Cold Storage that isn't glacial

Joshua Hollander, Principal Software Engineer

ProtectWise

Who are we and what do we do?

Who are we and what do we do?

We record full fidelity Network Data

Who are we and what do we do?

We record full fidelity Network Data

- Every network conversation (NetFlow)

Who are we and what do we do?

We record full fidelity Network Data

- Every network conversation (NetFlow)
- Full fidelity packet data (PCAP)

Who are we and what do we do?

We record full fidelity Network Data

- Every network conversation (NetFlow)
- Full fidelity packet data (PCAP)
- Searchable detailed protocol data for:
    - HTTP
    - DNS
    - DHCP
    - Files
    - Security Events (IDS)
    - and more...

Who are we and what do we do?

We record full fidelity Network Data

- Every network conversation (NetFlow)
- Full fidelity packet data (PCAP)
- Searchable detailed protocol data for:
  - HTTP
  - DNS
  - DHCP
  - Files
  - Security Events (IDS)
  - and more...

We analyze all that data and detect threats others can't see

Network data piles up fast!

Network data piles up fast!

- Over 300 C* servers in production
- Over 1200 servers in EC2
- Over 150TB in C*
- About 90TB of SOLR indexes
- 100TB of cold storage data
- 2 PB of PCAP

Network data piles up fast!

- Over 300 C* servers in production
- Over 1200 servers in EC2
- Over 150TB in C*
- About 90TB of SOLR indexes
- 100TB of cold storage data
- 2 PB of PCAP

And we are growing rapidly!

So what?

So what?

All those servers cost a lot of money

Right sizing our AWS bill

Right sizing our AWS bill

This is all time series data:

- Lots of writes/reads to recent data
- Some reads and very few writes to older data

Right sizing our AWS bill

This is all time series data:

- Lots of writes/reads to recent data
- Some reads and very few writes to older data

So just move all that older, *cold data*, to cheaper storage.

> How hard could that possibly be?

Problem 1:

Data distributed evenly across all these expensive servers

- Using Size Tiered Compaction
- Can't just migrate old SSTables to new cheap servers

Problem 1:

Data distributed evenly across all these expensive servers

- Using Size Tiered Compaction
- Can't just migrate old SSTables to new cheap servers

Solution: ***use Date Tiered Compaction?***

- We update old data regularly
- What SSTables can you safely migrate?

Result:

Problem 1:

Data distributed evenly across all these expensive servers

- Using Size Tiered Compaction
- Can't just migrate old SSTables to new cheap servers

Solution: ***use Date Tiered Compaction?***

- We update old data regularly
- What SSTables can you safely migrate?

Result:

Problem 2: DSE/SOLR re-index takes *forever*!

We have nodes with up to 300GB of SOLR indexes

- Bootstraping a new node requires re-index after streaming
- Re-index can take a ***week or more***!!!
- At that pace we simply cannot bootstrap new nodes

Problem 2: DSE/SOLR re-index takes *forever*!

We have nodes with up to 300GB of SOLR indexes

- Bootstraping a new node requires re-index after streaming
- Re-index can take a ***week or more***!!!
- At that pace we simply cannot bootstrap new nodes

Solution: ***Time sharded clusters in application code***

- Fanout searches for large time windows
- Assemble results in code (using same algorithm SOLR does)

Problem 2: DSE/SOLR re-index takes *forever*!

We have nodes with up to 300GB of SOLR indexes

- Bootstraping a new node requires re-index after streaming
- Re-index can take a ***week or more***!!!
- At that pace we simply cannot bootstrap new nodes

Solution: ***Time sharded clusters in application code***

- Fanout searches for large time windows
- Assemble results in code (using same algorithm SOLR does)

Result:

¯_(ツ)_/¯

What did we gain?

What did we gain?

We can now migrate older timeshards to cheaper servers!

What did we gain?

We can now migrate older timeshards to cheaper servers!

However:

- Cold data servers are still too expensive
- DevOps time suck is massive
- Product wants us to store even more data!

Throwing ideas at the wall

Throwing ideas at the wall

We have this giant data warehouse, let's use that

- Response time is too slow: 10 seconds to pull single record by ID!
- Complex ETL pipeline where latency measured in hours
- Data model is different
- Read only
- Reliability, etc, etc

Throwing ideas at the wall

We have this giant data warehouse, let's use that

- Response time is too slow: 10 seconds to pull single record by ID!
- Complex ETL pipeline where latency measured in hours
- Data model is different
- Read only
- Reliability, etc, etc

What about Elastic Search, HBase, Hive/Parquet, MongoDB, etc, etc?

Throwing ideas at the wall

We have this giant data warehouse, let's use that

- Response time is too slow: 10 seconds to pull single record by ID!
- Complex ETL pipeline where latency measured in hours
- Data model is different
- Read only
- Reliability, etc, etc

What about Elastic Search, HBase, Hive/Parquet, MongoDB, etc, etc?

Wait. Hold on! This Parquet thing is interesting...

Parquet

Parquet

Columnar

- Projections are very efficient
- Enables vectorized execution

Parquet

Columnar

- Projections are very efficient
- Enables vectorized execution

Compressed

- Using Run Length Encoding
- Throw Snappy, LZO, or LZ4 on top of that

Parquet

Columnar

- Projections are very efficient
- Enables vectorized execution

Compressed

- Using Run Length Encoding
- Throw Snappy, LZO, or LZ4 on top of that

Schema

- Files encode schema and other meta-data
- Support exists for merging disparate schema amongst files

Parquet: Some details

Row Group

- Horizontal grouping of columns
- Within each row group data is arranged by column in chunks

Column Chunk

- Chunk of data for an individual column
- Unit of parallelization for I/O

Page

- Column chunks are divided up into pages for compression and encoding

So you have a nice file format…

Now what?

Need to get data out of Cassandra

Need to get data out of Cassandra

Spark seems good for that

Need to get data out of Cassandra

Spark seems good for that

Need to put the data somewhere

Need to get data out of Cassandra

Spark seems good for that

Need to put the data somewhere

S3 is really cheap and fairly well supported by Spark

Need to get data out of Cassandra

Spark seems good for that

Need to put the data somewhere

S3 is really cheap and fairly well supported by Spark

Need to be able to query the data

Need to get data out of Cassandra

Spark seems good for that

Need to put the data somewhere

S3 is really cheap and fairly well supported by Spark

Need to be able to query the data

Spark seems good for that too

So we are using Spark and S3...

Now what?

Lots and lots of files

Sizing parquet

Lots and lots of files

Sizing parquet

- Parquet docs recommend 1GB files for HDFS

Lots and lots of files

Sizing parquet

- Parquet docs recommend 1GB files for HDFS
- For S3 the sweet spot appears to be 128 to 256MB

Lots and lots of files

Sizing parquet

- Parquet docs recommend 1GB files for HDFS
- For S3 the sweet spot appears to be 128 to 256MB

We have terabytes of files

Scans take *forever*!

Lots and lots of files

Sizing parquet

- Parquet docs recommend 1GB files for HDFS
- For S3 the sweet spot appears to be 128 to 256MB

We have terabytes of files

Scans take *forever*!

Partitioning

Our queries are always filtered by:

1. Customer
2. Time

Partitioning

Our queries are always filtered by:

1. Customer
2. Time

So we Partition by:

```
├── cid=X
|   ├── year=2015
|   └── year=2016
|         └── month=0
|               └── day=0
|                     └── hour=0
└── cid=Y
```

Partitioning

Our queries are always filtered by:

1. Customer
2. Time

So we Partition by:

```
├── cid=X
|   ├── year=2015
|   └── year=2016
|         └── month=0
|               └── day=0
|                     └── hour=0
└── cid=Y
```

Spark understands and translates query filters to this folder structure

Big Improvement!

Now a customer can query a time range quickly

Partitioning problems

Partitioning problems

Customers generally ask questions such as:

"Over the last 6 months, how many times did I see IP X using protocol Y?"

Partitioning problems

Customers generally ask questions such as:

"Over the last 6 months, how many times did I see IP X using protocol Y?"

"When did IP X not use port 80 for HTTP?"

Partitioning problems

Customers generally ask questions such as:

"Over the last 6 months, how many times did I see IP X using protocol Y?"

"When did IP X not use port 80 for HTTP?"

"Who keeps scanning server Z for open SSH ports?"

Partitioning problems

Customers generally ask questions such as:

"Over the last 6 months, how many times did I see IP X using protocol Y?"

"When did IP X not use port 80 for HTTP?"

"Who keeps scanning server Z for open SSH ports?"

Queries would take minutes.

Queries spanning large time windows

```
select count(*) from events where ip = '192.168.0.1' and cid = 1 and year = 2016
```

Queries spanning large time windows

```
select count(*) from events where ip = '192.168.0.1' and cid = 1 and year = 2016
```

```
├── cid=X
|   ├── year=2015
|   └── year=2016
|         |── month=0
|         |    └── day=0
|         |        └── hour=0
|         |            └── 192.168.0.1_was_NOT_here.parquet
|         └── month=1
|              └── day=0
|                  └── hour=0
|                      └── 192.168.0.1_WAS_HERE.parquet
└── cid=Y
```

Problem #1

- Requires listing out all the sub-dirs for large time ranges.
- Remember S3 is not really a file system
- **Slow!**

Problem #1

- Requires listing out all the sub-dirs for large time ranges.
- Remember S3 is not really a file system
- **Slow!**

Problem #2

- Pulling potentially thousands of files from S3.
- **Slow and Costly!**

Solving problem #1

Put partition info and file listings in a db ala Hive.

Solving problem #1

Put partition info and file listings in a db ala Hive.

Why not just use Hive?

Solving problem #1

Put partition info and file listings in a db ala Hive.

Why not just use Hive?

- Still not fast enough
- Also does not help with Problem #2

DSE/SOLR to the Rescue!

DSE/SOLR to the Rescue!

Store file meta data in SOLR

DSE/SOLR to the Rescue!

Store file meta data in SOLR

Efficiently skip elements of partition hierarchy!

```
select count(*) from events where month = 6
```

DSE/SOLR to the Rescue!

Store file meta data in SOLR

Efficiently skip elements of partition hierarchy!

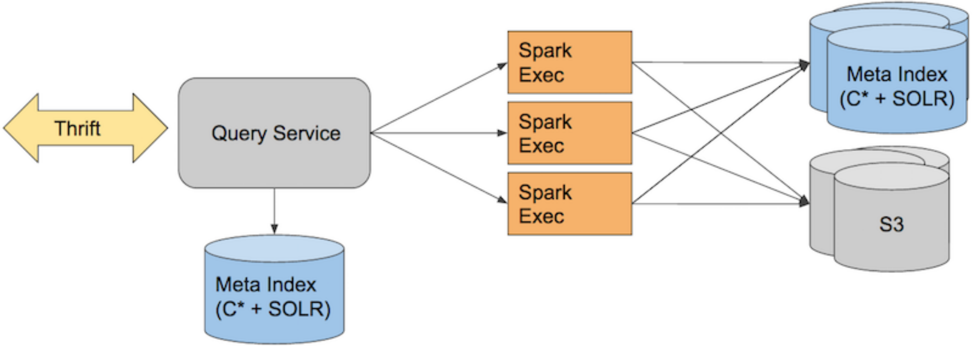
```
select count(*) from events where month = 6
```

Avoids pulling all meta in Spark driver

1. Get partition counts and schema info from SOLR driver-side
2. Submit SOLR RDD to cluster
3. Run `mapPartitions` on SOLR RDD and turn into Parquet RDDs

   As an optimization for small file sets we pull the SOLR rows driver side

Boxitecture



Performance gains!

Source	Scan/Filter Time
SOLR	< 100 milliseconds
Hive	> 5 seconds
S3 directory listing	> 5 ***minutes!!!***

Problem #1 Solved!

Problem #1 Solved!

What about Problem #2?

Solving problem #2

Still need to pull potentially thousands of files to answer our query!

Solving problem #2

Still need to pull potentially thousands of files to answer our query!

Can we partition differently?

Solving problem #2

Still need to pull potentially thousands of files to answer our query!

Can we partition differently?

Field	Cardinality	Result
Protocol	Medium (9000)	
Port	High (65535)	
IP Addresses	Astronomically High (3.4 undecillion)	

Nope! Nope! Nope!

Searching High Cardinality Data

Searching High Cardinality Data

Assumptions

1. Want to reduce # of files pulled for a given query
2. Cannot store all exact values in SOLR
3. We are okay with a few **false positives**

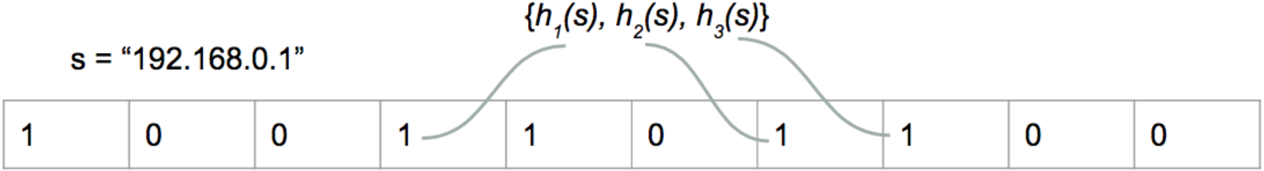
Searching High Cardinality Data

Assumptions

1. Want to reduce # of files pulled for a given query
2. Cannot store all exact values in SOLR
3. We are okay with a few **false positives**

This sounds like a job for...

Bloom Filters!



Towards a "Searchable" Bloom Filter

Towards a "Searchable" Bloom Filter

Normal SOLR index looks vaguely like

Term	Doc IDs
192.168.0.1	1,2,3,5,8,13…
10.0.0.1	2,4,6,8…
8.8.8.8	1,2,3,4,5,6

Towards a "Searchable" Bloom Filter

Normal SOLR index looks vaguely like

Term	Doc IDs
192.168.0.1	1,2,3,5,8,13...
10.0.0.1	2,4,6,8...
8.8.8.8	1,2,3,4,5,6

Terms are going to grow out of control

Towards a "Searchable" Bloom Filter

Normal SOLR index looks vaguely like

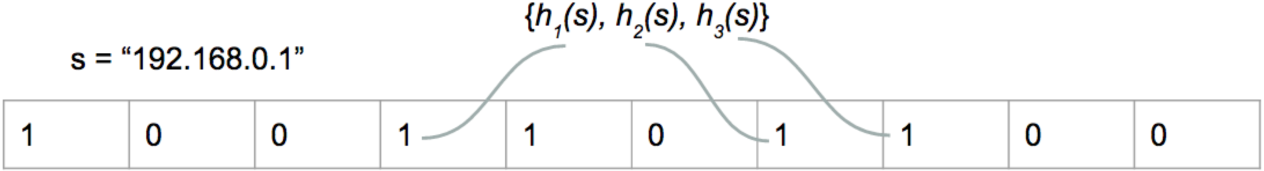
Term	Doc IDs
192.168.0.1	1,2,3,5,8,13...
10.0.0.1	2,4,6,8...
8.8.8.8	1,2,3,4,5,6

Terms are going to grow out of control

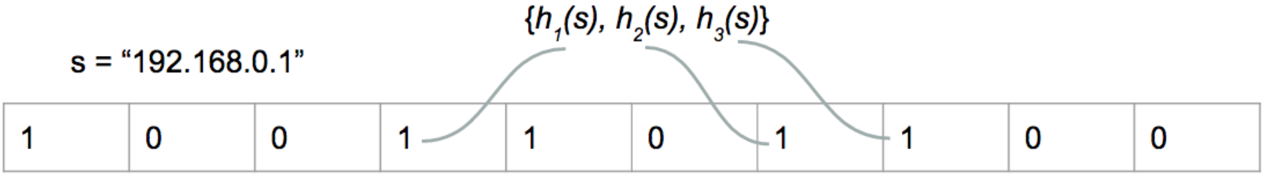
If only we could constrain to a reasonable number of values?

Terms as a "bloom filter"

Terms as a "bloom filter"

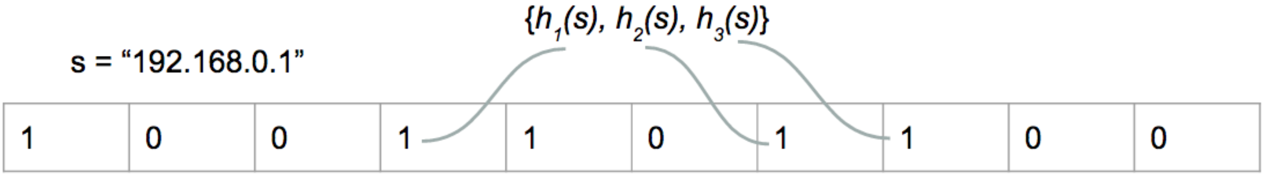


Terms as a "bloom filter"



What if our terms were the ***offsets*** of the Bloom Filter values?

Terms as a "bloom filter"



What if our terms were the ***offsets*** of the Bloom Filter values?

Term	Doc IDs
0	1,2,3,5,8,13…
1	2,4,6,8…
2	1,2,3,4,5,6
3	1,2,3
…	…
N	1,2,3,4,5…

Searchable Bloom Filters

Index

Term	Doc IDs
0	0,1,2
1	1,2
2	1
3	0
4	1,2
5	0

Indexing

Field	Value	Indexed Values	Doc ID
ip	192.168.0.1	{0, 3, 5}	0
ip	10.0.0.1	{1, 2, 4}	1
ip	8.8.8.8	{0, 1, 4}	2

Queries

Field	Query String	Actual Query
ip	ip:192.168.0.1	ip_bits:0 AND 3 AND 5
ip	ip:10.0.0.1	ip_bits:1 AND 4 AND 5

Problem #2 Solved!

Problem #2 Solved!

Enormous filtering power

Problem #2 Solved!

Enormous filtering power

Relatively minimal cost in space and computation

Key Lookups

Key Lookups

Need to retain this C* functionality

Key Lookups

Need to retain this C* functionality

Spark/Parquet has no direct support

- What partition would it choose?

- The partition would have to be encoded in the key?!

Key Lookups

Need to retain this C* functionality

Spark/Parquet has no direct support

What partition would it choose?

The partition would have to be encoded in the key?!

Solution:

- Our keys have time encoded in them
- Enables us to generate the partition path containing the row

Key Lookups

Need to retain this C* functionality

Spark/Parquet has no direct support

What partition would it choose?

The partition would have to be encoded in the key?!

Solution:

- Our keys have time encoded in them
- Enables us to generate the partition path containing the row

That was easy!

Other reasons to "customize"

Other reasons to "customize"

Parquet has support for filter pushdown

Other reasons to "customize"

Parquet has support for filter pushdown

Spark has support for Parquet filter pushdown, but...

Other reasons to "customize"

Parquet has support for filter pushdown

Spark has support for Parquet filter pushdown, but...

- Uses INT96 for Timestamp
  - No pushdown support: SPARK-11784
  - All our queries involve timestamps!

Other reasons to "customize"

Parquet has support for filter pushdown

Spark has support for Parquet filter pushdown, but...

- Uses INT96 for Timestamp
  - No pushdown support: SPARK-11784
  - All our queries involve timestamps!

- IP Addresses
  - Spark, Impala, Presto have no direct support
  - Use string or binary
  - Wanted to be able to push down CIDR range comparisons

Other reasons to "customize"

Parquet has support for filter pushdown

Spark has support for Parquet filter pushdown, but...

- Uses INT96 for Timestamp
  - No pushdown support: SPARK-11784
  - All our queries involve timestamps!
- IP Addresses
  - Spark, Impala, Presto have no direct support
  - Use string or binary
  - Wanted to be able to push down CIDR range comparisons

Lack of pushdown for these leads to wasted I/O and GC pressure.

Archiving

Archiving

Currently, when Time shard fills up:

1. Roll new hot time shard
2. Run Spark job to Archive data to S3
3. ~~Swap out "warm" shard for cold storage~~ (automagical)
4. Drop the "warm" shard

Archiving

Currently, when Time shard fills up:

1. Roll new hot time shard
2. Run Spark job to Archive data to S3
3. ~~Swap out "warm" shard for cold storage~~ (automagical)
4. Drop the "warm" shard

Not an ideal process, but deals with legacy requirements

Archiving

Currently, when Time shard fills up:

1. Roll new hot time shard
2. Run Spark job to Archive data to S3
3. ~~Swap out "warm" shard for cold storage~~ (automagical)
4. Drop the "warm" shard

Not an ideal process, but deals with legacy requirements

TODO:

1. Stream data straight to cold storage
2. Materialize customer edits in to hot storage
3. ~~Merge hot and cold data at query time~~ (already done)

What have we done so far?

What have we done so far?

1. Time sharded C* clusters with SOLR

What have we done so far?

1. Time sharded C* clusters with SOLR
2. Cheap speedy Cold storage based on S3 and Spark

What have we done so far?

1. Time sharded C* clusters with SOLR
2. Cheap speedy Cold storage based on S3 and Spark
3. A mechanism for archiving data to S3

That's cool, but…

That's cool, but...

How do we handle queries to 3 different stores?

- C*
- SOLR
- Spark

That's cool, but...

How do we handle queries to 3 different stores?

- C*
- SOLR
- Spark

Handle Timesharding and Functional Sharding?

That's cool, but...

How do we handle queries to 3 different stores?

- C*
- SOLR
- Spark

Handle Timesharding and Functional Sharding?

Lot's of Scala query DSL libraries:

- Quill
- Slick
- Phantom
- etc

Lot's of Scala query DSL libraries:

- Quill
- Slick
- Phantom
- etc

AFAIK nobody supports:

1. Simultaneously querying heterogeneous data stores

Lot's of Scala query DSL libraries:

- Quill
- Slick
- Phantom
- etc

AFAIK nobody supports:

1. Simultaneously querying heterogeneous data stores
2. Stitching together time series data from multiple stores

Lot's of Scala query DSL libraries:

- Quill
- Slick
- Phantom
- etc

AFAIK nobody supports:

1. Simultaneously querying heterogeneous data stores
2. Stitching together time series data from multiple stores
3. Managing sharding:
    - Configuration
    - Discovery

Enter Quaero:

Enter Quaero:

Abstracts away data store differences

- Query AST (Algebraic Data Type in Scala)
- Command/Strategy pattern for easily plugging in new data stores

Enter Quaero:

Abstracts away data store differences

- Query AST (Algebraic Data Type in Scala)
- Command/Strategy pattern for easily plugging in new data stores

Deep understanding of sharding patterns

- Handles merging of time/functional sharded data
- Adding shards is configuration driven

Enter Quaero:

Abstracts away data store differences

- Query AST (Algebraic Data Type in Scala)
- Command/Strategy pattern for easily plugging in new data stores

Deep understanding of sharding patterns

- Handles merging of time/functional sharded data
- Adding shards is configuration driven

Exposes a "typesafe" query DSL similar to Phantom or Rogue

- Reduce/eliminate bespoke code for retrieving the same data from all 3 stores

Open Sourcing? Maybe!?

There is still a lot of work to be done

We are hiring!

Interwebs: Careers @ Protectwise

Twitter: @ProtectWise