertica does not support the same SQL subquery in a JOIN ON clause, but instead offers the very useful INTERPOLATE operation, which finds the nearest timestamp for us.

Example SQL Query:
```
SELECT DATE(message_submission.apache_timestamp) as logdate,data_version,COUNT(DISTINCT
message_submission.device_uuid) as uuids
FROM messages_original.message_submission
LEFT OUTER JOIN messages_original.status_submission
    ON status_submission.device_uuid = message_submission.device_uuid
    AND status_submission.apache_timestamp =
        ( SELECT MAX(apache_timestamp) FROM messages_original.status_submission
        WHERE device_uuid=message_submission.device_uuid
        AND apache_timestamp <= message_submission.apache_timestamp )
 WHERE message_submission.apache_timestamp BETWEEN '2013-12-16' AND '2014-01-16'
GROUP BY logdate,data_version ORDER BY logdate,data_version DESC;
```

Example Vertica Query:
```
SELECT DATE(message_submission.apache_timestamp) as logdate,data_version,COUNT(DISTINCT
message_submission.device_uuid) as uuids
FROM message_submission
LEFT OUTER JOIN status_submission
    ON message_submission.device_uuid = status_submission.device_uuid
    AND message_submission.apache_timestamp INTERPOLATE PREVIOUS VALUE
        status_submission.apache_timestamp
WHERE message_submission.apache_timestamp BETWEEN '2013-12-16' AND '2014-01-16'
GROUP BY logdate,data_version ORDER BY logdate,data_version DESC;
```

Process:

As there are no changing parameters for this query, we run it 5 times with no changes.

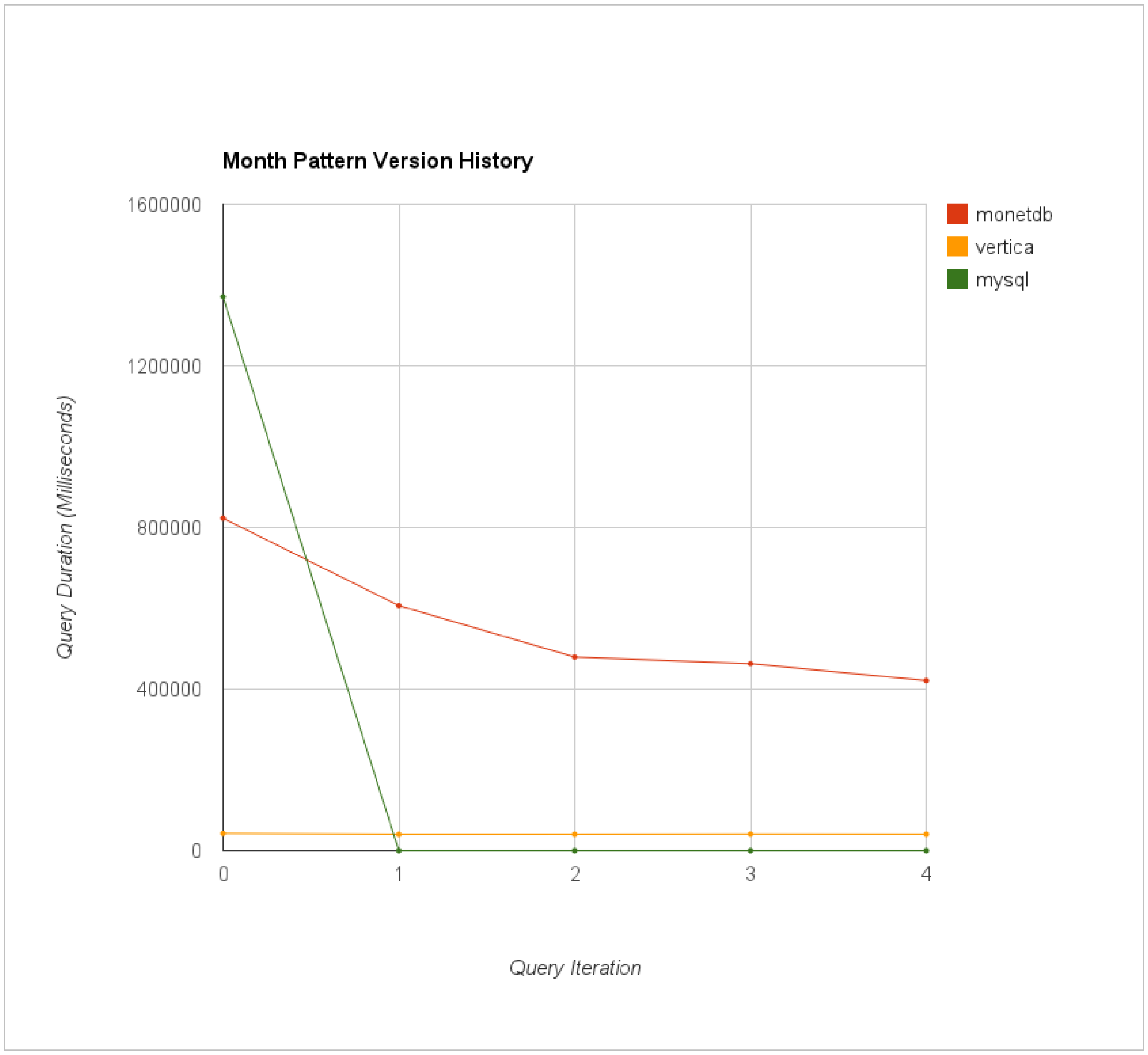

Figure 9.1 - Month Data Version History

Explanation

- In Figures 9.1 we see that MonetDB performs relatively poorly, but linearly improves its performance over time. Vertica performs very well compared to MySQL and MonetDB.
- InnoDB is noticeably vacant, as it did not support this specific query, and we were unable to find support with our free license in order to determine an alternative query.

Data Size

Here we investigate how much storage space on disk each database requires to store information.

MySQL	**MonetDB**	**Vertica**	**InfiniDB**
12.11 GB	6.1 GB	0.94 GB	4.9 GB

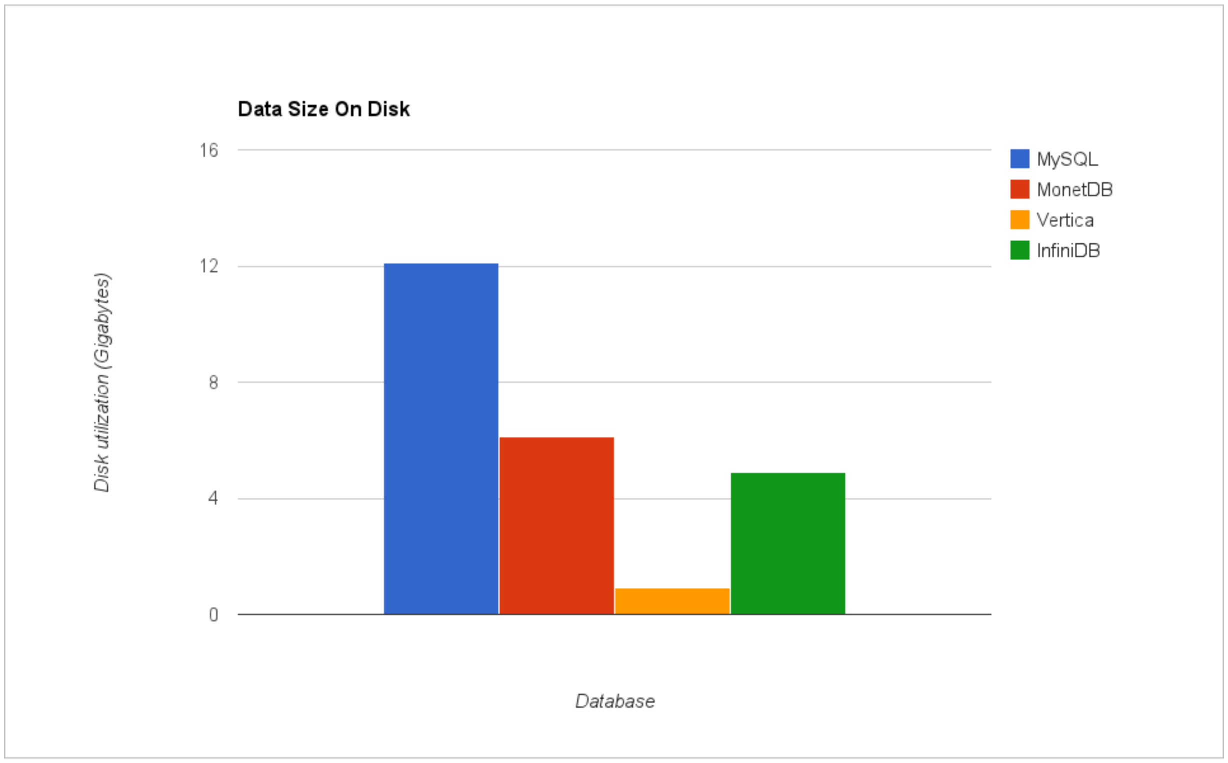

Figure 10.1 - Data Size on Disk

Explanation

- Column stores tend to compress their indexes very aggressively to try and achieve faster lookups. This is apparent in Figure 10.1, with Vertica seemingly the most aggressive.

Conclusions

In summary, all tested column-store databases performed better than MySQL. MonetDB performed better than MySQL in most cases but as queries became more complex the difference between them decreased by an order of magnitude. In all cases InfiniDB and Vertica significantly outperformed MySQL and MonetDB. Vertica consistently outperformed InfiniDB.

Our tests show that for our analytical use case, any of the tested database technologies would improve our system performance and ability to mine information from our existing data over our existing MySQL-based infrastructure. Vertica in particular stands out as a great performer, and has a extremely good track record for scalability based on industry publications as does InfiniDB.

We expect as a result of faster queries and lower disk storage requirements, that less maintenance will be required on our part, more responsive applications will be possible, and better overarching decisions and analytics will be achievable using a tool like Vertica.

InfiniDB has a similar set of pros associated with is, however performance in our tests was not quite as good as Vertica, and support we found to be lacking in comparison.

MonetDB was less competitive than the other two column stores, even though more performant than MySQL in our tests. The community and documentation was lacking and we opted not to further our work with it.

Sources

Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., & Stonebraker, M. (2009, June). A comparison of approaches to large-scale data analysis. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (pp. 165-178). ACM.

Idreos, S., Groffen, F., Nes, N., Manegold, S., Mullender, K. S., & Kersten, M. (2012). MonetDB: Two decades of research in column-oriented database architectures. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 35, 40-45.

Abadi, D., Madden, S., & Ferreira, M. (2006, June). Integrating compression and execution in column-oriented database systems. In Proceedings of the 2006 ACM SIGMOD international conference on Management of data (pp. 671-682). ACM.

Ślęzak, D., Wróblewski, J., Eastwood, V., & Synak, P. (2008). Brighthouse: an analytic data warehouse for ad-hoc queries. Proceedings of the VLDB Endowment, 1(2), 1337-1345.

(2012). Columnstore Indexes. Retrieved from http://technet.microsoft.com/en-us/library/gg492088.aspx

Abadi, D. J., Madden, S. R., & Hachem, N. (2008, June). Column-Stores vs. Row-Stores: How different are they really?. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (pp. 967-980). ACM.

Stonebraker, M., Abadi, D. J., Batkin, A., Chen, X., Cherniack, M., Ferreira, M., ... & Zdonik, S. (2005, August). C-store: a column-oriented DBMS. InProceedings of the 31st international conference on Very large data bases (pp. 553-564). VLDB Endowment.

Plattner, H. (2009, June). A common database approach for OLTP and OLAP using an in-memory column database. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (pp. 1-2). ACM.

Abadi, D. J. (2007, January). Column Stores for Wide and Sparse Data. In CIDR(pp. 292-297).