Flow Storage Revisited: Is it time to re-architect flow storage and processing systems?

FloCon 2015

John M^c^Hugh

REDJACK

Overview

- The Landscape
- Redshift Structure and Economics
- Very Preliminary Results
- Inventories and other ideas
- Discussion & Future Work

REDJACK

The Landscape

- Early in the history of SiLK, we experimented with relational databases as a possible repository.
  - Storage requirements went up by a factor of 10+
  - Response time and cost went up, as well.
  - Looked like a lose / lose / lose proposition.
- The flat hourly files and Unix like command line tools looked pretty good in comparison. As needs evolved, we rolled our own parallel architectures.
- Lots of things have changed since the early 2000s. Perhaps it is time to revisit some of our earlier decisions. They were definitely right then. Are they now?

The landscape (cloudscape?)

- It started when it became possible to rent a server in a lights out center (no one is home) for a $’small / month.
- These started out as physical machines, but were soon virtual boxes (repeating the mainframe -> time sharing -> mini -> timesharing -> micro -> cycle yet once again)
- Now cloud providers such as Amazon (a money losing bookseller located North of here) offer a wide variety of computing services off in never never land.
- One, Redshift, is a data warehousing service.
  - Conceptually, Redshift is a lot like Centaur
  - Even if it doesn’t do flow well, it might teach us things that would make Centaur [more] usable.

Redshift is not quite a full RDBMS

- Warehouses are not factories.
  - You store stuff there, you don't transform it.
  - More like a roach motel. Deletions can be hard.
  - Fits with a never throw anything away mentality
- Fits the store once, query many times model that should characterize analysis.
- Can support single or multiple tables, but, if multiple tables have the same structure, there are advantages to combining them into a common view.
  - Sort of like what you get when you say to rwfilter
    - --flowtype = all/all

REDJACK

Blocks, Columns, the Index, and Nodes

- Redshift stores its data in blocks of a MB
  - reads whole blocks and processes them
- Within a block, data from a column is stored together
  - can get really good compression
  - can ignore columns not in query
  - select * is not a good thing to do (ask Sensage)
- Data is stored in a single sort order, blocks are ordered.
  - We know the min – max key for block (can ignore)
- Data is distributed across nodes (1-32 or 2-128)
  - Provides parallel query execution.

Economics (TANSTAAFL)

- It is easy (and cheap) to get started
  - dw2.large node (160GB SSD) $0.25/hour on demand
  - dw1.8xlarge node (16TB disk) $6.80/hour on demand
  - no cost when you shut down (but snapshot storage)
  - For production costs, the dw1s are $1K/year/TB on a 3 year contract. $2.2k/year/TB on a 1 year contract.
  - The upper limit with the dw1 is 2 PB on 128 nodes. That would be $2M/year.
- We are currently experimenting with 1, 4, and 16 node dw2.large clusters.

Table Design

- Flat table, (like Centaur) Time series of tables 1 / month
- Fields: starthour, {s,d}IP, Prot, {s,e}time, duration,{s,d}port, icmp_{type,code},{syn-zun}_flag_{init,ses} packets, bytes, {t,c,s,f}_flag, stime_rev, prot_rev, stime_{parent,child}, sid
  - 3 times avoid calculation in queries
  - All 12 possible TCP flags included (twice)
  - Parent / child allow continuation stitching
  - Flow matching with rev_ (including ICMP cases in sensor
  - Sort key is (starthour, sip, protocol, stime)
- Use of starthour allows hourly appends of sorted data.

Redshift Interface

- We are using SQL Workbench to interact with Redshift.
  - Queries are formatted on the local workstation (MacBook Pro) and sent over the network to the Redshift cluster
  - Results are returned to the workbench and presented on the screen.
- There is some evidence, based of differences in query timings reported by the Workbench and Redshift that returning large amounts of data to the Workbench affects the reported timing at the workbench (used here).
  - This requires further evaluation.
  - In general, queries that produce large results should target a table or file, not a display.

Initial results

- We have about 6 months of data to/from 208.78.4.0/22
  - Single dw2.large node – 82,548,396 rows.
  - 6 monthly tables joined by a UNION ALL view
    - Not much, but provides a sanity check.
- select * from yaf_all where protocol = 50 and (continued_flag = TRUE or continuation_flag = TRUE);
  - 32502 rows in 48.78 seconds. 42.08, 41.11, 42.22
- select sip, dip, stime, duration, packets, bytes from yaf_all where protocol = 50 and (continued_flag = TRUE or continuation_flag = TRUE);
  - 32502 rows in 16.56 seconds. 10.01, 8.21, 8.30
- SiLK gets 32502 rows in 81.39 seconds. 16.28, 16.53, 16.42 on what is probably a faster machine.

So what? 80M records is a toy example

- True, but it does provide a sanity check and tells us:
  - select * is not desirable in a column store
    - To reconstruct the row, you have to index into every column, possibly decompressing on the fly.
    - This may lead to different ways of constructing analyses
      - You don’t need the ports or TCP flags here.
  - Data uses about 5% of the disk on a 1 node instance
  - This query does not really use the index.
- Getting more real data in a hurry is not possible, so we built a data multiplier.
  - Have created a 20X set (1.6B records) and a 400X set (33B records)

The Data Multiplier

REDJACK

- Records are CSV. 1 record in gives 2 to 1000 out
  - Initially in a scripting language – rewrote in C
- Original record is included in the output
- All fields in replicates are preserved, as is, except:
  - addresses in 208.78.4.0/22 are translated into a series of /22s starting at 230.0.0.0/22 – no addresses from this region appear in the original data.
  - the last 3 digits of the microseconds in the timestamps get the replica number 001 … 999
- Any query against multiplied data that involves an address in the 208.78.4.0/22 block will produce the same result as with the original data.
  - More data has to be processed to find it.

A misstep at 20X

- We now have 1.6 B records rather than 80M
- The protocol=50 and continuation flags query that ran in about 10 seconds now takes about 150.
  - Returns 650,040 records rather than 32,502
  - This is slightly better than 20X the original time, but nowhere near the 5X that parallelism would predict.
  - Not completely sure what is going on, but suspect that rendering the values and transmitting the data is a major contributor.
  - In any event, this is not what we intended to ask.

20X results (much better)

- 4 nodes, 20X data - Linear scaling would be 5X time
- Original results now buried in the noise records so add the address ranges of the endpoints
  - select sip, dip, stime, duration, packets, bytes from yaf_all where ((sip > 3494773759 and sip < 3494774780) or (dip > 3494773759 and dip < 3494774780)) and protocol = 50 and (continued_flag = TRUE or continuation_flag = TRUE);
    - Redshift does not have an IP address type.
  - 32502 rows in 28.93 sec (27.16, 26.9, 26.54)
    - less than 3X the single node time!!
    - The single node may not be a good base case as much of the computational approach assumes parallelism

400X results (Better yet)

- 16 nodes, 400X data - Linear scaling would be 25X base time or 5X the 20X time.
  - Use same query as in the 20X case
  - 32502 rows in 143sec (124, 121, 124, 120)
    - about 4.5X the 20X time. (little better than linear scaling)
  - 33B records uses ~50% of 2.5TB disk available
    - Required about 6 hours to load data in 1 month chunks
      - Did 1 month as daily loads 1-2 min/day
      - Hourly loads would probably go in seconds
    - Saves go in seconds (based on incremental?)
    - Restore takes about an hour

An ipset query at 400X

REDJACK

- IP sets are commonly used in SiLK queries. SQL lacks explicit support for these unless …
  - A set is just a table with a single column
    - by extension, a multiset is just a table with multiple columns
  - A JOIN operation can match fields in one table with those in another. In general, this is expensive, but
    - if both tables are sorted this is a single pass operation.
    - preselection can help.
- Example
  1. Find external users of protocol 50 (used for VPN)
  2. Create subset of interest
  3. Find all protocol 50 traffic outbound
  4. Join with subset of interest to find their outbound flows

Finding the subset of interest

- Find the set of external users of protocol 50
  - select distinct sip into table vpn_users from yaf_all where (dip > 3494773759 and dip < 3494774780 and protocol = 50) group by 1 order by 1;
  - This gives us 12 addresses in about 59 seconds
- The subset of interest is those addresses above 77.0.0.0
  - select * into vpn77plus from vpn_users where sip > 1291845632 order by 1; giving a set of 7 addresses.
  - This gives us 7 addresses in a fraction of a second

VPN traffic and result

REDJACK

- The protocol 50 (VPN) outbound traffic
  - select sip, dip, stime, duration, packets, bytes into table vpn_candidates from yaf_all where protocol = 50 and sip > 3494773759 and sip < 3494774780 order by sip;
  - 16261 rows in 2min 5 sec
- The portion of the traffic to the addresses of interest
  - select vpn_candidates.sip, vpn_candidates.dip, vpn_candidates.stime, vpn_candidates.duration, vpn_candidates.packets, vpn_candidates.bytes from vpn77plus inner join vpn_candidates on vpn77plus.sip = vpn_candidates.dip order by stime, dip, sip;
  - Gives us 104 records. in 0.63 sec

Set query discussion

- 2 min+ to locate 104 specific records in 33B
  - Another minute for preliminaries
- But, expanding the set is slightly faster
  - select sip, dip, stime, duration, packets, bytes from yaf_all where protocol = 50 and ((dip = 1625703938) or … or (dip = 3428060146)) and sip > 3494773759 and sip < 3494774780 order by stime, sip, dip;
  - Returns the same 104 rows in 1 min 39 sec
- We can do sets as joins or expansions, CIDR blocks as ranges. Finding the best ways to formulate common analytic queries as SQL will require experimentation
  - Results will change as Redshift evolves

General Discussion

REDJACK

- The index does not seem to be an effective query time reduction mechanism
  - Sample queries not time limited.
    - Still, brute force scans seem to be fairly effective.
      - A few seconds / billion records.
  - Primary key as hour probably limits utility of others
  - Volume summary (f, p, b) by hour for Sept.(7.9B rows) takes ~80 sec.
    - Restricting to /22 of interest takes ~8 sec for 7.3M rows
    - Including protocol 50 drops to ~7sec for 2.5K rows
    - Using /22 as dip (not indexed) gives ~12 sec for 12M rows
  - Additional work needed to sort this out.
    - Performance may be a function of columns inspected and columns extracted on a hit.

Inventories and other ideas

REDJACK

- The single index requires a choice after the start hour
  - index by sIP and the sources are easy within an hour
    - but the dIPs are more or less randomly spread about
  - index by dIP and the opposite is true
- What if we maintained a monthly inventory indexed by IP by month with 744 columns, one for each hour in the month. Similar inventories for ports / protocols trivial.
  - A query for the key gets all the activity for the month
    - encode src/dst, tcp/udp/icmp/other volume categories for flows, packets, bytes. All of this will fit in 1 64 bit number.
    - current month has active updates – regular DBMS problem
    - past months are a warehouse problem.
- This is one of many support processes that can reduce query loads. System design effort is needed.

The Politics

- Customer data in the cloud?
  - in most cases, absolutely. Many are using it already.
- Loss of control over the environment, security concerns, etc.
  - see above. In many cases, making maintenance, service, etc. someone else's problem is a good trade of money for relief.
- The “Not invented here.” problem.
  - Sometimes, you have to learn to live with others decisions, which brings us to …

REDJACK

Technical issues

- The interface sucks. It is almost, but not quite, exactly unlike postgressql. Everything you want is in the unsupported list.
  - At the very least, this means an analyst friendly front end that translates things like IP addresses, TCP flags, etc., going both ways.
  - Input and output data are effectively text. Something in the collection to Redshift chain needs to produce the right kind of text.
  - On the plus side, many SiLK style reports can be generated with a single query
    - I'm not an SQL expert, but I can see where this is going.

Limitations

- 9900 tables seems like a lot, but if you did a table per hour per sensor per flowtype you'd run out pretty quickly.
- Data is sorted on input (probably accounted for my disk explosion on monthly input). You can add to the end of a table at no cost, but, if inserting replaces records, you pay to clean up the table (vacuuming).
- Timestamp keys are recommended as are time series of tables since you can simply drop a stale table from one end of the series and add a new one at the other. replacing the view is trivial.
- The single index means that some fields will require brute force search. Hourly keys serve to limit searches as hourly files do in Centaur.

REDJACK

Conclusions and next steps (1)

- Redshift may be a viable way to store flow data. The current evidence points that way.
  - We need to collect representative queries from practicing analysts. This is sensitive, but doable. More interested in features used than values; sizes of lists and sets useful, date / time ranges, 3 days, 2.5 hours, not specific dates. Use of IP sets, wildcards, CIDRs, lists, etc.
  - If this approach is viable, the overhead involved in setting up a repository and keeping it going is greatly reduced. Both small operators and researchers benefit.

Conclusions and next steps (2)

- Even if redshift is not the way to go, some of its approaches could be adopted to better Centaur.
  - Initial sorting (use most common search key)
  - Uniform distribution (maximize effects of parallelism)
  - Some levels of indexing to minimize futile searches
- Columnar store seems inappropriate for flow, but looking at representations for more efficient compression might produce a win.
- Replicating the data for the most recent months could improve performance for the most active query regions
  - Taper replication as usage falls off

REDJACK

Questions?

Contact Information

John McHugh
john dot mchugh at redjack dot com